

JAI 2025

44^a Jornada de Atualização em Informática

Organizadores

Alexandre Paes dos Santos

Soraia Raupp Musse



CSBC

45° EDIÇÃO | MACEIÓ - AL | 2025



Organizadores

Soraia Raupp Musse
Alexandre Paes dos Santos

44ª Jornada de Atualização em Informática JAI 2025

Porto Alegre
Sociedade Brasileira de Computação – SBC
2025



Esta obra está sob a licença Creative Commons Atribuição 4.0 (CC-BY). Você pode redistribuir este livro em qualquer suporte ou formato e copiar, remixar, transformar e criar a partir do conteúdo deste livro para qualquer fim, desde que cite a fonte.

Dados Internacionais de Catalogação na Publicação (CIP)

C749 Congresso da Sociedade Brasileira de Computação (45.: 20 jul.-
24 jul. 2025 : Maceió)
44ª Jornada de Atualização em Informática (JAI 2025)
[recurso eletrônico] / organização: Soraia Raupp Musse,
Alexandre Paes dos Santos. Dados eletrônicos. - Porto Alegre:
Sociedade Brasileira de Computação, 2025.
205 p.; il.: PDF; 5,30 MB

Inclui bibliografia
ISBN 978-85-7669-648-3 (e-book)

1. Computação - Brasil - Congressos. 2. Informática. I.
Musse, Soraia Raupp. II. Santos, Alexandre Paes dos. III.
Sociedade Brasileira de Computação. IV. Título.

CDU 004(063)

Ficha catalográfica elaborada por Annie Casali - CRB-10/2339

Biblioteca Digital da SBC - SBC OpenLib



Sociedade Brasileira de Computação

Av. Bento Gonçalves, 9500
Setor 4 | Prédio 43.412 | Sala 219 | Bairro
Agronomia Caixa Postal 15012 | CEP 91501-970
Porto Alegre - RS
Fone: (51) 99252-
6018
sbc@sbcc.org.br

44ª Jornada de Atualização em Informática JAI 2025

Editora

Sociedade Brasileira de Computação (SBC)

Coordenação Geral do CSBC 2025

Elvys Soares (IFAL)

Raquel Cabral (UFAL)

Coordenação da JAI 2025

Soraia Raupp Musse (PUCRS)

Alexandre Paes dos Santos (UFAL)

Comitê de Programa da JAI 2025

Alba Cristina Magalhaes A. Melo (UnB)

Alexandre de Andrade Barbosa (UFAL)

Avelino Francisco Zorzo (PUCRS)

Cesar De Rose (PUCRS)

Douglas Afonso Tenório de Menezes (IFAL)

Luan Fonseca Garcia (PUCRS)

Lucas Silveira Kupssinskü (PUCRS)

Patrick Henrique da Silva Brito (UFAL)

Thiago Bruno Melo de Sales (UFAL)

Prefácio

É com prazer que apresentamos a edição de 2025 do livro da série Jornadas de Atualização em Informática. Os capítulos deste livro compõem o material de apoio aos minicursos selecionados para o evento e realizados como parte do Congresso da Sociedade Brasileira de Computação (CSBC). As JAI são de grande importância acadêmica, voltadas à atualização científica e tecnológica da comunidade brasileira de Computação.

Os minicursos desta edição foram selecionados e revisados por especialistas, cobrindo temas atuais e de alto impacto como as áreas de Segurança, LLMs, Inteligência Artificial na Educação e ainda um curso mais focado em implementação com Abstrações em Linguagens Contemporâneas. A seguir, os quatro capítulos que compõem este volume são brevemente apresentados.

- Capítulo 1 apresenta o protocolo Matrix, uma estrutura de comunicação e colaboração descentralizada que permite interoperabilidade entre plataformas. O capítulo aborda os fundamentos do protocolo, sua arquitetura, casos de uso e relevância no contexto atual de privacidade e descentralização.
- Capítulo 2 discute a incorporação de Inteligência Artificial na educação, abordando tendências, ferramentas e práticas pedagógicas mediadas por tecnologias inteligentes. O texto também debate os impactos e desafios do uso de IA no contexto educacional.
- Capítulo 3 traz uma introdução prática à programação *multithread*, cobrindo desde os conceitos básicos até técnicas de sincronização e paralelismo. O capítulo utiliza exemplos ilustrativos para demonstrar como desenvolver aplicações mais eficientes com múltiplas threads.

- Capítulo 4 explora a integração de modelos de LLM em aplicações práticas, especialmente no contexto de sistemas inteligentes e aprendizado de máquina. São discutidas estratégias de incorporação, treinamento e uso de modelos computacionais para melhorar a eficiência e a acurácia dos sistemas computacionais.

Agradecemos aos autores pelos textos de alta qualidade, e à comissão avaliadora pelo trabalho criterioso durante o processo de seleção, que visou escolher com base no interesse e inovação da área, na senioridade dos autores e também na regionalidade e diversidade das propostas recebidas. Esperamos que este material contribua significativamente para a formação e atualização de estudantes, professores e profissionais da área.

Soraia Raupp Musse

Alexandre Paes dos Santos

Coordenadores da JAI 2025

Sumário

1. Protocolo Matrix: Conceitos, Arquitetura, Aplicações e Desafios.....1

José A. P. Martins (UFC)
Paulo A. L. Rego (UFC)
José A. F. de Macêdo (UFC)
Rossana M. C. Andrade (UFC)
Michel S. Bonfim (UFC)
Roberto F. Ivo (UFC)
Vinícius L. R. da Costa (ABIN)
Rodrigo P. Pacheco (ABIN)
Francisco A. P. da Silva (UFPI)

2. Inteligência Artificial para a educação47

Laura O. Moraes (UNIRIO)
Carla A. D. M. Delgado (UFRJ)

3. Programação Multithread: Modelos e Abstrações em Linguagens Contemporâneas97

Gerson Geraldo H. Cavalheiro (UFPEL)
Alexandro Baldassin (Unesp)
André Rauber Du Bois (UFPEL)

4. Incorporação de Modelos de Linguagem em Larga Escala em Dispositivos Móveis: Otimização, Personalização e Desafios..... 147

Nicollas R. de Oliveira (UFF)
João Vitor V. Silva (UFF)
Guilherme N. N. Barbosa (UFF)
Dianne S. V. de Medeiros (UFF)
Diogo M. F. Mattos (UFF)

Capítulo

1

Protocolo Matrix: Conceitos, Arquitetura, Aplicações e Desafios

José A. P. Martins, Paulo A. L. Rego, José A. F. de Macêdo, Rossana M. C. Andrade, Michel S. Bonfim, Roberto F. Ivo, Vinícius L. R. da Costa, Rodrigo P. Pacheco e Francisco A. P. da Silva

Abstract

This chapter presents a technical and applied analysis of the Matrix protocol, an open solution for secure and federated digital communication. It begins by discussing the context of digital sovereignty and the limitations of centralized platforms. The architecture of the protocol, its key features, and practical applications using the Element client are then explored. The chapter also covers the deployment of private Matrix servers with Synapse, automation via APIs, and the development of custom bots. Finally, it outlines current technical challenges and research opportunities related to interoperability, performance, and distributed security. The content integrates theoretical foundations with hands-on guidance, providing a comprehensive basis for institutional adoption of Matrix.

Resumo

Este capítulo apresenta uma análise técnico-aplicada do protocolo Matrix, uma solução aberta para comunicação segura e federada. Inicialmente, discute-se o contexto da soberania digital e os limites de plataformas centralizadas. Em seguida, são exploradas a arquitetura do protocolo, suas principais funcionalidades e aplicações práticas por meio do cliente Element. O texto também aborda a implantação de servidores privados com Synapse, automações via APIs e desenvolvimento de bots. Por fim, são discutidos desafios técnicos e oportunidades de pesquisa, com foco em interoperabilidade, desempenho e segurança distribuída. O conteúdo combina fundamentos teóricos com orientações práticas, oferecendo uma base sólida para adoção institucional do Matrix.

1.1. Introdução

Nos últimos anos, o debate sobre soberania digital tem ganhado relevância em diversas esferas, impulsionado por preocupações com privacidade, dependência tecnológica e concentração de poder em plataformas digitais. A comunicação, como componente central

das infraestruturas digitais, passou a depender majoritariamente de sistemas fechados, centralizados e de difícil auditoria [Floridi 2020]. Essa configuração favorece assimetrias de controle, limita a autonomia de organizações públicas e privadas, e dificulta o alinhamento tecnológico a valores locais e institucionais.

Diante disso, surgem como uma das alternativas os sistemas federados e o uso de *open-source software* (OSS) como pilares técnicos e estratégicos. Essas abordagens promovem a descentralização do controle, aumentam a auditabilidade das infraestruturas e ampliam a capacidade de adaptação institucional. A soberania digital, nesse contexto, é compreendida como a capacidade de indivíduos, organizações e Estados de exercerem controle sobre dados e sobre os sistemas que os armazenam e transportam [Klare and Lechner 2023, Moerel and Timmers 2021].

Apesar do avanço dessas discussões, persiste uma dependência estrutural de soluções proprietárias na comunicação digital cotidiana. Ferramentas como WhatsApp, Microsoft Teams ou Telegram não oferecem suporte à hospedagem própria, nem garantias de interoperabilidade¹ ou de segurança auditável. Esse cenário impõe obstáculos à adoção de políticas públicas baseadas em independência tecnológica e proteção de dados sensíveis.

Como resposta a esse problema, o presente capítulo examina o protocolo *Matrix* como um caminho viável e estratégico para comunicação digital federada, segura e auditável. O Matrix é um protocolo aberto para comunicação em tempo real, que adota uma arquitetura federada e descentralizada, permitindo que múltiplas instâncias se comuniquem entre si sem depender de servidores centralizados. [Foundation 2025].

Este capítulo propõe uma abordagem técnico-analítica sobre o protocolo, articulando seus fundamentos arquiteturais, aplicações práticas e potenciais para fortalecer a soberania digital em contextos institucionais e governamentais. O objetivo é apresentar o Matrix não apenas como ferramenta tecnológica, mas como recurso estratégico na formulação de soluções autônomas e auditáveis de comunicação.

O capítulo está estruturado em sete seções. A **Seção 1.1** introduz o tema, expõe o problema e apresenta o escopo do minicurso. As seções seguintes abordam a arquitetura do protocolo (**Seção 1.2**), aplicações práticas (**Seção 1.3**), aspectos avançados em servidores (**Seção 1.4**), uso das APIs para automação (**Seção 1.5**), desafios atuais e frentes de pesquisa (**Seção 1.6**). Finalmente, apresentamos as conclusões na **Seção 1.7**.

1.1.1. Fundamentos e Conceitos Principais

A compreensão do protocolo *Matrix* exige a familiarização com alguns fundamentos que estruturam a comunicação digital descentralizada e o desenvolvimento de software de código aberto. Esses elementos ajudam a situar o Matrix em um ecossistema mais amplo de soluções técnicas orientadas à transparência, autonomia e interoperabilidade.

A comunicação digital, desde seus primórdios, oscilou entre modelos centralizados e descentralizados. Nas arquiteturas centralizadas, amplamente adotadas por plataformas comerciais, a mediação das trocas informacionais depende de um servidor único que controla autenticação, roteamento, armazenamento e acesso. Esse modelo favorece

¹Capacidade de sistemas distintos se comunicarem e funcionarem em conjunto de forma transparente e eficiente, sem necessidade de adaptações específicas.

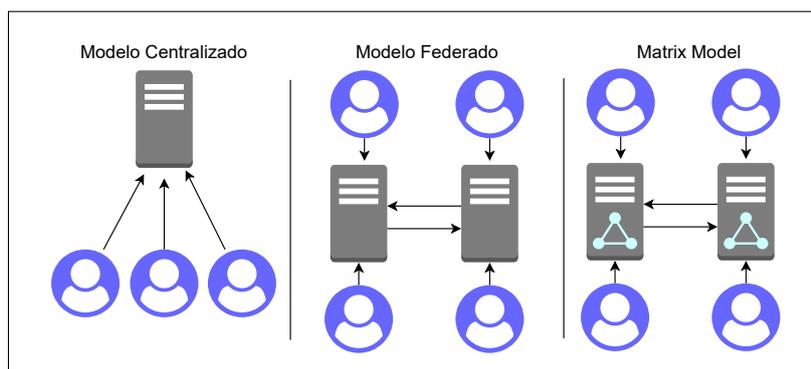


Figura 1.1. Comparação entre modelos de comunicação digital.

desempenho e controle operacional, mas concentra poder e limita a auditabilidade. Em contraste, sistemas descentralizados distribuem essas funções entre múltiplas entidades ou nós, mitigando pontos únicos de falha e ampliando a resiliência e o controle sobre os dados [Moerel and Timmers 2021, Floridi 2020].

Entre os modelos descentralizados, destacam-se as redes federadas, nas quais servidores distintos podem operar de forma autônoma e ainda assim interagir entre si com base em padrões abertos.² Esse modelo é amplamente adotado em sistemas como o e-mail e, mais recentemente, em protocolos como o ActivityPub, que sustenta redes sociais distribuídas como o Mastodon. A federação permite que diferentes organizações mantenham sua própria infraestrutura, com regras locais de controle e privacidade, sem abdicar da conectividade entre domínios distintos.

É nesse espectro que se insere o protocolo *Matrix*, projetado para viabilizar comunicação em tempo real entre múltiplas instâncias federadas, com foco em mensagens, chamadas e troca de arquivos. O Matrix busca aliar a flexibilidade de um modelo federado com o rigor técnico necessário para comunicação segura e verificável. A proposta original do protocolo surgiu a partir da constatação de que, embora existissem diversas soluções de mensageria, poucas ofereciam um padrão interoperável, aberto e extensível capaz de sustentar comunicação em tempo real entre domínios distintos [Li et al. 2023, Foundation 2025].

Um dos pilares do projeto é o compromisso com a transparência, refletido na publicação aberta de suas especificações e no desenvolvimento colaborativo de suas implementações. A abertura do código-fonte, aliada a práticas de auditoria e revisão comunitária, permite identificar falhas, adaptar componentes e estender funcionalidades. Em ambientes institucionais, essa característica torna-se particularmente relevante ao permitir que políticas de segurança e privacidade sejam adaptadas às necessidades locais [Bechara and Lechner 2024, Biström et al. 2024].

Ao articular descentralização, federação e software livre, o protocolo Matrix oferece um exemplo consistente de como infraestruturas de comunicação podem ser projeta-

²Federação é o modelo em que múltiplas instâncias de um sistema operam de forma independente, mas interoperam entre si segundo protocolos compartilhados. Exemplos incluem o e-mail, o XMPP e o ActivityPub.

das com base em princípios de soberania digital, auditabilidade e controle distribuído.

1.1.2. Características Essenciais do Protocolo

Diante da crescente preocupação com a privacidade, segurança e autonomia digital, torna-se fundamental analisar as características das plataformas de mensageria utilizadas por instituições públicas e privadas. A escolha da tecnologia impacta diretamente o controle sobre os dados e a infraestrutura de comunicação. Nesse cenário, critérios como abertura, criptografia e interoperabilidade ganham relevância estratégica. A Tabela 1.1 apresenta um comparativo entre plataformas populares de mensageria, destacando critérios relevantes no contexto da soberania digital: abertura do código-fonte, suporte à criptografia ponta-a-ponta (E2EE), interoperabilidade e possibilidade de hospedagem própria. Esses aspectos são particularmente sensíveis em ambientes institucionais que buscam controle sobre infraestrutura e dados. Observa-se que o Matrix se diferencia por combinar todos esses atributos em uma única solução, enquanto outras plataformas, como WhatsApp e Slack, mantêm modelos centralizados e de código fechado. O Telegram oferece abertura parcial, mas sem suporte nativo à hospedagem autônoma ou interoperabilidade plena.

Tabela 1.1. Comparativo entre plataformas de mensageria

Plataforma	Código Aberto	E2EE	Interoperável	Hospedagem Própria
Matrix (Element)	Sim	Sim	Sim (via bridges)	Sim
WhatsApp	Não	Sim	Não	Não
Telegram	Parcial	Opcional	Limitada	Não
Signal	Sim	Sim	Não	Sim
Slack (padrão)	Não	Não	Limitada (via API)	Não

O protocolo *Matrix* apresenta um conjunto de características técnicas que o distinguem de outras soluções de comunicação digital. Essas propriedades resultam de escolhas arquiteturais voltadas à interoperabilidade, controle descentralizado, segurança por padrão e extensibilidade. Ainda que a estrutura interna do protocolo seja explorada na seção seguinte, algumas de suas funcionalidades podem ser compreendidas a partir de seu comportamento observável e das possibilidades que oferece a usuários e desenvolvedores.

A interoperabilidade é um dos pilares do Matrix. O protocolo permite a construção de *bridges*, que funcionam como conectores entre o ecossistema Matrix e outras plataformas de comunicação, como Slack, Discord, IRC ou Microsoft Teams. Essas pontes facilitam a transição entre sistemas legados e ambientes federados, promovendo inclusão e coexistência entre diferentes tecnologias [Cui et al. 2023, Foundation 2025].

Outra característica distintiva é a possibilidade de controle efetivo sobre dados e infraestrutura. Ao permitir a hospedagem autônoma de servidores, o Matrix viabiliza que organizações e comunidades mantenham sua comunicação sob políticas locais de segurança, retenção e governança. Essa capacidade se alinha aos princípios da soberania digital e da privacidade institucional, frequentemente negligenciados em soluções centralizadas [Bechara and Lechner 2024, Moerel and Timmers 2021].

A segurança é incorporada ao protocolo desde sua concepção. A criptografia ponta-a-ponta (E2EE) é suportada por padrão em salas privadas, com mecanismos adi-

cionais como *cross-signing* de dispositivos e *key backup* criptografado, permitindo autenticação mútua e recuperação segura de chaves. Esses recursos ampliam a proteção sem comprometer a usabilidade, e posicionam o Matrix como uma alternativa robusta para contextos sensíveis [Albrecht et al. 2023, Authority 2024].

A resiliência do protocolo também é notável. Por operar em um modelo federado e assíncrono, o sistema consegue manter a continuidade da comunicação mesmo em casos de falha ou indisponibilidade temporária de servidores individuais. Essa propriedade é essencial para aplicações que exigem tolerância a falhas e continuidade operacional, como ambientes críticos e redes comunitárias distribuídas.

Finalmente, o Matrix adota especificações públicas e APIs bem documentadas, facilitando sua integração com outras ferramentas e o desenvolvimento de aplicações sob medida. Essa extensibilidade tem viabilizado o surgimento de clientes alternativos, bots, sistemas de autenticação customizados e integrações com sistemas corporativos e educacionais [matrix nio 2025, Rahman and Wang 2024].

Ao combinar interoperabilidade, controle local, segurança nativa e abertura à inovação, o protocolo Matrix apresenta um conjunto coeso de funcionalidades que o tornam adequado a contextos onde a autonomia tecnológica é prioridade.

1.1.3. Soberania Digital e Privacidade

A soberania digital refere-se à capacidade de Estados, organizações e indivíduos de exercerem controle efetivo sobre seus dados, sistemas e infraestruturas tecnológicas. Essa noção ganha relevância diante da crescente interdependência entre decisões políticas e arquiteturas técnicas, especialmente em áreas como segurança nacional, privacidade e autonomia tecnológica [Floridi 2020, Klare and Lechner 2023]. A centralização de serviços digitais sob o controle de grandes empresas transnacionais limita a autodeterminação dos atores locais e expõe vulnerabilidades críticas associadas à vigilância, manipulação de dados e dependência estratégica.

Ainda, a soberania digital deve ser entendida não apenas como uma questão estratégica, mas como um imperativo ético e jurídico em defesa da privacidade, da democracia e dos direitos digitais do cidadão. Além da segurança nacional, a ausência da soberania digital compromete direitos fundamentais, especialmente em contextos de conflitos externos, nos quais nações estrangeiras possam interferir diretamente nos serviços oferecidos. Portanto, torna-se imperativo que Estados e instituições públicas incorporem não apenas soluções técnicas descentralizadas, mas também políticas robustas de fomento à pesquisa, desenvolvimento local e cooperação internacional voltada à construção de uma infraestrutura digital verdadeiramente soberana e democrática.

Nesse sentido, o uso de software livre e arquiteturas federadas surge como base tecnológica para a construção de alternativas soberanas. O *open-source software* (OSS) viabiliza auditoria, adaptação e apropriação institucional dos sistemas, enquanto redes federadas permitem a descentralização operacional, mantendo a interoperabilidade entre entidades autônomas. Essa combinação amplia a capacidade de instituições públicas e comunitárias de construir soluções próprias sem renunciar à conectividade global [Moerel and Timmers 2021, Biström et al. 2024].

O protocolo *Matrix* ilustra esse modelo. Por sua natureza federada, permite que órgãos públicos e organizações mantenham controle sobre servidores, políticas de segurança e dados sensíveis. Um exemplo emblemático é o *Tchap*, sistema de mensageria utilizado pelo governo francês e desenvolvido sobre o Matrix. Essa iniciativa substituiu plataformas comerciais em comunicações oficiais, garantindo criptografia ponta a ponta e hospedagem soberana sob infraestrutura estatal [Government 2020, Bechara and Lechner 2024]. Outras experiências institucionais reforçam o papel estratégico de soluções abertas e federadas como ferramentas de independência tecnológica.

Apesar de seus avanços, a adoção de soluções soberanas enfrenta obstáculos relevantes. Barreiras técnicas, como a complexidade de implantação e manutenção de infraestruturas próprias, coexistem com desafios geopolíticos, incluindo a influência de padrões dominantes impostos por corporações e a dependência de componentes críticos produzidos por terceiros. Riscos como a vigilância extraterritorial e a presença de *backdoors* em sistemas proprietários mantêm a urgência de alternativas auditáveis e adaptáveis [Misra et al. 2025].

Ao integrar princípios técnicos de descentralização com requisitos políticos de autonomia e segurança, o Matrix se consolida como um dos caminhos possíveis para o fortalecimento da soberania digital em ambientes institucionais e governamentais.

1.1.4. Evolução do protocolo Matrix e eventos da comunidade

Desde sua introdução em 2014, o protocolo Matrix tem passado por uma evolução constante, marcada por melhorias estruturais, expansão da comunidade e consolidação de padrões formais por meio da Matrix Specification (*spec*)³. Essa trajetória técnica foi acompanhada pelo fortalecimento da governança aberta e do ecossistema de desenvolvedores e usuários ao redor do protocolo.

Tabela 1.2. Linha do tempo de desenvolvimento do protocolo Matrix

Ano	Evento / Marco
2014	Publicação inicial da spec Matrix 1.0 e criação do Synapse como implementação de referência.
2017	Introdução das primeiras MSCs (Matrix Spec Changes) ⁴ como processo formal de melhoria da especificação.
2019	Lançamento oficial da Matrix Foundation ; Element (antigo Riot) se torna cliente principal.
2021	Introdução de <i>Spaces</i> (hierarquias de salas) e suporte nativo a <i>Threads</i> .
2022	Proposta de Sliding Sync e melhoria drástica na sincronização de clientes móveis.
2023	Lançamento do Element Call com chamadas de áudio/vídeo nativamente federadas via WebRTC.
2024	Consolidação da visão Matrix 2.0 , com E2EE por padrão, sincronização rápida e governança aberta de spec.

A Tabela 1.2 apresenta uma síntese da linha do tempo com os marcos principais de desenvolvimento do protocolo. Além disso, evidencia não apenas a continuidade no

³<https://spec.matrix.org>

desenvolvimento técnico do protocolo, mas também a consolidação de uma cultura de padronização aberta e evolução incremental. Cada marco representa um avanço específico — desde a estruturação inicial da *spec*, passando pela formalização das MSCs, até a adoção de novos paradigmas de usabilidade e desempenho, como os introduzidos pelo Matrix 2.0. Essa trajetória demonstra o compromisso da comunidade em equilibrar inovação com interoperabilidade e retrocompatibilidade, aspectos essenciais para a adoção institucional de longo prazo.

O desenvolvimento e a disseminação do protocolo também são impulsionados por uma comunidade ativa, articulada em diversos eventos e canais de comunicação. A *Matrix Community Summit*⁵, realizada anualmente na Alemanha, é o principal ponto de encontro técnico e institucional da comunidade. Nela, desenvolvedores, pesquisadores e representantes de instituições compartilham avanços, experimentos e casos de adoção. O canal semanal *Matrix Live*⁶ oferece atualizações constantes sobre o ecossistema, incluindo entrevistas com autores de MSCs, novidades em clientes, bridges e projetos institucionais. Outros espaços relevantes incluem apresentações na *FOSDEM*⁷ e no *Chaos Communication Congress (CCC)*⁸, onde o protocolo tem sido debatido como alternativa aberta e soberana para mensageria segura.

O marco mais recente dessa trajetória é o **Matrix 2.0**. Trata-se de uma consolidação conceitual que incorpora avanços recentes como a sincronização ultrarrápida (*Sliding Sync*), a ativação da criptografia ponta-a-ponta (E2EE) por padrão, a integração de chamadas nativas via *Element Call*, e a reestruturação das funcionalidades de salas e espaços. Além de melhorias técnicas, o Matrix 2.0 representa um compromisso com a estabilidade, interoperabilidade e usabilidade, mantendo compatibilidade retroativa com versões anteriores. Esse momento é interpretado pela comunidade como a transição do protocolo de uma base experimental para uma solução madura, pronta para adoções institucionais e em escala.

1.2. Arquitetura e Algoritmos do Matrix

Nesta seção, descrevemos os principais componentes técnicos do protocolo Matrix. São abordadas suas estruturas arquitetônicas, mecanismos de sincronização distribuída, soluções de criptografia, interoperabilidade com outras plataformas e suas APIs fundamentais. O objetivo é oferecer uma visão sistematizada da infraestrutura que sustenta a comunicação federada, segura e extensível no contexto do Matrix.

1.2.1. Arquitetura Descentralizada e Federada

O protocolo Matrix adota uma arquitetura federada e descentralizada como resposta aos desafios impostos por sistemas de comunicação centralizados, que frequentemente sofrem com problemas de escalabilidade, pontos únicos de falha e limitações de controle sobre dados. Na arquitetura federada do Matrix, a responsabilidade pelo armazenamento, processamento e controle dos dados é distribuída entre múltiplos servidores in-

⁵<https://matrix.org/events>

⁶<https://matrix.org/blog/tags/matrix-live>

⁷<https://fosdem.org>

⁸<https://events.ccc.de>

dependentes, denominados homeservers⁹. Cada homeserver é operado de forma autônoma, mantendo a propriedade e o controle dos dados dos usuários que lhe são associados. Essa abordagem elimina a dependência de uma autoridade central, oferecendo maior resiliência, autonomia institucional e controle sobre os dados trafegados na rede [Li et al. 2023, Foundation 2025].

A arquitetura do protocolo Matrix é estruturada em torno de três componentes principais que interagem para viabilizar a comunicação descentralizada, interoperável e segura: os homeservers, os clientes e os app services.

Os homeservers constituem o núcleo da arquitetura. Eles são responsáveis por armazenar o estado dos usuários e das salas (*rooms*), gerenciar a autenticação e o processamento de eventos, além de coordenar a comunicação com outros homeservers dentro da federação. Cada homeserver mantém o histórico completo das conversas e o estado das salas associadas aos seus usuários, atuando como o guardião do domínio e das identidades que lhe pertencem. Essa autonomia permite que cada servidor opere de forma independente, ao mesmo tempo, em que participa da rede federada por meio de APIs RESTful, que asseguram a replicação do estado das salas e a propagação segura e confiável das mensagens entre diferentes domínios.

Em termos de infraestrutura, os homeservers são serviços backend que expõem interfaces HTTP e utilizam bancos de dados persistentes, geralmente relacionais (como PostgreSQL), para armazenar dados e garantir consistência eventual. Implementações populares de homeservers, como o Synapse (escrito em Python) [Matrix Foundation 2025b], Dendrite (em Go) [Matrix Foundation 2025a] e Conduit (em Rust) [Conduit 2025], exemplificam as abordagens técnicas para lidar com escalabilidade, persistência e federação. A Tabela 1.3 apresenta uma comparação das principais características dessas implementações.

Tabela 1.3. Comparação de Implementações de Homeserver Matrix

Implementação	Linguagem	Banco de Dados	Escalabilidade	Federação	Aberto
Synapse	Python	PostgreSQL	Moderada	Sim	Sim
Dendrite	Go	PostgreSQL/MySQL	Alta	Sim	Sim
Conduit	Rust	SQLite/PostgreSQL	Alta	Sim	Sim

Os clientes são as interfaces utilizadas pelos usuários finais para interagir com a rede Matrix. Eles incluem aplicativos para dispositivos móveis, desktops e navegadores web, tais como Element Web, Element X, FluffyChat e outros. Cada cliente está vinculado a um homeserver específico, com o qual interage através da Client-Server API para realizar operações essenciais como autenticação, envio de mensagens, gerenciamento de salas e sincronização de eventos. Essa comunicação assegura que o estado apresentado ao usuário esteja sempre atualizado e consistente com o servidor. Além disso, os clientes frequentemente implementam criptografia de ponta a ponta, utilizando os protocolos Olm e Megolm, fornecendo confidencialidade e integridade às comunicações. A Tabela 1.4 oferece uma comparação dos principais clientes Matrix, destacando plataformas suportadas, recursos de segurança e modelos de código aberto.

⁹Homeservers são instâncias autônomas do protocolo Matrix, responsáveis por gerenciar usuários, salas e eventos sob seu domínio.

Tabela 1.4. Comparação dos Principais Clientes Matrix

Cliente	Plataforma	E2EE	Aberto	Funcionalidades Notáveis
Element	Web, Desktop, Mobile	Sim	Sim	Interface rica, suporte a VoIP, E2EE
Element X	Mobile (iOS/Android)	Sim	Sim	Versão mais leve, design moderno
FluffyChat	Mobile (iOS/Android)	Sim	Sim	Interface simples, foco em usabilidade
Quaternion	Desktop (Linux/Win)	Sim	Sim	Cliente leve, interface Qt, ideal para desktop
Nheko	Desktop (Linux/Win/macOS)	Sim	Sim	Cliente desktop moderno, suporte a E2EE
Neo	Desktop (Windows)	Sim	Sim	Cliente Windows focado em usabilidade e recursos
SchildiChat	Mobile (Android)	Sim	Sim	Fork do Element focado em privacidade e performance
Mirage	Mobile (iOS)	Sim	Sim	Cliente iOS leve com foco em simplicidade

Paralelamente, a comunicação entre diferentes homeservers é realizada pela Federation API, que mantém a consistência do estado das salas e permite a interoperabilidade entre usuários pertencentes a domínios distintos. A federação é essencial para a arquitetura descentralizada do Matrix, permitindo que servidores independentes compartilhem eventos de forma segura e confiável.

Os *app services* representam componentes externos integrados ao ecossistema Matrix que expandem suas funcionalidades e capacidades. Esses serviços podem atuar como *bridges*, que são responsáveis por conectar a rede Matrix a outras plataformas de comunicação, como IRC, Slack, Discord e XMPP. *Bridges* realizam a tradução bidirecional de mensagens e eventos entre Matrix e os protocolos externos, permitindo a interoperabilidade entre diferentes ecossistemas de comunicação e ampliando o alcance da rede Matrix. Além das *bridges*, os *app services* englobam *bots* e outros agentes programáveis que automatizam tarefas, respondem a comandos, moderam salas ou fornecem funcionalidades adicionais. Esses *bots* podem interagir utilizando a API Client ou operar como *app services*, escutando eventos e respondendo conforme a lógica implementada, como é o caso do `mautrix-python`, uma biblioteca que facilita a criação de *bots* e *bridges* [Mautrix 2025]. Essa arquitetura aberta e extensível possibilita uma variedade de integrações e personalizações que enriquecem a experiência dos usuários e ampliam o potencial do protocolo.

A Figura 1.2 ilustra os principais componentes do ecossistema Matrix e suas interações fundamentais. O diagrama destaca a relação entre os **clientes Matrix** (1), os **homeservers** responsáveis pela mediação e armazenamento das comunicações (2), e os **servidores de identidade** (3), que auxiliam na descoberta e verificação de usuários. Também são representadas as **bridges** (4), que atuam como conectores entre o Matrix e outras plataformas de comunicação, possibilitando a interoperabilidade com **clientes externos** (5) como Slack e Skype.

Ao adotar esse modelo, o Matrix possibilita que organizações públicas, instituições acadêmicas, comunidades ou indivíduos mantenham suas próprias instâncias do pro-

protocolo, com regras locais de autenticação, retenção e segurança ¹⁰ A arquitetura federada torna possível que essas instâncias colaborem entre si sem abrir mão de suas políticas de governança e infraestrutura própria, alinhando-se aos princípios de soberania digital.

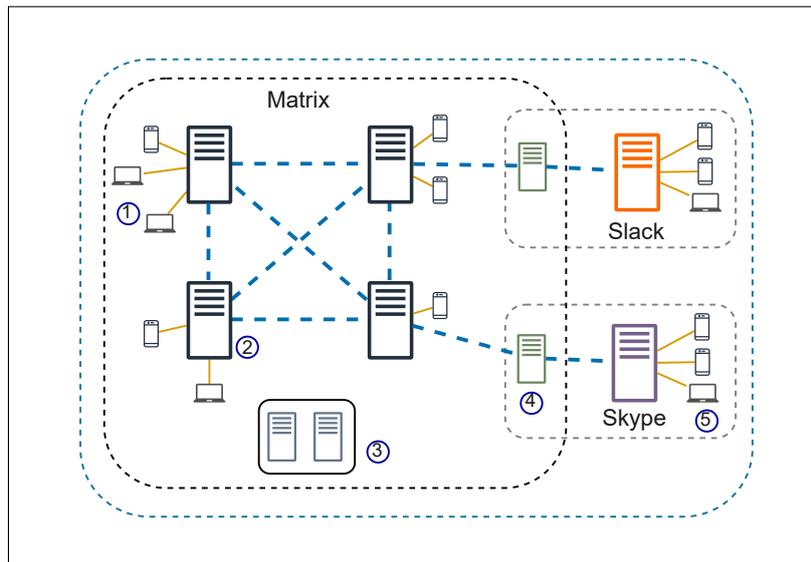


Figura 1.2. Diagrama da arquitetura geral do Matrix: clientes, homeservers e bridges.

1.2.2. Modelo de Dados e Eventos

No cerne da arquitetura do Matrix está o modelo de dados orientado a eventos, que fundamenta toda a comunicação e sincronização dentro do protocolo. Cada ação realizada no sistema — seja o envio de uma mensagem, uma alteração na configuração da sala, ou um evento de presença — é representada como um evento codificado em JSON. Esses eventos são a unidade atômica da comunicação, encapsulando tanto os dados da ação quanto metadados essenciais, como o identificador do remetente, o timestamp, e a relação com eventos anteriores.

O protocolo Matrix define uma variedade de tipos de eventos para suportar diferentes funcionalidades e tipos de mídia. Entre os tipos mais comuns estão:

- Mensagem de texto (`m.room.message`): Eventos que carregam mensagens em texto simples, sendo o tipo mais frequente nas conversas.
- Envio de arquivos (`m.room.message` com `msgtype` como `m.file` ou `m.image`): Eventos que transportam metadados sobre arquivos, imagens ou vídeos compartilhados, incluindo URLs para acesso e informações de tamanho.
- Áudio e Vídeo (`m.room.message` com `msgtype` como `m.audio` ou `m.video`): Eventos similares ao envio de arquivos, mas para conteúdo multimídia de áudio e vídeo, suportando mensagens de voz, clipes e chamadas.

¹⁰A capacidade de impor políticas locais torna o Matrix adequado a ambientes com requisitos normativos específicos, como órgãos públicos e instituições educacionais.

- Chamadas de voz e vídeo (`m.call.invite`, `m.call.answer`, `m.call.hangup`): Eventos específicos que coordenam a sinalização para iniciar, aceitar ou encerrar chamadas em tempo real, permitindo comunicação multimídia interativa entre usuários.
- Alteração do nome da sala (`m.room.name`): Eventos que definem ou atualizam o nome de uma sala.
- Atualização do tópico da sala (`m.room.topic`): Eventos responsáveis por definir ou modificar o tópico de uma sala, que normalmente serve como descrição ou assunto principal.
- Gerenciamento de membros (`m.room.member`): Eventos usados para adicionar, remover ou modificar permissões de usuários dentro de uma sala.
- Controle de permissões (`m.room.power_levels`): Eventos que definem os níveis de permissão dos membros, determinando quem pode enviar certos tipos de eventos ou alterar configurações da sala.
- Criação da sala (`m.room.create`): Evento inicial que sinaliza a criação da sala e contém informações sobre seu criador e propriedades básicas.
- Estado de presença (`m.presence`): Eventos que informam o estado de presença dos usuários, como online, offline ou ausente.
- Indicador de digitação (`m.typing`): Eventos temporários que sinalizam que um usuário está digitando em uma sala.
- Reações (`m.reaction`): Eventos que registram reações, como emojis, a mensagens existentes.
- Envio de figurinhas (`m.sticker`): Eventos que permitem o envio de figurinhas (stickers) nas conversas.
- Alteração do avatar da sala (`m.room.avatar`): Eventos que definem ou modificam o avatar associado a uma sala.

Os Listings 1.2 e 1.1 apresentam exemplos de eventos no protocolo Matrix. O Listing 1.1 ilustra a entrada do usuário Maria em uma sala, representada por um evento de alteração de membro, enquanto o Listing 1.2 mostra um evento de envio de mensagem de texto. Ambos os eventos compartilham campos essenciais que fundamentam a estrutura do protocolo: o campo `type` indica o tipo do evento, determinando sua função e interpretação; o campo `sender` identifica o usuário que originou o evento; `room_id` especifica a sala em que o evento ocorre; e `event_id` é um identificador único para cada evento. O campo `origin_server_ts` representa o timestamp do evento no servidor de origem, usado para ordenação e auditoria. Em eventos de estado, como o de membro, o campo `state_key` identifica o usuário ao qual o estado se aplica, e o conteúdo (`content`) detalha a ação ou dado associado, como o corpo da mensagem ou o tipo de participação na sala.

```

1 {
2   "type": "m.room.member",
3   "sender": "@maria:example.org",
4   "state_key": "@maria:example.org",
5   "room_id": "!reuniao12345:example.org",
6   "event_id": "$entryevent12345:example.org",
7   "origin_server_ts": 1685100000000,
8   "content": {
9     "membership": "join"
10  }
11 }

```

Listing 1.1. Exemplo de evento representando a entrada do usuário @maria na sala "reuniao" do homeserver example.org

```

1 {
2   "type": "m.room.message",
3   "sender": "@maria:example.org",
4   "room_id": "!reuniao12345:example.org",
5   "event_id": "$sendevent56789:example.org",
6   "origin_server_ts": 1685000010000,
7   "content": {
8     "msgtype": "m.text",
9     "body": "Oi!"
10  }
11 }

```

Listing 1.2. Exemplo simplificado de evento de envio de mensagem

1.2.3. Matrix Event Graph (MEG): Algoritmos e Sincronização Distribuída

O mecanismo central de sincronização e consistência no protocolo Matrix é o *Matrix Event Graph* (MEG), uma estrutura baseada em grafos direcionados acíclicos (DAG)¹¹. Cada evento gerado por um cliente é representado como um nó no grafo, contendo referências a eventos anteriores. Essa estrutura permite a replicação assíncrona do histórico entre servidores, garantindo que múltiplas instâncias mantenham uma visão consistente dos dados mesmo em presença de desconexões temporárias [Jacob et al. 2021, Foundation 2025].

Cada homeserver mantém uma cópia parcial do MEG para as salas em que participa. Ao receber eventos de outros servidores, o homeserver executa algoritmos de fusão¹², validação de assinaturas, verificação de integridade e resolução de conflitos. A ordenação dos eventos se baseia em vetores causais, utilizando o conceito de `depth` e `prev_events` para garantir uma linearização compatível entre instâncias federadas. A

¹¹DAGs (Directed Acyclic Graphs) são estruturas comuns em sistemas distribuídos para representar relações causais sem ciclos.

¹²No contexto do MEG, a fusão de grafos refere-se à incorporação de eventos recebidos de outras réplicas no DAG local, preservando relações de causalidade.

sincronização se dá de forma eventual¹³, priorizando consistência sobre disponibilidade imediata.

A Figura 1.3 ilustra a evolução do MEG em duas réplicas distintas de um mesmo ambiente federado. Cada novo evento (vértice) é anexado às extremidades¹⁴ do grafo local, e a sincronização entre réplicas é feita por meio da troca dessas atualizações.

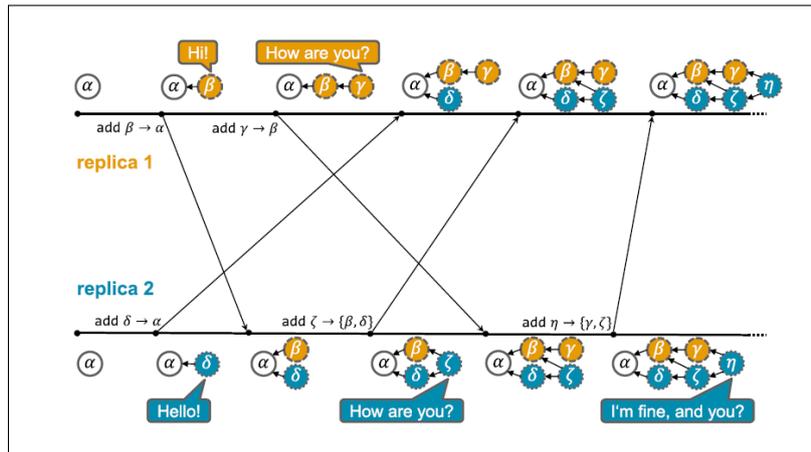


Figura 1.3. Evolução da estrutura MEG em duas réplicas com atualizações concorrentes [Jacob et al. 2021].

A figura mostra como eventos concorrentes são tratados: ao serem detectados por réplicas distintas, são integrados ao DAG por meio de vértices com múltiplos pais, respeitando a aciclicidade e permitindo fusões posteriores que convergem o estado. Esse comportamento caracteriza o MEG como uma estrutura do tipo CRDT¹⁵ orientada a operações.

A Figura 1.4 complementa a visão mostrando a estrutura conceitual do MEG como um CRDT. São destacados os principais componentes: vértices, arestas, regras de geração e aplicação, e o papel das extremidades como pontos de entrada para novos eventos.

A Tabela 1.5 resume as principais características técnicas do Matrix Event Graph no contexto de replicação distribuída. Cada linha destaca uma propriedade fundamental do MEG, combinando aspectos estruturais e operacionais com implicações diretas para a escalabilidade, confiabilidade e consistência do protocolo.

Esse resumo reforça como o MEG opera como uma estrutura de dados robusta para ambientes federados, conciliando replicação assíncrona com convergência eventual. No entanto, também evidencia limitações, como o possível crescimento da largura do DAG e a necessidade de estratégias específicas de otimização para ambientes de alta concorrência.

¹³Consistência eventual é a propriedade de sistemas distribuídos em que, após algum tempo sem novas atualizações, todas as réplicas convergem para o mesmo estado.

¹⁴As extremidades de um DAG são os vértices que não possuem filhos — ou seja, que ainda não foram referenciados por nenhum evento subsequente.

¹⁵CRDTs são tipos de dados replicados projetados para funcionar corretamente em sistemas distribuídos, mesmo com atualizações concorrentes, sem necessidade de sincronização ou ordenação global.

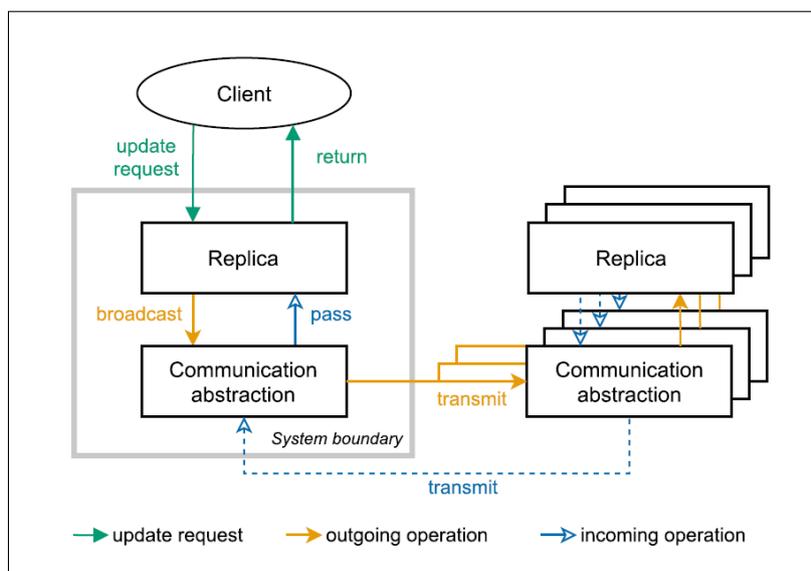


Figura 1.4. Esquema da estrutura do MEG como CRDT orientado a operações [Jacob et al. 2021].

Tabela 1.5. Propriedades do Matrix Event Graph (MEG) como estrutura de replicação distribuída

Propriedade	Descrição
Modelo estrutural	DAG (grafo acíclico direcionado), com nós representando eventos e arestas representando causalidade.
Replicação	Assíncrona e eventual, com trocas parciais de estado entre homerservers federados.
Fusão de eventos	Baseada em múltiplos pais; preserva causalidade sem ordenação global.
Detecção de extremidades	Nós sem filhos servem como pontos de referência para anexar novos eventos.
Consistência eventual	As réplicas convergem mesmo com atualizações concorrentes ou perda de conectividade.
Desempenho em larga escala	Potencial crescimento da largura do DAG pode afetar performance; requer otimizações.
Classificação	CRDT orientado a operações com semântica de merge e resolução de conflitos embutida.

Esse modelo de replicação permite alta robustez frente a falhas de conectividade e auditoria do histórico completo de eventos, o que o torna adequado para ambientes federados e descentralizados como os propostos pelo Matrix. No entanto, seu crescimento pode impactar o desempenho de servidores em larga escala. A ausência de uma política padronizada de podagem e a exigência de que todos os pais estejam presentes na réplica antes da aplicação de um evento aumentam a complexidade operacional em ambientes instáveis.

Estratégias como *state resolution algorithms*, compactação de grafos, otimização da propagação de eventos e algoritmos para controle da largura do DAG continuam sendo

objeto de estudo e aprimoramento contínuo [Jacob et al. 2021]. Esses pontos reforçam que o MEG, embora elegante e funcional, ainda representa uma fronteira ativa de pesquisa em replicação distribuída.

1.2.4. Mecanismos Avançados de Criptografia (Olm e Megolm)

A segurança no protocolo Matrix é baseada em E2EE, com suporte dos protocolos Olm¹⁶ e Megolm¹⁷, voltados respectivamente para comunicações *Peer-to-Peer* (P2P) e em grupo [Albrecht et al. 2023, Foundation 2025]. O Olm baseia-se no *Triple Diffie-Hellman* (3DH)¹⁸ e no *Double Ratchet*¹⁹ e estabelece sessões seguras entre pares. O Megolm, por sua vez, é uma extensão voltada à comunicação em grupo, concebida para maximizar a eficiência em contextos com múltiplos participantes.

Adicionalmente, o Matrix implementa recursos como *cross-signing*, que permite que os próprios usuários autentiquem os dispositivos confiáveis de sua conta, contribuindo para a verificação de identidade e mitigação de ataques do tipo *man-in-the-middle*. Há também suporte a backup criptografado de chaves nos servidores domésticos, permitindo a recuperação de mensagens após perda de dispositivos [Li et al. 2023].

Embora o protocolo forneça E2EE por padrão, também emprega TLS para proteger o tráfego entre clientes e servidores, e entre servidores na federação. A E2EE é conduzida exclusivamente por Olm e Megolm, o primeiro responsável pela distribuição de material criptográfico para os integrantes da sala e o segundo utilizado para troca de mensagens, tanto individuais como em grupos, já que todos os chats são tratados como salas no Matrix.

De forma mais detalhada, o Olm é responsável por estabelecer um canal seguro entre dois dispositivos, oferecendo confidencialidade, autenticidade, *Forward Secrecy* (FS), *Post-Compromise Security* (PCS) e negabilidade. Sua implementação original, em C/C++, foi disponibilizada como uma API, mas atualmente integra a biblioteca *vodozemac*²⁰, escrita em Rust.

Já o Megolm implementa sessões unidirecionais entre o dispositivo remetente e cada destinatário, tais sessões distribuídas individualmente por canais Olm. A sessão é composta por uma catraca simétrica que é avançada, derivando uma nova chave de cifra para cada mensagem. A autenticação é fornecida por assinaturas Ed25519 e HMAC-SHA256, e as propriedades de segurança dependem da integridade dos canais Olm usados na distribuição.

A Figura 1.5 ilustra o processo simplificado de envio de uma mensagem criptografada no Matrix entre Alice e Bob. Os passos de ❶ a ❿ abrangem desde a geração e troca de chaves até a cifração, assinatura e posterior decifração no dispositivo do receptor.

No passo ❶, ocorre a geração das chaves de sigilo e de assinatura. As partes

¹⁶<https://gitlab.matrix.org/matrix-org/olm>

¹⁷[\[https://gitlab.matrix.org/matrix-org/olm/-/blob/master/docs/megolm.md\]](https://gitlab.matrix.org/matrix-org/olm/-/blob/master/docs/megolm.md)

¹⁸O 3DH proporciona uma troca de chaves inicial entre as partes através de curvas elípticas.

¹⁹O Double Ratchet é um protocolo criptográfico que combina chaves assimétricas e simétricas para garantir sigilo direto e sigilo futuro.

²⁰[\[https://matrix-org.github.io/vodozemac/vodozemac/index.html\]](https://matrix-org.github.io/vodozemac/vodozemac/index.html)

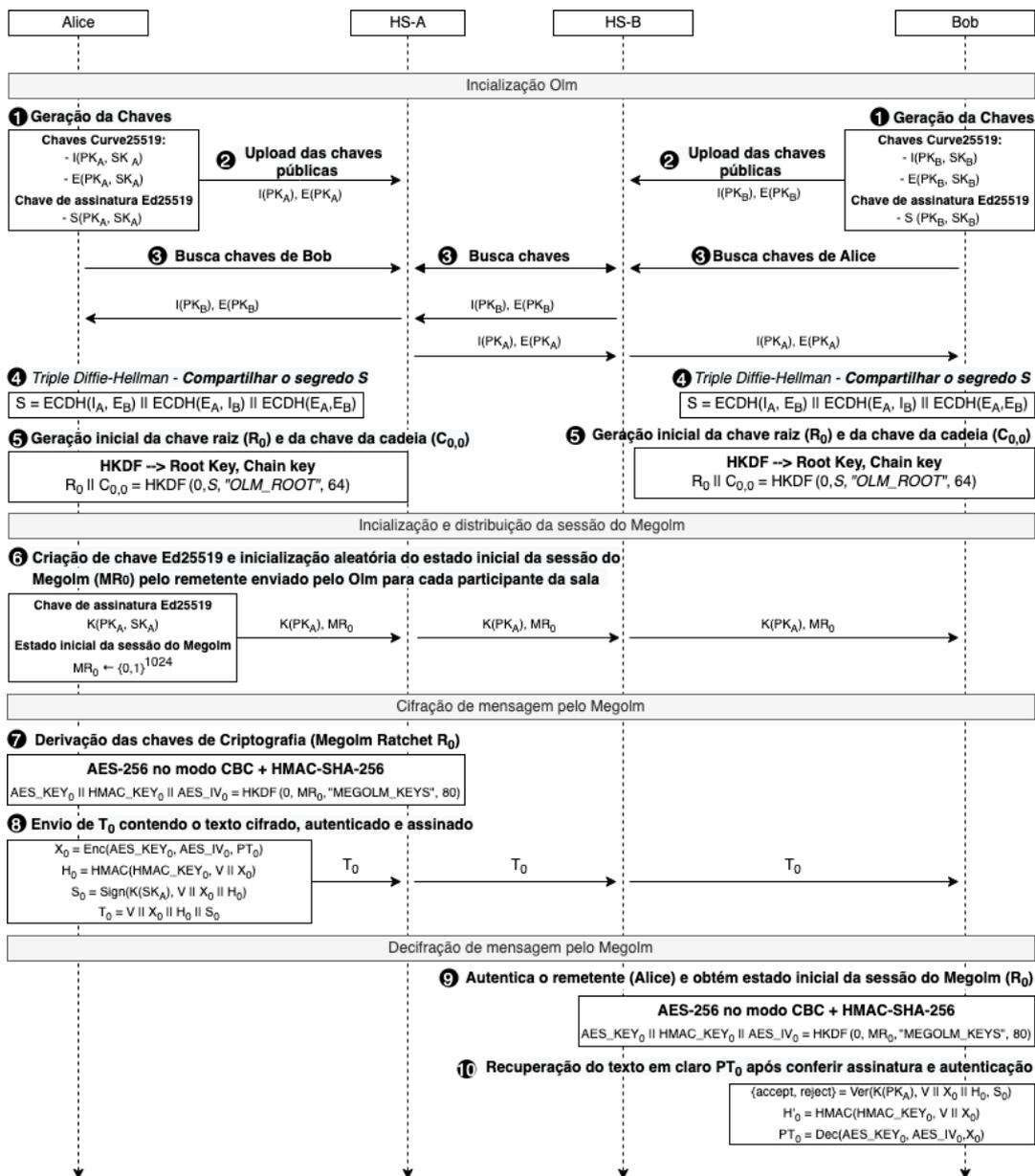


Figura 1.5. E2EE via Olm e Megolm.

públicas dessas chaves são publicadas nos respectivos *homeserver* (HS) de Alice e Bob ②. Quando Alice e Bob desejam se comunicar, cada um busca nos HS apropriados as chaves da outra parte ③, para que o protocolo 3DH seja executado ④. O resultado do 3DH é um segredo compartilhado S , que serve como insumo para a derivação da root key (R_0) e da chain key ($C_{0,0}$) do Double Ratchet presente no Olm ⑤, concluindo assim a inicialização da sessão Olm.

A etapa seguinte consiste na inicialização e distribuição da sessão do Megolm. Para isso, Alice — a remetente, neste cenário — cria um par de chaves para assinatura e inicializa aleatoriamente o estado inicial da sessão do Megolm (MR_0) ⑥. A parte pública

da chave de assinatura criada por Alice e o estado inicial da sessão do Megolm precisam ser transmitidos de forma segura para Bob, a fim de que ele consiga, posteriormente, verificar e decifrar as mensagens cifradas por Alice. Essa transmissão segura é realizada utilizando o Olm, cuja sessão já foi previamente estabelecida. Assim, Bob recebe de maneira segura o estado inicial da sessão Megolm (MR_0).

Com o estado inicial compartilhado, Alice pode dar início ao processo de cifração da mensagem em claro PT_0 . Para isso, ela realiza uma derivação de chaves utilizando *HMAC-based Extract-and-Expand Key Derivation Function* (HKDF) sobre o estado inicial do Megolm (MR_0), obtendo as chaves de cifração, o vetor de inicialização (IV) e o material para autenticação ⑦. De posse desses parâmetros, a mensagem é cifrada, autenticada e assinada, sendo então transmitida para Bob ⑧.

Bob, por sua vez, como já possui de forma segura o estado inicial da sessão Megolm (MR_0), pode replicar o processo descrito no passo ⑦, obtendo exatamente a mesma chave de cifração, o IV e o material para autenticação ⑨. Dessa forma, Bob é capaz de verificar a assinatura da mensagem com a chave pública de Alice previamente recebido em ⑥, confirmar sua autenticidade e, em caso de sucesso, proceder com a decifração do conteúdo ⑩.

A Tabela 1.6 sintetiza a comparação entre Olm e Megolm em aspectos como finalidade, mecanismo de inicialização, sigilo, autenticação, eficiência e escalabilidade.

Tabela 1.6. Comparação entre os protocolos Olm e Megolm

Característica	Olm	Megolm
Uso Principal	Mensagens ponto-a-ponto (1:1)	Mensagens em grupo (1 remetente → N receptores)
Inicialização	Triple Diffie-Hellman (3DH)	Distribuição via canais Olm
Ratchet Criptográfico	Double Ratchet simétrico e assimétrico	Megolm Ratchet simétrico
FS	Forte (ratchet por mensagem)	Limitado (receptores podem manter estados anteriores)
PCS	Presente (se pre-keys são assinadas)	Limitado (requer renovação de sessão manual)
Autenticação	Chaves de identidade (Curve25519)	Chave de assinatura do remetente (Ed25519)
Criptografia Utilizada	AES-CBC-256 + HMAC-SHA256	AES-CBC-256 + HMAC-SHA256
Granularidade de Segurança	Por mensagem	Por sessão (unidirecional)
Eficiência	Menos eficiente, mais seguro	Mais eficiente para grupos, com compromissos de segurança
Escalabilidade	Limitada: requer um canal Olm por par de dispositivos	Alta: uma única sessão pode ser compartilhada com vários dispositivos receptores

Enquanto Olm aplica o Ratchet simétrico do Double Ratchet para fornecer FS por mensagem (envio de sessões do Megolm), o Ratchet assimétrico é responsável por prover PCS. Já o Megolm apresenta limitações nesse aspecto, especialmente quando sessões não são atualizadas regularmente. Conforme destacado pela documentação oficial [Matrix.org 2022], é necessário que os clientes descartem sessões antigas — seja avançando os valores da catraca ou reiniciando sessões — para minimizar a reutilização de chaves. Além disso, preservar o PCS requer a criação periódica de novas sessões Megolm, distribuídas por canais Olm seguros.

Por fim, embora Megolm seja menos robusto em certos aspectos de segurança, ele proporciona vantagens significativas em termos de desempenho e escalabilidade em grupos. Uma única sessão pode ser compartilhada com vários dispositivos receptores, enquanto Olm exige um canal separado para cada par de dispositivos, o que o torna menos adequado para ambientes com muitos participantes. A escolha entre os dois protocolos depende, portanto, do cenário de uso e dos requisitos de segurança envolvidos.

O principal desafio, tanto no Olm quanto no Megolm, para o fortalecimento da soberania digital reside na capacidade de integrar, de forma nativa, algoritmos criptográficos proprietários do Estado ao protocolo. Essa medida visa eliminar qualquer vulnerabilidade que comprometa a soberania, especialmente diante da hipótese de que algoritmos comerciais venham a ser quebrados por agências estrangeiras. Nessas circunstâncias, a dependência de algoritmos de mercado poderia configurar riscos reais de influência indevida, vazamento de informações sensíveis ou até mesmo de espionagem externa.

Portanto, a verdadeira soberania digital no contexto da comunicação criptografada não se resume à adoção de software livre ou à descentralização das infraestruturas. Ela exige, também, a capacidade técnica e política de incorporar algoritmos criptográficos sob controle estatal, auditáveis e alinhados às normativas locais de segurança e privacidade. A flexibilidade arquitetural do protocolo Matrix — especialmente nos mecanismos Olm e Megolm — oferece um ponto de partida promissor, mas seu pleno aproveitamento ainda depende de esforços de pesquisa, padronização e engenharia criptográfica soberana.

1.2.5. Interoperabilidade e Integração com Outras Plataformas

Um dos diferenciais do protocolo Matrix em relação a outras soluções de mensageria é sua arquitetura voltada à interoperabilidade. O sistema permite a criação de *bridges* para conectar o ecossistema Matrix a outras plataformas de comunicação como Slack, Telegram, Discord, IRC e até servidores de e-mail. Essas bridges atuam como tradutores de protocolos, possibilitando que usuários interajam entre redes distintas sem a necessidade de múltiplas contas²¹ ou interfaces [Cui et al. 2023, Foundation 2025].

A estrutura básica das bridges envolve a escuta e tradução dos eventos Matrix para o formato do protocolo externo, e vice-versa. Elas podem ser implementadas de forma unidirecional, que permite o envio de mensagens em apenas uma direção, geralmente do Matrix para outra plataforma, ou de forma bidirecional, que suportam a troca mútua de mensagens, mantendo a comunicação sincronizada entre os dois sistemas.

²¹Em muitas implementações, as bridges podem mapear identidades externas para usuários Matrix virtuais, criando experiências unificadas.

A Figura 1.6 ilustra a integração realizada por uma bridge entre um homeserver Matrix e a plataforma Slack. As mensagens enviadas na sala `friends` do homeserver `matrix.org` são capturadas pela bridge, traduzidas e encaminhadas para a sala `friends` no Slack, e o processo ocorre igualmente no sentido inverso.

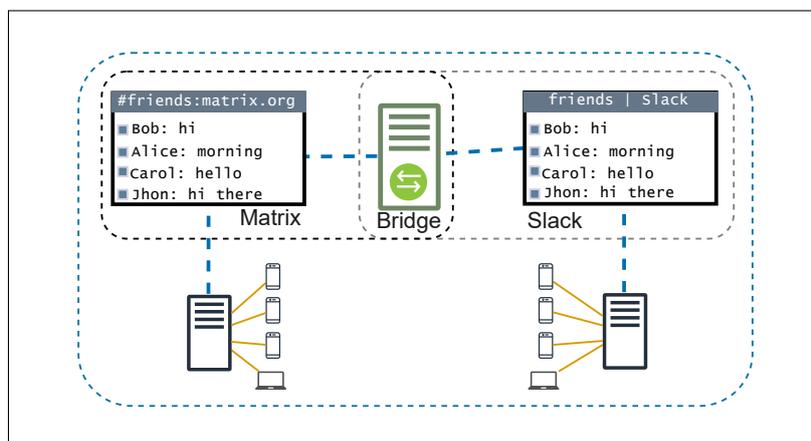


Figura 1.6. Integração entre Matrix e outras plataformas via bridge (ex: Slack, Telegram, IRC)

O gerenciamento dos estados e eventos nas bridges é fundamental para preservar a coerência das conversas e dos históricos, exigindo mecanismos que lidem com diferenças nos modelos de dados e nos formatos dos protocolos envolvidos.

O ecossistema Matrix conta com uma ampla variedade²² de bridges que permitem a integração com diversas plataformas de comunicação, tais como Slack, Discord, Telegram, WhatsApp, Signal, IRC, XMPP, Messenger, Skype, Google Chat, WeChat, SMS, iMessage, LinkedIn, Twitter, Instagram, Mastodon, entre outros. A diversidade de bridges disponíveis amplia significativamente o alcance e a aplicabilidade do Matrix, tornando-o uma solução versátil para comunicação descentralizada em ambientes heterogêneos. Alguns cenários de uso estratégicos são comunicação segura entre agências com infraestruturas heterogêneas, integração de canais institucionais legados e ambientes de colaboração em tempo real. Além disso, aplicações descentralizadas baseadas no Matrix têm sido exploradas em áreas como Internet das Coisas (IoT), redes comunitárias, jornalismo seguro e plataformas educacionais, reforçando seu papel como protocolo versátil e extensível [Li et al. 2023].

Apesar dos benefícios que a interoperabilidade traz para o protocolo Matrix, essa integração enfrenta desafios técnicos e conceituais que impactam diretamente sua eficiência e escalabilidade. Um dos principais desafios é a complexidade envolvida na tradução entre protocolos heterogêneos. Cada plataforma de comunicação possui seu próprio modelo de presença, formatos específicos para mensagens, regras para gerenciamento de salas, permissões e sinais de estado, além de diferentes capacidades para lidar com recursos multimídia e eventos em tempo real. Traduzir esses modelos distintos de forma fiel e eficiente requer um esforço para preservar a integridade da comunicação e evitar perdas ou distorções de informação.

²²Lista de bridges: <https://matrix.org/ecosystem/bridges/>

Outro desafio reside na manutenção da coerência e sincronização dos estados e mensagens entre sistemas diversos. Em ambientes distribuídos e federados, onde múltiplos eventos podem ocorrer simultaneamente ou fora de ordem, garantir que todos os participantes tenham uma visão consistente das conversas torna-se uma tarefa complexa. Isso é especialmente desafiador em cenários de alta concorrência, quando múltiplas bridges e usuários interagem intensamente, e nos casos de eventos assíncronos, que podem introduzir latências e divergências temporárias no estado das salas.

Além disso, questões relacionadas à segurança e privacidade ganham maior complexidade na interoperabilidade. Proteger dados sensíveis enquanto se mantém a compatibilidade entre diferentes sistemas com políticas e mecanismos de segurança variados é uma tarefa delicada e contínua.

1.2.6. Estrutura Conceitual das APIs

O protocolo Matrix é estruturado sobre um conjunto de APIs formais que definem a comunicação entre seus componentes principais. Essas interfaces são publicadas pela Matrix Foundation em uma especificação técnica padronizada, mantida de forma aberta e evolutiva.

A evolução do protocolo Matrix não se dá apenas no plano técnico, mas também na forma como suas especificações são desenvolvidas, debatidas e mantidas publicamente. A governança técnica do ecossistema é estruturada de maneira transparente e colaborativa, com base em dois mecanismos principais:

- **Matrix Spec:** a especificação técnica formal do protocolo, publicada e mantida pela Matrix Foundation²³. A spec define o comportamento esperado das APIs (Client-Server, Server-Server, Identity Service e Application Service), os tipos de eventos suportados, os fluxos de autenticação, a criptografia ponta-a-ponta e os formatos de comunicação padronizados entre clientes e servidores.
- **Matrix Spec Changes (MSCs):** propostas públicas de alteração ou extensão da spec, discutidas abertamente pela comunidade técnica via GitHub²⁴. As MSCs passam por processo formal de revisão técnica e, quando aprovadas, são incorporadas em versões futuras da spec oficial. Esse processo permite que a evolução do protocolo se mantenha ágil e responsiva às necessidades dos desenvolvedores.

Dentre essas interfaces, destacam-se principalmente a *Client-Server API* (CS API) e a *Federation API* (também chamada de Server-Server API). A CS API é a principal interface utilizada pelos clientes para interagir com seus homeservers. Ela oferece um conjunto rico de funcionalidades, incluindo autenticação de usuários, envio e recebimento de mensagens, criação e gerenciamento de salas, sincronização incremental do estado da rede e verificação de dispositivos para garantir a segurança das comunicações, especialmente em contextos com criptografia de ponta a ponta. Por sua vez, a Federation API

²³Especificações oficiais disponíveis em <https://spec.matrix.org>

²⁴Repositório de propostas MSC: <https://github.com/matrix-org/matrix-spec-proposals>

possibilita a comunicação entre homeservers, lidando com a replicação de eventos, resolução de conflitos de estado, verificação das assinaturas digitais entre servidores e a propagação de atualizações na rede federada. Essa distinção entre CS API e Federation API reflete a separação lógica entre o plano do usuário final e o plano da infraestrutura do servidor, garantindo descentralização e independência tecnológica dos participantes da rede [Li et al. 2023, Foundation 2025].

Além dessas APIs centrais, o ecossistema Matrix define outras interfaces especializadas que expandem suas capacidades. A *Application Service API* é projetada para permitir que *app services* se integrem profundamente ao protocolo. Essa API possibilita que aplicações terceiras monitorem eventos, enviem mensagens e participem da lógica de salas, tudo isso com permissões específicas e controle granular.

A *Identity Service API* complementa a arquitetura ao facilitar a associação e verificação de identidades externas ao Matrix, como endereços de e-mail ou números de telefone, mapeando-os para usuários Matrix. Esse serviço auxilia em fluxos de registro, recuperação de contas e provisionamento de identidades, ampliando a usabilidade e integração do protocolo com sistemas legados²⁵.

Por fim, a *Push Gateway API* trata da entrega de notificações push para dispositivos móveis e outros clientes que operam em ambientes restritivos, como redes móveis, onde conexões persistentes são inviáveis. Essa API fornece um canal eficiente e padronizado para que os homeservers encaminhem notificações externas, garantindo que os usuários sejam alertados em tempo real sobre mensagens e eventos relevantes, sem comprometer a autonomia da infraestrutura Matrix²⁶.

Em suma, essa arquitetura baseada em múltiplas APIs especializadas permite que o protocolo Matrix mantenha uma separação clara entre funções, promovendo uma rede federada, extensível e segura, na qual diferentes atores cooperam para viabilizar comunicações abertas e interoperáveis.

1.3. Aplicações Práticas do Protocolo Matrix

A compreensão de um protocolo de comunicação não se limita a seus fundamentos técnicos e arquiteturais. A forma como suas funcionalidades se traduzem em práticas cotidianas — especialmente do ponto de vista do usuário — é essencial para avaliar sua maturidade, segurança e aplicabilidade institucional. Esta seção apresenta uma abordagem prática das operações com o protocolo Matrix, tendo como foco a utilização do cliente Element X²⁷. O objetivo é demonstrar, com base em uma perspectiva prática, como usuários podem operar a infraestrutura federada sem a necessidade de instalar ou administrar um servidor próprio.

As subseções abordam, de forma aplicada, a criação de contas, o uso das funcionalidades básicas, o gerenciamento de salas e permissões e, por fim, os recursos de comunicação segura. Ao invés de apresentar um tutorial passo a passo, o foco está na

²⁵<https://spec.matrix.org/latest/identity-service-api/>

²⁶<https://spec.matrix.org/latest/push-gateway-api/>

²⁷O Element X é a nova geração de cliente oficial do protocolo Matrix, com ênfase em usabilidade e desempenho, disponível para dispositivos móveis e navegadores.

estrutura conceitual dessas operações e nas boas práticas recomendadas para seu uso em ambientes profissionais, governamentais ou educacionais.

1.3.1. Aspectos Conceituais na Configuração e Operação de Clientes Matrix

O protocolo Matrix foi projetado para ser acessível tanto por usuários finais quanto por desenvolvedores, permitindo sua adoção em larga escala sem a necessidade de operar servidores próprios. Essa característica é particularmente importante para instituições que desejam experimentar o protocolo de forma imediata, avaliando sua experiência de uso e funcionalidades antes de realizar implantações locais.

Uma das formas mais diretas de explorar o Matrix é por meio do serviço público `matrix.org`, que disponibiliza um homeserver gratuito e funcional mantido pela fundação responsável pelo protocolo. Usuários podem registrar-se nesse serviço e utilizar clientes como o *Element X*, o novo cliente oficial do ecossistema Matrix voltado para dispositivos móveis e desktops [Foundation 2025].

O processo de entrada envolve a criação de uma conta no domínio `matrix.org`, seguida da autenticação no aplicativo. O *Element X* oferece uma interface moderna e intuitiva, compatível com práticas atuais de design de aplicativos de comunicação. Após o login, o cliente realiza a sincronização inicial com o homeserver, permitindo ao usuário acessar suas salas, mensagens, notificações e dispositivos registrados.

O menu de configurações do *Element X* inclui opções para alterar nome de exibição, gerenciar dispositivos ativos, habilitar ou desabilitar notificações, e ativar E2EE em salas privadas. É possível também buscar por salas públicas por meio de diretórios federados, ingressar em discussões abertas e iniciar conversas privadas com outros usuários do ecossistema.

A Tabela 1.7 apresenta uma seleção representativa dos principais clientes Matrix disponíveis atualmente, com destaque para suas plataformas-alvo, linguagens de implementação e características técnicas mais relevantes. O *Element*, nas versões para desktop e dispositivos móveis, destaca-se como cliente oficial e mais amplamente adotado no ecossistema, oferecendo suporte completo à criptografia ponta a ponta, Spaces, Threads e a sincronização rápida introduzida pelo Sliding Sync. Sua variante móvel, o *Element X*, foi redesenhada para melhorar a experiência em smartphones, com ênfase em desempenho, fluidez e integração nativa.

Entre as alternativas, o *FluffyChat* se sobressai pela leveza, portabilidade e proposta de interface amigável, ideal para novos usuários. O *SchildiChat*, por sua vez, adapta o *Element* com foco em usabilidade tradicional e múltiplas contas. Já o *Nheko* é reconhecido por sua leveza, desempenho nativo e foco em segurança, sendo especialmente atrativo para usuários avançados em ambientes desktop. Essa diversidade de clientes evidencia a flexibilidade e maturidade do protocolo Matrix, permitindo adoções personalizadas em diferentes contextos de uso, desde ambientes institucionais até comunidades autônomas.

Como exercício prático, recomenda-se que os leitores criem uma conta no `matrix.org`, acessem o cliente *Element X*, localizem uma sala pública e realizem o envio de sua primeira mensagem. Essa atividade fornece uma compreensão imediata das capacidades do

Tabela 1.7. Comparativo entre clientes Matrix populares

Cliente	Plataforma	Linguagem	Destaques
Element X	Android, iOS	Kotlin, Swift	Interface moderna, Sliding Sync, suporte E2EE por padrão
Element	Desktop, Web, Linux, Android, macOS, Windows	JavaScript/React	Interface web baseada em Electron, suporte completo a Spaces e Threads
FluffyChat	Android, iOS, Web	Dart (Flutter)	Leve, amigável, alternativa ao Element, código aberto
SchildiChat	Android, Web, Desktop	Kotlin	Derivado do Element, interface estilo "mensageiros clássicos", suporte a múltiplas contas
Nheko	Linux, Windows	C++/Qt	Cliente leve, nativo, foco em desempenho e segurança

protocolo e do fluxo básico de uso de seus clientes oficiais.

1.3.2. Gerenciamento de Salas, Grupos e Permissões

No protocolo Matrix, a comunicação ocorre principalmente por meio de salas, que funcionam como espaços lógicos para interação entre usuários. As salas podem ser públicas, privadas ou protegidas, dependendo de seus parâmetros de visibilidade, acesso e autenticação. Compreender a estrutura dessas salas e a lógica de permissões associada é essencial para garantir segurança, organização e governança nas comunicações institucionais.

Salas públicas são acessíveis por qualquer usuário da federação Matrix. Elas podem ser descobertas por meio de diretórios globais ou locais e permitem ingresso livre, sem convite. Já as salas privadas são visíveis apenas a seus membros, exigindo convite para entrada. Salas protegidas funcionam como intermediárias: são visíveis publicamente, mas exigem autenticação ou aprovação manual para ingresso [Foundation 2025].

A gestão de salas envolve a atribuição de papéis aos participantes. O protocolo define níveis hierárquicos de poder, como administrador (gerencia configurações e usuários), moderador (controla permissões e conteúdo) e participante comum. Esses papéis são atribuídos por meio de valores numéricos chamados *power levels*, configuráveis em cada sala. A Tabela 1.8 resume os principais tipos de sala e suas configurações típicas de permissão.

Tabela 1.8. Resumo dos tipos de sala no Matrix e suas propriedades de acesso

Tipo de sala	Visível publicamente	Entrada livre	Criptografia E2EE
Pública	Sim	Sim	Opcional
Privada	Não	Não (convite)	Recomendado
Protegida	Sim	Requer aprovação/autenticação	Recomendado

É recomendável que salas privadas ou protegidas adotem E2EE, recurso suportado nativamente pelo Matrix, garantindo que somente os dispositivos autorizados possam descriptografar o conteúdo. A ativação do E2EE é feita por sala, e pode ser combinada com verificação de dispositivos e autenticação cruzada para ambientes mais sensíveis.

A governança local torna-se ainda mais relevante em contextos organizacionais. A capacidade de definir políticas de acesso, criar grupos de usuários com papéis distintos e aplicar mecanismos de moderação adequados é essencial para garantir integridade, privacidade e conformidade legal na comunicação digital [Bechara and Lechner 2024].

1.3.3. Comunicação Segura: Chamadas de Áudio e Vídeo (Element X)

O suporte a chamadas de áudio e vídeo é uma funcionalidade estratégica do protocolo Matrix, especialmente relevante em contextos que demandam privacidade, segurança e descentralização. Por meio do cliente Element X, é possível realizar chamadas diretas entre usuários com suporte à E2EE, sem intermediação de servidores centrais de mídia.

A tecnologia base utilizada para o transporte de mídia é o WebRTC (Web Real-Time Communication), que permite comunicação peer-to-peer entre dispositivos. O papel do protocolo Matrix é limitado à troca de sinalização (session initiation) entre os clientes envolvidos na chamada. Uma vez estabelecida, a chamada utiliza canais diretos com negociação de rota realizada via ICE, incluindo o uso de servidores STUN/TURN para contornar firewalls e NATs [Foundation 2025, Rahman and Wang 2024].

Do ponto de vista de segurança, o protocolo Matrix adota os algoritmos Olm (mensagens individuais) e Megolm (grupos) para proteger as comunicações. Chamadas realizadas entre usuários em salas com E2EE ativado herdam os mecanismos de proteção criptográfica já definidos para a sessão. Além disso, recomenda-se que os dispositivos envolvidos realizem verificação cruzada (cross-signing), assegurando que os pares são legítimos e não foram adulterados durante o processo de emparelhamento.

Por fim, o Element X também permite ativar backups criptografados de chaves, garantindo que sessões anteriores possam ser restauradas com segurança em caso de reinstalação ou perda de dispositivos. Esse recurso, aliado à verificação entre dispositivos, fortalece a proteção contra ataques de intermediário (MITM) e amplia a confiabilidade da comunicação em tempo real.

Como exercício opcional, sugere-se que os participantes iniciem uma chamada com outro usuário confiável e testem a verificação cruzada de dispositivos, observando as etapas de emparelhamento, ativação de E2EE e início da sessão multimídia.

1.4. Aspectos Avançados em Servidores Privados Matrix

A adoção do protocolo Matrix em contextos institucionais demanda, em muitos casos, a operação de servidores próprios. Essa prática permite maior controle sobre políticas de segurança, retenção de dados e conformidade regulatória — aspectos centrais da soberania digital. Nesta seção, exploramos a arquitetura do *Synapse*, o servidor de referência para Matrix, e discutimos desafios técnicos associados à sua implantação em ambientes reais.

1.4.1. Arquitetura do Servidor Synapse

Embora o protocolo Matrix permita o uso de servidores públicos como o `matrix.org`, uma das principais vantagens da arquitetura federada está na possibilidade de implantação de servidores privados. Essa abordagem garante maior controle sobre políticas de auten-

- **Proxy Reverso:** Realiza as funções de API Gateway e Proxy Reverso. Gerencia o tráfego entre os clientes e os serviços *backend*. Redireciona tráfego para os serviços da *Stack* do Synapse ou para o *Delegation Server*.
- **Delegation Server:** Antes de tudo, vale ressaltar que necessitamos de duas URLs para o funcionamento do sistema. São elas: (i) **URL do Matrix:** permite que um administrador de servidor doméstico mantenha um `server_name` de “example.com” para que IDs de usuário, aliases de sala, etc. continuem parecendo: “example.com”; e (ii) **URL dos Serviços:** permite acessar os *endpoints* dos diferentes serviços na arquitetura. Por exemplo, “matrix.example.com”. Neste cenário, o *Delegation Server* é um recurso do Matrix que permite que um administrador de servidor doméstico mantenha um `server_name` de example.com para que IDs de usuário, aliases de sala, etc. continuem parecendo “:example.com”, enquanto o tráfego de federação é roteado para um servidor e/ou porta diferente (por exemplo, “synapse.example.com:443”). O Delegation Server pode ser implementado em servidores Web tradicionais, como Apache e NGINX.
- **Synapse Stack:** Compõe o ecossistema Synapse os seguintes processos:
 - **Main Process:** Processo principal do servidor Synapse. Responsável pelas funções administrativas do ecossistema Synapse (Synapse Admin API), bem como pelo processo de sincronização entre os diferentes workers do ecossistema. Em uma instalação típica, o Synapse mantém somente um *Main Process* que processa eventos de clientes locais, sincroniza dados com outros servidores da federação, aplica lógica de acesso a salas e persiste informações em bancos relacionais como PostgreSQL. A comunicação federada é realizada por meio da Federation API, enquanto o relacionamento com os clientes é feito pela Client-Server API.
 - **Workers:** Os “workers” são aplicados quando é necessário escalar o Synapse, ajudando a distribuir a carga de trabalho do *Main Process* e melhorar a escalabilidade e o desempenho do servidor. Detalharemos os diferentes tipos de *workers* mais à frente.
- **Push Gateway (Signal):** Signal é um Push Gateway de referência para Matrix. Ele envia mensagens via Google/Firebase Cloud Messaging (GCM/FCM) e Apple Push Notification Service (APNS) e, portanto, pode ser usado para entregar notificações para aplicativos Android e iOS.
- **Camada de Dados:**
 - **Postgres:** Banco de dados relacional para persistência de dados do Synapse (todos os tipos de instâncias).
 - **Redis:** Cache e armazenamento de dados em memória, voltado para a implementação de cache, armazenamento de dados temporários e para a sincronização entre as diferentes instâncias do Synapse.
 - **Repositório de mídias:** voltado para o armazenamento de arquivos de mídia (imagens, vídeos, documentos, etc). Podem ser utilizados diretórios locais e

remotos (via *Network File System* - NFS, por exemplo), bem como um *storage* S3.

- **Servidor de Autorização:** Sua principal função é verificar a identidade dos usuários e conceder ou negar permissão para que eles realizem ações específicas dentro da rede Matrix. Essencialmente, ele atua como um porteiro digital, garantindo que apenas usuários autenticados e com as devidas permissões possam: acessar salas específicas, enviar mensagens, modificar configurações, etc. No Synapse, o servidor de autorização utiliza os protocolos modernos OAuth 2.0 e OpenID Connect (OIDC). Neste sentido, até então, temos dois serviços de autorização suportados pelo Synapse (considerando o Matrix 2.0):
 - O **MAS (Matrix Authentication Service)**: Agora mais conhecido como *matrix-authentication-service*, é um serviço de autenticação e autorização independente e autônomo projetado para o ecossistema Matrix;
 - **Keycloak**: É uma solução de software de código aberto para Gerenciamento de Identidade e Acesso (IAM). Ele permite que você proteja aplicações e serviços com pouco ou nenhum código, adicionando funcionalidades de autenticação e autorização.

1.4.1.2. Tipos de Workers

Workers são processos separados que executam tarefas específicas na infraestrutura do servidor Matrix. Eles são responsáveis por lidar com diferentes aspectos do funcionamento do Synapse, como processamento de eventos, manipulação de solicitações de API, replicação de dados, entre outros. Os “workers” são uma parte essencial para implementar uma arquitetura distribuída do Synapse, ajudando a distribuir a carga de trabalho e melhorar a escalabilidade e o desempenho do servidor. Até a data da escrita deste documento, existem os seguintes tipos de *workers*:

- **pusher**: É possível designar *workers* para enviar notificações *push* para um *gateway push*, como Sygnal e E-mail. Isso interromperá o processo principal de enviar notificações *push*. Vários *pushers* podem ser adicionados, caso em que a carga de trabalho é balanceado entre eles. Ou seja, suporta balanceamento de carga via *Homemserver*. Importante destacar a necessidade de reiniciar o *Main Process* e todos os *workers* após alterar esta opção. Esses *workers* não precisam aceitar solicitações HTTP de entrada para enviar notificações *push*;
- **user_dir**: Um diretório de usuários é mantido com base nos usuários que são “visíveis” para o servidor doméstico, ou seja, aqueles locais no servidor e aqueles com os quais qualquer usuário local compartilha uma sala. Além disso, permite pesquisa de texto completo com base em IDs de usuário e nomes de exibição. Esse *worker* é responsável por executar trabalhos em segundo plano relacionados à atualização do diretório do usuário e executar uma busca por usuários;

- **media_repository:** Clientes pode fazer *upload* e *download* de conteúdos de um repositório de mídias. Este *worker* trata as requisições direcionadas para o gerenciamento do repositório de mídias. Se estiver executando vários repositórios deste tipo, eles devem estar no mesmo servidor e deve-se especificar uma única instância para executar as tarefas em segundo plano na configuração compartilhada. Suporta balanceamento de carga com escalabilidade horizontal automática;
- **appservice:** Lida com o envio de tráfego de saída para *Application Services*. Este *worker* não pode ter carga balanceada. Portanto, deve haver apenas um *worker appservice*.
- **federation_sender:** É possível delegar os processos que manipulam o envio de solicitações de federação de saída. Isso removerá a necessidade de realizar essa função pelo *Main Process*. Vários *workers* deste tipo podem ser adicionados, onde o trabalho é balanceado entre eles. A maneira como o balanceamento de carga funciona é que qualquer solicitação de federação de saída será atribuída a um *worker* remetente de federação com base no *hash* do nome do servidor de destino. Isso significa que todas as solicitações enviadas ao mesmo destino serão processadas pela mesma instância de *worker*. Vários *federation_sender_instances* são úteis se houver uma federação com vários servidores. Essa configuração deve ser compartilhada entre todos os *workers* que manipulam o envio de federação e, se alterada, todos os *workers* remetentes de federação devem ser interrompidos ao mesmo tempo e, em seguida, iniciados para garantir que todas as instâncias estejam sendo executadas com a mesma configuração (caso contrário, os eventos podem ser descartados).
- **client_reader:** Responsável pelo Client-Server API. Suporta balanceamento de carga com escalabilidade horizontal automática. Contudo, solicitações de registro/login podem ser tratadas separadamente (*Main Process* ou um *Client Reader* só para elas) apenas para garantir que cargas inesperadas não afetem novos logins e inscrições;
- **federation_reader:** lida com o envio de solicitações de federação (Server-Server API). Suporta balanceamento de carga com escalabilidade horizontal automática;
- **federation_inbound:** Responsável por enviar EDUs (Ephemeral Data Unit) e PDUs entre homeservers, que representam atividades ao vivo. Cada PDU contém um único evento de Sala que o servidor de origem deseja enviar ao destino. EDUs, em comparação com PDUs, não têm um ID, um ID de sala ou uma lista de IDs “anteriores”. Elas são destinadas a serem dados não persistentes, como presença do usuário, notificações de digitação, etc. Suporta balanceamento de carga via proxy reverso.
- **event_persister:** Responsável por receber novos eventos, vinculá-los aos que já estão no DAG (gráfico acíclico direcionado) da sala, persisti-los no banco de dados e, finalmente, atualizar o fluxo de eventos. O fluxo de eventos suporta ter vários *Event Persisters*, onde a carga é dividida entre eles por ID da sala. Ou seja, suporta balanceamento de carga via *Homeserver*. Certifique-se de que o *Main Process* e

todos os *workers* sejam reiniciados após alterar esta opção. Esses *workers* não precisam aceitar solicitações HTTP de entrada;

- **background_worker:** O *worker* é usado para executar tarefas em segundo plano (por exemplo, limpeza de dados expirados). Se não for fornecido, o padrão será o processo principal. Este *worker* não pode ser fragmentado. Portanto, deve haver apenas um *background worker*. Isso é imposto para a segurança do seu banco de dados.
- **event_creator:** Escuta solicitações de clientes para criar novos eventos. Ele então passará esses eventos por replicação HTTP para quaisquer *Event Persisters* configurados (ou para o *Main Process*, se nenhum estiver configurado). Suporta balanceamento de carga com escalabilidade horizontal automática.
- **frontend_proxy:** O cliente envia requisições para este *worker* para publicar as chaves públicas de criptografia de ponta a ponta e algoritmos suportados pelo seu dispositivo.
- **account_data:** Este *worker* persiste as alterações nos dados da conta de usuários locais. Funciona como um escritor para o fluxo de notificações de atualizações de dados de conta.
- **presence:** Cada usuário tem o conceito de informação de presença. Isso codifica: (1) Se o usuário está online no momento; (2) Quão recentemente o usuário esteve ativo pela última vez (conforme visto pelo servidor); (3) Se um determinado cliente considera o usuário como ocioso no momento; (4) Informações arbitrárias sobre o status atual do usuário (por exemplo, “em uma reunião”). Essas informações são coletadas de dados por dispositivo (online, ocioso, last_active) e por usuário (status), agregados pelo *Homeserver* e transmitidos como um evento “m.presence”. Eventos de presença são enviados para partes interessadas onde os usuários compartilham uma associação de sala. Este *worker* funciona como um escritor para o fluxo de notificações de presença.
- **receipts:** Recibos são uma forma de reconhecimento de um evento. Os clientes devem enviar recibos quando houver alguma certeza de que o evento em questão foi exibido ao usuário. Este *worker* funciona como um escritor para o fluxo de notificações de recibos.
- **to_device:** Este *worker* fornece um meio pelo qual os clientes podem trocar mensagens de sinalização sem que elas sejam armazenadas permanentemente como parte de um histórico de comunicação compartilhado. Uma mensagem é entregue exatamente uma vez para cada dispositivo cliente. A principal motivação para esta API é trocar dados que não têm sentido ou que não são desejados para persistir no DAG da sala - por exemplo, tokens de autenticação única ou dados de chave. Não se destina a dados de conversação, que devem ser enviados usando a API normal “/rooms/<room_id>/send” para consistência em todo o Matrix.

- **typing:** Os usuários podem desejar ser informados quando outro usuário estiver digitando em uma sala. Isso pode ser feito usando notificações de digitação. Este *worker* é responsável por lidar com a escrita das notificações de digitação.
- **push_rules:** Os usuários também podem configurar várias regras que determinam quais eventos geram notificações *push*. Uma regra *push* (push-rule) é uma regra única que define sob quais condições um evento deve ser passado para um *push gateway* e como a notificação deve ser apresentada. Todas elas são armazenadas e gerenciadas pelo *homeserver* do usuário. Isso permite que as configurações de *push* específicas do usuário sejam reutilizadas entre aplicativos clientes. Este *worker* funciona como um escritor para o fluxo de notificações de *push-rules*.

Cada tipo de *worker* no Synapse desempenha um papel específico no processamento e na entrega de mensagens e eventos na rede Matrix. Eles trabalham juntos para garantir que o servidor Synapse funcione de forma eficiente e confiável, mesmo em ambientes de alta carga e escalabilidade.

1.4.1.3. Componentes para Comunicação em Tempo Real

Tradicionalmente, a operação de um servidor Matrix privado envolvia a integração com ferramentas externas para viabilizar funcionalidades complementares, como chamadas de voz e vídeo. Nesse modelo, os componentes mais utilizados eram o *Coturn*, um servidor TURN/STUN necessário para permitir a comunicação em redes com NAT, e o *Jitsi Meet*, utilizado para videoconferência por meio da ponte `jitsi-meet-tokens` e integração via bot/bridge com salas Matrix [Foundation 2020b, Foundation 2020a].

Nesse cenário, a sinalização de chamadas era realizada por meio do protocolo Matrix, enquanto a mídia era transmitida fora do domínio Matrix, através de conexões diretas entre clientes mediados por servidores Jitsi e Coturn. Tal separação trazia limitações à implementação de E2EE em chamadas em grupo, comprometendo a privacidade em certos cenários institucionais.

Recentemente, com o surgimento do padrão MSC3401 — que especifica a sinalização WebRTC diretamente sobre o protocolo Matrix — surgiu a possibilidade de um novo paradigma: o de chamadas nativamente integradas à rede Matrix [Foundation 2023]. Essa mudança foi operacionalizada pela ferramenta *Element Call*, que implementa chamadas individuais e em grupo com suporte completo a E2EE, utilizando o próprio Matrix como meio de sinalização e gerenciamento de sessões. Para o transporte eficiente da mídia em larga escala, o *Element Call* pode ser integrado ao *LiveKit*, uma solução moderna baseada em WebRTC com suporte a SFU (Selective Forwarding Unit), melhorando o desempenho e escalabilidade de videoconferências [Element 2025a, Element 2025c]. A Tabela 1.9 resume as diferenças principais entre as duas arquiteturas.

Tabela 1.9. Comparativo entre arquiteturas de mídia no ecossistema Matrix.

Componente	Arquitetura Tradicional	Arquitetura Moderna
Sinalização	Matrix + Bot Jitsi	Matrix MSC3401 (nativo)
Serviço de Mídia	Jitsi Meet	LiveKit (via Element Call)
E2EE Suportado	Parcial (chamadas diretas)	Sim (grupo e P2P)
Relay de Mídia	Coturn	Coturn (ainda necessário)
Integração com Matrix	Indireta (bot externo)	Direta via Client/Server API

1.4.2. Desafios Técnicos: Segurança, Desempenho e Escalabilidade

A implantação de um servidor Synapse em ambiente institucional requer uma abordagem técnica cuidadosa. Desde a instalação inicial até ajustes finos de desempenho e segurança, cada etapa influencia diretamente a confiabilidade do sistema e a proteção dos dados trafegados. Esta subseção apresenta orientações práticas e discute os principais desafios técnicos enfrentados durante a operação de um homeserver privado Matrix.

O processo de instalação pode ser realizado por meio de pacotes pré-compilados, imagens Docker ou scripts automatizados mantidos pela Matrix Foundation²⁸. Após a instalação, é necessário configurar arquivos como `homeserver.yaml`, onde são definidas as portas de escuta, domínios autorizados, políticas de autenticação e parâmetros de retenção.

A segurança do Synapse envolve múltiplos aspectos. Primeiramente, recomenda-se ativar o suporte a TLS por meio de um proxy reverso (como Nginx ou Caddy), garantindo que toda comunicação seja cifrada. Além disso, a autenticação pode ser integrada a serviços externos como LDAP ou OpenID Connect, permitindo unificação de credenciais e autenticação de múltiplos fatores. A configuração de permissões por sala e a limitação de recursos por usuário são práticas recomendadas para evitar abusos e contenção de recursos.

Em relação ao armazenamento, o Synapse oferece suporte a SQLite e PostgreSQL. Para ambientes de produção, é altamente recomendado utilizar PostgreSQL, que oferece melhor desempenho, suporte a concorrência e maior robustez em operações complexas [Synapse 2025]. O SQLite pode ser suficiente para testes locais ou pequenas implantações de avaliação, mas não é adequado para uso contínuo com múltiplos usuários simultâneos.

A escalabilidade do Synapse pode ser alcançada por meio de técnicas como separação de workers, cache persistente, tuning de banco de dados e otimização do uso de CPU e disco. A documentação oficial oferece guias detalhados sobre como distribuir a carga de trabalho entre processos distintos, possibilitando o crescimento progressivo da infraestrutura conforme a demanda [Foundation 2025].

²⁸Ver documentação oficial: <https://matrix-org.github.io/synapse/latest/>.

Quadro – Checklist para configuração segura e escalável do Synapse

- Configurar proxy reverso com TLS.
- Integrar autenticação externa (LDAP, SSO).
- Habilitar verificação de dispositivos e E2EE.
- Utilizar PostgreSQL como backend.
- Limitar conexões, tamanho de mensagens e eventos.
- Aplicar separação de workers e tuning de banco.

Recomenda-se que os leitores instalem um servidor local de teste, configurem um usuário administrativo e validem a comunicação entre clientes Matrix e o homeserver implantado. Essa experiência prática permite compreender o ciclo completo de autenticação, envio e sincronização de mensagens em uma rede federada.

1.4.3. Comunicação em Tempo Real: Considerações Práticas

O suporte a comunicação síncrona, como chamadas de voz e vídeo, é um dos diferenciais do protocolo Matrix em relação a outras soluções federadas. Esse recurso é especialmente importante em contextos institucionais que demandam autonomia na infraestrutura de comunicação. Nesta subseção, discutimos os requisitos técnicos, desafios e boas práticas relacionados à operação de chamadas em tempo real em servidores Synapse.

As chamadas no Matrix são viabilizadas por meio da tecnologia WebRTC, que permite a comunicação direta entre dispositivos sem a necessidade de servidores intermediários. O papel do Synapse nesse processo se limita à sinalização entre os participantes da chamada — o transporte de mídia ocorre por conexões peer-to-peer sempre que possível [Foundation 2025, Rahman and Wang 2024].

Para que essa comunicação seja bem-sucedida, é necessário configurar corretamente a rede e os serviços auxiliares. Em especial, o uso de servidores STUN e TURN é essencial para atravessar NATs e firewalls. O protocolo ICE (Interactive Connectivity Establishment) é utilizado para negociar o melhor caminho entre os pares. A ausência ou má configuração desses elementos pode impedir completamente o estabelecimento de chamadas²⁹.

O desempenho das chamadas também depende da infraestrutura de rede e dos recursos computacionais do servidor. Embora a mídia não transite diretamente pelo Synapse, sobrecargas no sistema — como lentidão na sincronização de eventos ou picos de carga no banco de dados — podem prejudicar a experiência do usuário, especialmente no tempo de sinalização inicial. Recomenda-se monitorar o consumo de CPU, memória e uso de rede, além de aplicar limites a salas ou usuários com uso abusivo.

Do ponto de vista da segurança, recomenda-se ativar criptografia ponta-a-ponta

²⁹Ver guia de configuração de chamadas em <https://matrix-org.github.io/synapse/latest/usage/voip.html>.

em salas privadas e realizar verificação cruzada de dispositivos antes do início das sessões. Em ambientes institucionais, é desejável restringir a funcionalidade de chamadas a salas ou domínios autorizados, mitigando riscos relacionados a chamadas não autenticadas ou a possíveis abusos da rede federada.

Como atividade prática, sugere-se realizar chamadas entre contas registradas em diferentes homeservers, observando o comportamento da sinalização e dos logs do servidor durante o processo de estabelecimento e encerramento da sessão.

1.4.4. Tutorial de Instalação do Matrix

Uma instalação mínima funcional de um servidor Matrix utilizando Synapse e Element pode ser facilmente implementada com Docker Compose. Um exemplo prático e bem documentado dessa configuração encontra-se no projeto oficial `element-docker-demo`, mantido pela equipe do Element [Element 2025b].

Os componentes essenciais utilizados neste projeto são:

- **Element Web:** Interface web oficial do Matrix para usuários finais.
- **Element Call:** Aplicação para chamadas de voz e vídeo com criptografia ponta-a-ponta.
- **Synapse:** Servidor Matrix responsável pela autenticação, sinalização e federação.
- **Matrix Authentication Service (MAS):** Serviço dedicado para autenticação integrada e OpenID Connect.
- **LiveKit:** Plataforma SFU para WebRTC, gerenciando mídia em chamadas grupais.
- **PostgreSQL:** Banco de dados relacional para armazenamento persistente e eficiente dos dados.
- **Nginx + Let's Encrypt ou mkcert (para TLS):** Proxy reverso configurado para segurança e criptografia HTTPS.
- **Coturn:** Servidor TURN/STUN para viabilizar chamadas através de redes restritivas.

A Figura 1.8 ilustra um diagrama simplificado desta instalação mínima.

Passos para execução:

1. Instale Docker Compose.
2. Caso utilize localmente, instale o `mkcert` para gerenciamento local de certificados TLS.

Execute o seguinte comando:

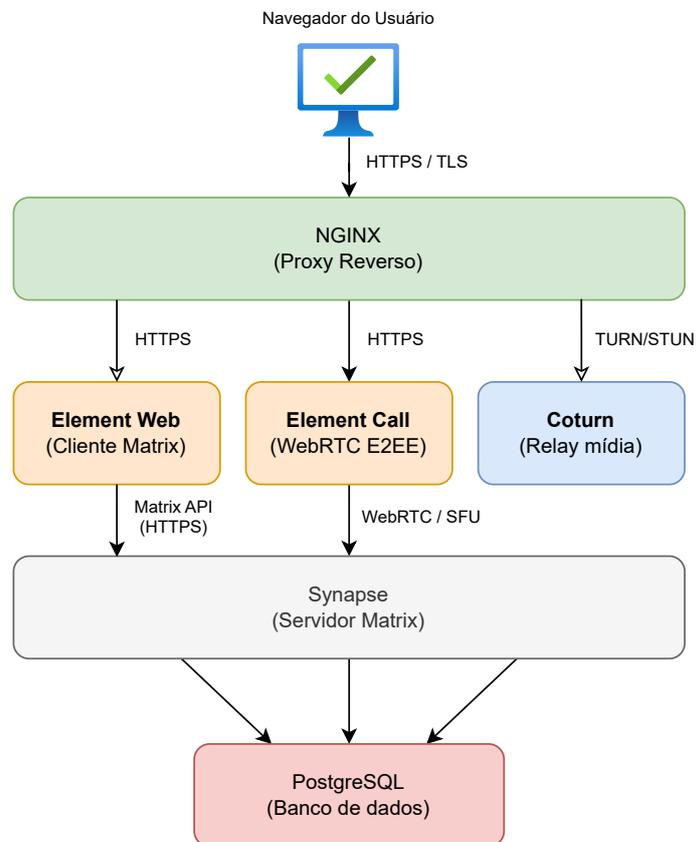


Figura 1.8. Diagrama simplificado da infraestrutura instalada

```
1 ./setup.sh
```

Configure o DNS para apontar `*.seu-dominio` para o host Docker. Caso utilize localmente com `mkcert`, execute:

```
1 source .env
2 sudo sh -c "echo 127.0.0.1 $DOMAINS >> /etc/hosts"
```

Em seguida, inicie os containers:

```
1 docker compose up
```

Após iniciar, acesse pelo navegador em `https://element.seu-dominio`.

Configuração personalizada: Verifique o arquivo `.env` ou personalize os templates em `/data-templates`. Reinicie com os comandos:

```
1 docker compose down
2 docker compose up -d
```

Os dados dos containers são armazenados no diretório `./data`, e segredos em `./secrets`. Note que arquivos de configuração em `./data` são recriados pelos templates a cada inicialização.

Administração do sistema: Para atualizar os containers:

```
1 docker compose pull
```

Registrar um novo usuário:

```
1 docker compose exec mas mas-cli -c /data/config.yaml manage
   register-user
```

Diagnóstico e resolução de problemas: Para verificar a configuração OIDC e diagnosticar problemas com TLS:

```
1 docker compose exec mas mas-cli -c /data/config.yaml doctor
```

Agora temos uma infraestrutura Matrix pronta, funcional e fácil de utilizar.

1.5. Automação com APIs e Bots no Matrix

Apresentar o uso prático das APIs do Matrix para automação de tarefas e criação de bots, utilizando Python. A seção foca na interação programática com homeservers via Client-Server API, com exemplos reais e reproduzíveis, aplicáveis a contextos institucionais e educacionais.

1.5.1. Automação e Gerenciamento Através das APIs Matrix

A automação de interações com servidores Matrix pode ser realizada por meio de sua *Client-Server API* (CS API), que fornece uma interface RESTful³⁰ padronizada para envio de mensagens, autenticação, gerenciamento de salas e outras operações essenciais. Essa API é amplamente documentada e permite que scripts, aplicações e bots interajam programaticamente com um homeserver, sem depender da interface gráfica de clientes tradicionais como o Element.

Para fins de automação com Python, é possível utilizar diretamente bibliotecas como `requests`³¹, construindo manualmente as requisições HTTP. No entanto, para

³⁰RESTful é um estilo arquitetural para comunicação entre sistemas baseado em operações HTTP padronizadas (como GET, POST, PUT, DELETE). Interfaces RESTful são amplamente utilizadas em APIs modernas pela simplicidade e compatibilidade com a Web.

³¹`requests` é uma biblioteca HTTP popular do ecossistema Python que permite enviar requisições REST com sintaxe simplificada.

simplificar o desenvolvimento e garantir conformidade com a especificação, é recomendável adotar bibliotecas especializadas como a `matrix-nio`³², que encapsula as chamadas da CS API e fornece uma estrutura orientada a eventos e sessões persistentes.

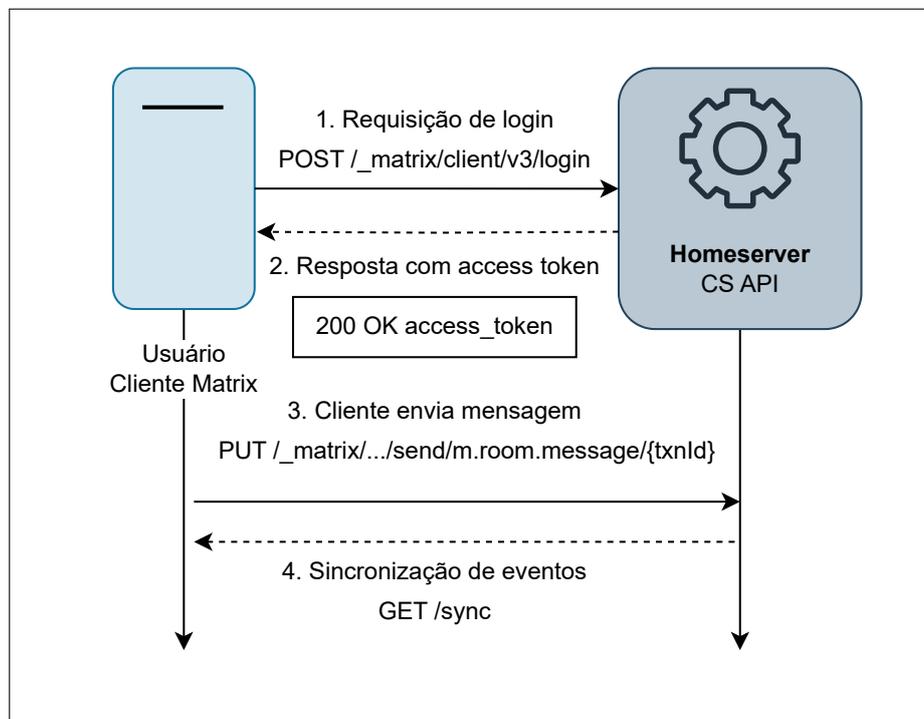


Figura 1.9. Fluxo de autenticação e envio de mensagem via CS API.

A seguir, apresenta-se um exemplo simples de script em Python que autentica um usuário com **token de acesso**³³ e envia uma mensagem a uma sala:

```

1 import requests
2
3 access_token = "YOUR_ACCESS_TOKEN"
4 room_id = "!roomid:matrix.org"
5 homeserver = "https://matrix.org"
6
7 message = {
8     "msgtype": "m.text",
9     "body": "Ola, esta e uma mensagem automatica!"
10 }
11
12 url = f"{homeserver}/_matrix/client/v3/rooms/{room_id}/send/m
    .room.message"
13 headers = {
14     "Authorization": f"Bearer {access_token}"
  
```

³²Disponível em: <https://github.com/poljar/matrix-nio>. Biblioteca Python para interação com a Client-Server API do Matrix, com suporte assíncrono, criptografia e gerenciamento de sessões.

³³Tokens de acesso são credenciais temporárias que identificam um usuário autenticado. No Matrix, eles são usados em requisições HTTP via cabeçalhos `Authorization`.

```

15 }
16
17 response = requests.post(url, json=message, headers=headers)
18 print("Status:", response.status_code)

```

Listing 1.3. Envio de mensagem via CS API com Python

Comentários sobre o código:

O Listing 1.3 utiliza a biblioteca `requests` para interagir diretamente com o endpoint da Client-Server API. Primeiramente, são definidos o token de acesso, o `room_id` da sala e a URL base do homeserver. Em seguida, o corpo da mensagem é montado no formato JSON com o tipo de mensagem `m.text` e o texto desejado. A requisição POST é então enviada para o endpoint específico de envio de mensagens, com o token incluído no cabeçalho de autenticação (`Authorization: Bearer <token>`). A resposta HTTP indica o sucesso ou falha da operação.

A seguir uma lista de atividades úteis que podem ser executadas com **requests**:

- **Listar salas públicas disponíveis no servidor** (GET `/publicRooms`): Permite recuperar a lista de salas abertas em um homeserver Matrix, útil para explorar ambientes públicos sem autenticação completa.
- **Convidar um usuário para uma sala** (POST `/rooms/{roomId}/invite`): Envia um convite para outro usuário ingressar em uma sala. É uma forma prática de simular a gestão de participantes por bots ou scripts administrativos.
- **Modificar o nome da sala** (PUT `/rooms/{roomId}/state/m.room.name`): Atualiza o nome visível da sala para todos os participantes. Permite estudar os mecanismos de alteração de estado via API REST.
- **Definir ou alterar o tópico da sala** (PUT `/rooms/{roomId}/state/m.room.topic`): Define uma mensagem de topo visível no cabeçalho da sala. Essencial para sinalizar objetivos ou orientações de uso.
- **Encerrar a participação do usuário em uma sala** (POST `/rooms/{roomId}/leave`): Realiza a operação de saída voluntária da sala. Útil para estudar o ciclo de entrada e saída de ambientes federados.

Os exemplos com `requests` realizam o envio de uma mensagem simples com autenticação via token. Em contextos mais complexos, é possível usar a biblioteca `matrix-nio` para gerenciar sessões, receber eventos e reagir a comandos em tempo real, conforme será explorado na próxima subseção.

Exemplo com a biblioteca `matrix-nio`

A seguir, apresenta-se um exemplo simples de cliente usando a biblioteca assíncrona `matrix-nio` para enviar uma mensagem de forma assíncrona a uma sala. Essa abordagem é recomendada para aplicações persistentes e bots mais robustos, pois a biblioteca gerencia sessões, estado e eventos com maior controle.

```

1 from nio import AsyncClient, RoomMessageText
2 import asyncio
3
4 async def main():
5     homeserver = "https://matrix.org"
6     user_id = "@seu_usuario:matrix.org"
7     access_token = "YOUR_ACCESS_TOKEN"
8     room_id = "!roomid:matrix.org"
9
10    client = AsyncClient(homeserver, user_id)
11    client.access_token = access_token
12
13    await client.room_send(
14        room_id=room_id,
15        message_type="m.room.message",
16        content={
17            "msgtype": "m.text",
18            "body": "Mensagem enviada com matrix-nio!"
19        }
20    )
21    await client.close()
22
23    asyncio.run(main())

```

Listing 1.4. Envio de mensagem com matrix-nio

Comentários sobre o código:

No Listing 1.4, utilizamos a biblioteca assíncrona `matrix-nio` para enviar uma mensagem. Após instanciar o cliente com o endereço do homeserver e o ID do usuário, o token de acesso é atribuído diretamente. A função `room_send` é usada para enviar a mensagem com os parâmetros necessários: ID da sala, tipo da mensagem e o conteúdo JSON. A execução ocorre dentro de uma função assíncrona principal controlada por `asyncio`, garantindo que o envio aconteça corretamente. Por fim, o cliente é fechado com `client.close()`.

Esse exemplo assume que o token de acesso já foi obtido previamente. A biblioteca `matrix-nio` também permite lidar com eventos, respostas de usuários e múltiplos tipos de mensagens, sendo amplamente utilizada no desenvolvimento de bots e clientes personalizados.

Comparativo entre uso direto da API e uso de `matrix-nio`

Para automatizações simples, o uso direto da API via `requests` pode ser suficiente. Entretanto, para aplicações que exigem persistência, interação com eventos ou múltiplos recursos do protocolo Matrix, a biblioteca `matrix-nio` oferece uma base mais robusta. A Tabela 1.10 resume as principais diferenças entre as abordagens.

Tabela 1.10. Comparativo entre uso direto da CS API com `requests` e uso de `matrix-nio`

Aspecto	<code>requests</code> (HTTP direto)	<code>matrix-nio</code> (cliente dedicado)
Facilidade inicial	Simples para scripts pontuais e diretos	Requer conhecimento em programação assíncrona
Controle do fluxo HTTP	Total, com headers e JSON manualmente configurados	Abstraído por métodos prontos da biblioteca
Gerenciamento de estado	Manual (tokens, sessões, eventos)	Gerenciado internamente com persistência opcional
Recepção de eventos	Difícil (requere polling ou webhooks)	Nativo via callbacks e handlers reativos
Desempenho	Adequado para tarefas simples	Escalável e eficiente para bots e serviços contínuos
Complexidade do código	Reduzida, porém repetitiva e limitada	Inicialmente mais verboso, mas modular e reutilizável
Escalabilidade	Limitada a casos simples	Projetada para ambientes persistentes e multiusuário

Como mostrado na Tabela 1.10, a escolha entre as abordagens depende do escopo da aplicação. Para scripts pontuais, `requests` pode ser a melhor escolha. Já para bots reativos ou integrações mais complexas, a estrutura fornecida por `matrix-nio` proporciona maior robustez, modularidade e desempenho.

1.5.2. Desenvolvimento Conceitual e Prático de Bots

Bots são aplicações autônomas que interagem com o protocolo Matrix de maneira persistente e reativa. Embora scripts manuais também possam realizar operações programadas, os bots diferenciam-se por sua capacidade de escutar eventos, responder a mensagens em tempo real e manter sessões contínuas com o servidor.

Utilizando bibliotecas como `matrix-nio`, é possível desenvolver bots que monitoram salas, executam comandos personalizados, aplicam regras de moderação e coletam métricas de uso. Essa automação é útil em diversos contextos: comunidades técnicas, ambientes educacionais, suporte automatizado e gestão de canais institucionais.

A estrutura de um bot em Matrix geralmente inclui três elementos centrais: conexão ao servidor, escuta de eventos e lógica de resposta. A seguir, apresenta-se um exemplo simples de bot que responde a comandos iniciados com `!`:

```

1 from nio import AsyncClient, RoomMessageText
2 import asyncio
3
4 class SimpleBot:
5     def __init__(self, client):
6         self.client = client
7
8     async def message_callback(self, room, event):
9         if isinstance(event, RoomMessageText):
10            if event.body.startswith("!ping"):
11                await self.client.room_send(

```

```

12         room_id=room.room_id,
13         message_type="m.room.message",
14         content={"msgtype": "m.text", "body": "
15             pong!"}
16     )
17     async def run(self):
18         self.client.add_event_callback(self.message_callback,
19             RoomMessageText)
20         await self.client.sync_forever(timeout=30000)
21 client = AsyncClient("https://matrix.org", "@seu_usuario:
22     matrix.org")
23 client.access_token = "YOUR_ACCESS_TOKEN"
24 bot = SimpleBot(client)
25 asyncio.run(bot.run())

```

Listing 1.5. Bot simples com resposta automática a comandos

Comentários sobre o código:

Esse exemplo define uma classe `SimpleBot` que registra um *callback* para eventos de mensagens do tipo `RoomMessageText`. Quando uma mensagem que começa com `!ping` é detectada, o bot responde com `pong!`. A função `sync_forever` mantém a escuta ativa, garantindo reatividade contínua. O uso de `add_event_callback` permite modularizar o comportamento do bot com base em diferentes tipos de evento.

Exemplos de funcionalidades típicas de bots Matrix

Tabela 1.11. Exemplos de funcionalidades típicas de bots Matrix

Gatilho	Ação realizada	Finalidade / aplicação
<code>!ping</code>	Enviar resposta <code>pong!</code>	Testar conectividade e funcionamento do bot
<code>!contar</code>	Contabilizar número de mensagens na sala	Coleta de métricas de uso e atividade
<code>!ajuda</code>	Listar comandos disponíveis	Fornecer instruções aos usuários
Mensagem com palavra-chave proibida	Remover ou sinalizar mensagem	Moderação de conteúdo
Evento de horário específico	Enviar lembrete ou notificação	Agendamento de avisos ou atividades programadas
Entrada de novo usuário	Enviar mensagem de boas-vindas	Recepção automática de novos participantes

As funcionalidades listadas na Tabela 1.11 ilustram como bots podem ser adaptados a diferentes necessidades institucionais e comunitárias. A combinação de comandos

simples com ações programadas permite automatizar tarefas rotineiras, reduzir a sobrecarga de moderação manual e oferecer suporte contínuo a usuários. Além disso, a capacidade de interagir com eventos em tempo real torna os bots ferramentas versáteis para notificações contextuais, integração com sistemas externos e reforço de políticas de uso em salas de comunicação.

Bots podem ser estendidos para executar uma ampla variedade de ações. A combinação de `matrix-nio`, estrutura assíncrona e integração com outras bibliotecas Python torna o desenvolvimento acessível e poderoso mesmo em ambientes institucionais.

A seguir uma lista de bots úteis utilizando **matrix-nio**:

- **Bot que responde a comandos simples:** Detecta mensagens iniciadas por comandos prefixados como `!ajuda`, respondendo com instruções úteis. Base para integrações automáticas com usuários.
- **Bot que envia mensagens automáticas em intervalos:** Utiliza tarefas assíncronas (`asyncio`) para enviar lembretes, notificações ou relatórios regulares para uma sala específica.
- **Bot que registra mensagens recebidas:** Armazena localmente o histórico das mensagens em arquivos texto. Serve como base para auditoria, monitoramento ou coleta de dados de uso.
- **Bot que saúda automaticamente novos participantes:** Detecta eventos de entrada (`m.room.member`) e envia mensagens de boas-vindas personalizadas. Útil para experiências de onboarding automatizado.
- **Bot moderador que remove mensagens com conteúdo ofensivo:** Filtra palavras-chave indesejadas e utiliza a API de `redact` para apagar mensagens de forma automática, simulando políticas de moderação.

Os códigos-fonte dos exemplos de automação com `matrix-nio` e `requests` apresentados neste capítulo estão disponíveis publicamente em <https://github.com/atlabufc/matrixjai2025>.

1.6. Desafios e Oportunidades de Pesquisa Relacionados ao Matrix

A maturidade técnica e a crescente adoção do protocolo Matrix não eliminam os desafios ainda presentes em sua evolução. Muitos dos problemas enfrentados em ambientes reais decorrem justamente de sua ambição técnica: ser federado, auditável, seguro e interoperável. Esta seção apresenta um panorama crítico sobre os obstáculos mais relevantes enfrentados atualmente e propõe caminhos promissores para a investigação científica.

1.6.1. Desafios Técnicos e Perspectivas Futuras

A interoperabilidade segura por meio de *bridges* permanece como um dos desafios mais relevantes para o avanço do Matrix. Embora a arquitetura aberta facilite a criação de conectores com outras plataformas, como Slack, IRC e Discord, manter equivalência

semântica e segurança na troca de mensagens entre sistemas heterogêneos é complexo. Questões como preservação de identidade, sincronização de histórico, e compatibilidade com E2EE impõem limites técnicos e operacionais [Cui et al. 2023, Li et al. 2023].

Outro aspecto importante envolve o desempenho e a escalabilidade, especialmente em grandes redes federadas. O custo computacional da replicação de eventos, da resolução de conflitos e do gerenciamento de chaves criptográficas cresce rapidamente em salas com alta atividade ou número elevado de participantes. O próprio modelo do Matrix Event Graph (MEG) apresenta desafios relacionados ao aumento da largura do grafo e à latência de convergência entre servidores [Jacob et al. 2021].

Além disso, ambientes regulados como instituições governamentais, sistemas de justiça ou redes hospitalares impõem requisitos adicionais de conformidade, armazenamento seguro e auditoria de acessos. A adoção plena do protocolo em tais contextos exige aprimoramentos no suporte a políticas de retenção, mecanismos de logging auditável e ferramentas de governança descentralizada.

1.6.2. Desafios em Aberto na Operação de Servidores Matrix

Embora o protocolo Matrix e sua implementação de referência, o Synapse, ofereçam uma base robusta para comunicação federada, a operação de servidores privados ainda envolve uma série de desafios técnicos e estratégicos. Essas limitações afetam tanto a adoção institucional quanto a manutenção de longo prazo em ambientes autônomos.

Um dos principais desafios diz respeito à **escalabilidade vertical e horizontal**. Apesar do suporte à separação de processos via workers, o Synapse ainda depende de um núcleo monolítico que concentra grande parte das operações críticas. A decomposição do servidor em serviços independentes está em andamento, mas requer maior maturidade para facilitar implantações em larga escala com balanceamento eficiente de carga [Synapse 2025].

Outro ponto importante envolve a **observabilidade e gerenciamento operacional**. Embora o Synapse ofereça suporte à exportação de métricas por meio de Prometheus, a configuração é pouco acessível e carece de ferramentas integradas de visualização e alerta. Isso dificulta a análise proativa de desempenho e o diagnóstico de falhas [Foundation 2024].

A **resiliência a falhas federadas** também é uma preocupação. Em ambientes descentralizados, a instabilidade de um servidor federado pode afetar a experiência global de usuários em diferentes domínios. Mecanismos mais robustos de isolamento, cache inteligente e fallback de federação são áreas em que o protocolo pode evoluir.

Do ponto de vista de segurança, permanece o desafio da **atualização contínua de políticas criptográficas**. A gestão segura de chaves, autenticação de dispositivos e proteção contra abusos ainda requer intervenção manual significativa. A automação desses processos, aliada à detecção de anomalias, é um campo promissor para pesquisa e desenvolvimento.

Por fim, o processo de instalação e manutenção ainda apresenta **curva de aprendizagem elevada**, o que limita o uso por organizações com menor infraestrutura técnica. A criação de instaladores simplificados, interfaces gráficas de configuração e documen-

tação contextualizada por casos de uso são aspectos que podem contribuir decisivamente para a ampliação da adoção do Matrix em ambientes governamentais, educacionais e comunitários.

1.6.3. Potenciais Áreas para Inovação e Investigação Científica

O protocolo Matrix apresenta um campo fértil para pesquisa aplicada e desenvolvimento experimental. A otimização da estrutura de replicação MEG é uma das frentes mais promissoras, com investigações voltadas à compactação de grafos, reordenamento eficiente de eventos e resolução distribuída de conflitos com menor custo computacional [Jacob et al. 2021].

Outra área em expansão envolve a interoperabilidade avançada entre mensagens, com foco em modelos de conversão semântica entre protocolos distintos, autenticação cruzada e criptografia ponta-a-ponta mantida mesmo entre domínios heterogêneos [Cui et al. 2023]. Essa abordagem demanda novos modelos formais para tradução de eventos, bem como algoritmos de mediação que garantam integridade contextual das interações.

A segurança distribuída também segue como vetor central de inovação. Pesquisas sobre esquemas aprimorados de cross-signing, autenticação baseada em identidades descentralizadas (DIDs) e delegação criptográfica entre dispositivos podem consolidar o Matrix como padrão de mensageria segura e interoperável em escala global.

Tabela 1.12. Principais áreas de pesquisa e inovação no ecossistema Matrix

Área	Descrição	Objetivos de pesquisa
Replicação e MEG	Otimização do Matrix Event Graph (MEG) em redes de grande escala	Compactação, ordenação eficiente de eventos, resolução de conflitos com menor custo computacional
Interoperabilidade segura	Integração confiável entre Matrix e outras plataformas heterogêneas via <i>bridges</i>	Tradução semântica, preservação de E2EE, autenticação cruzada
Segurança distribuída	Fortalecimento da confiança e autenticação entre múltiplos dispositivos e domínios	Cross-signing, DIDs, delegação criptográfica, gerenciamento de dispositivos
Governança e conformidade	Adaptação do protocolo para ambientes regulados e institucionais	Auditoria de acessos, retenção de dados, logging e governança descentralizada

A Tabela 1.12 sintetiza essas frentes de pesquisa em quatro grandes áreas que refletem tanto os desafios técnicos quanto as exigências institucionais para a consolidação do protocolo Matrix. No eixo da replicação e desempenho, a estrutura do MEG demanda soluções escaláveis para ambientes federados extensos. Já a interoperabilidade segura segue como prioridade diante da diversidade de plataformas legadas. Na dimensão da segurança distribuída, aspectos como autenticação entre dispositivos e identidades descentralizadas se mostram essenciais para contextos sensíveis. Por fim, a governança e

conformidade regulatória emergem como requisitos críticos à adoção institucional, exigindo suporte nativo a políticas de auditoria, retenção e transparência operacional. Tais vetores indicam que o Matrix não é apenas um protocolo em evolução, mas também um campo estratégico para inovação em sistemas distribuídos.

1.7. Conclusão

Este curso proporcionou aos participantes uma compreensão abrangente e crítica sobre o protocolo Matrix, combinando fundamentos teóricos com aplicações práticas. Foram discutidas as bases conceituais e técnicas que sustentam o funcionamento do protocolo, incluindo sua arquitetura federada, os algoritmos de sincronização distribuída utilizados no MEG e os mecanismos de segurança criptográfica baseados nos protocolos Olm e Megolm.

Além dos aspectos conceituais, o curso explorou aplicações práticas do Matrix em cenários de comunicação segura, descentralizada e automatizada. Foram apresentados casos de uso em ambientes institucionais e exemplos de integração com ferramentas externas, incluindo automações via APIs e desenvolvimento de bots. Essas aplicações evidenciam o potencial do protocolo para atender demandas reais de privacidade, controle de dados e interoperabilidade.

Por fim, o curso destacou desafios técnicos ainda em aberto no desenvolvimento do Matrix, que representam oportunidades relevantes para pesquisas de mestrado e doutorado. Foram abordados temas como estratégias de sincronização federada em larga escala, interoperabilidade segura com outras plataformas e estudos de desempenho e escalabilidade em ambientes distribuídos. Ao apresentar essas questões de forma crítica e contextualizada, espera-se incentivar investigações acadêmicas e soluções inovadoras no campo da comunicação descentralizada, segurança digital e soberania tecnológica.

Agradecimentos

Os autores agradecem aos pesquisadores da Universidade Federal do Ceará (UFC), em especial a José G. R. Maia e Marcos D. Ortiz, cujas contribuições técnicas e discussões foram fundamentais para o amadurecimento e escrita deste capítulo.

Referências

- [Albrecht et al. 2023] Albrecht, M. R., Celi, S., Dowling, B., and Jones, D. (2023). Practically-exploitable cryptographic vulnerabilities in matrix. *Proceedings of the IEEE Symposium on Security and Privacy*, pages 164–181.
- [Authority 2024] Authority, L. (2024). Security audit report: vodozemac. Technical report, Least Authority.
- [Bechara and Lechner 2024] Bechara, J. and Lechner, U. (2024). Digital sovereignty and open-source software: A discussion paper. *Digital Policy, Regulation and Governance*.
- [Biström et al. 2024] Biström, M., Adolfsson, K. K., and Stocchetti, M. (2024). Open-source software and digital sovereignty: A technical case study on alternatives to mainstream tools. *Technology and Regulation*.

- [Conduit 2025] Conduit (2025). Conduit: Lightweight matrix homeserver in rust. <https://github.com/matrix-org/conduit>. Acessado em 2025.
- [Cui et al. 2023] Cui, M., Gu, C., and Nan, G. (2023). Crosschat: Instant messaging across different apps on mobile devices. In *Proceedings of the 38th ACM/SIGAPP Symposium on Applied Computing*.
- [Element 2025a] Element (2025a). Element call integration with livekit. Accessed: 2025-05-22.
- [Element 2025b] Element (2025b). Element docker demo. Accessed: 2025-05-22.
- [Element 2025c] Element (2025c). Self-hosting element call. Accessed: 2025-05-22.
- [Floridi 2020] Floridi, L. (2020). The fight for digital sovereignty: What it is, and why it matters, especially for the eu. *Philosophy and Technology*.
- [Foundation 2020a] Foundation, M. (2020a). Docker-jitsi. Accessed: 2025-05-22.
- [Foundation 2020b] Foundation, M. (2020b). Running matrix with jitsi. Accessed: 2025-05-22.
- [Foundation 2023] Foundation, M. (2023). Docker-jitsi. Accessed: 2025-05-22.
- [Foundation 2024] Foundation, M. (2024). Synapse metrics and monitoring. Acesso em 2025-05-16.
- [Foundation 2025] Foundation, M. (2025). An open network for secure, decentralised communication. Acessado em 2025.
- [Government 2020] Government, F. (2020). Tchapp: The official messaging platform for french civil servants. Accessed: 2024-08-25.
- [Jacob et al. 2021] Jacob, F., Beer, C., Henze, N., and Hartenstein, H. (2021). Analysis of the matrix event graph replicated data type. *IEEE Access*, 9:28317–28333.
- [Klare and Lechner 2023] Klare, M. and Lechner, U. (2023). A reference model to strengthen digital sovereignty in companies. In *2023 IEEE International Conference on Digital Ecosystems (ICDE)*.
- [Li et al. 2023] Li, H., Wu, Y., Huang, R., Mi, X., Hu, C., and Guo, S. (2023). Demystifying decentralized matrix communication network: Ecosystem and security. In *2023 IEEE 29th International Conference on Parallel and Distributed Systems (ICPADS)*, pages 260–267.
- [Matrix Foundation 2025a] Matrix Foundation (2025a). Dendrite: Next generation matrix homeserver. <https://github.com/matrix-org/dendrite>. Acessado em 2025.
- [Matrix Foundation 2025b] Matrix Foundation (2025b). Synapse: Reference matrix homeserver implementation. <https://github.com/matrix-org/synapse>. Acessado em 2025.

- [matrix nio 2025] matrix nio (2025). nio: a multilayered matrix client library. Acessado em 2025.
- [Matrix.org 2022] Matrix.org (2022). Megolm group ratchet. <https://gitlab.matrix.org/matrix-org/olm/-/blob/master/docs/megolm.md>. libolm repository.
- [Mautrix 2025] Mautrix (2025). mautrix-python: Matrix client library for bots and bridges. <https://github.com/mautrix/python>. Acessado em 2025.
- [Misra et al. 2025] Misra, S., Barik, K., and Kvalvik, P. (2025). Digital sovereignty in the era of industry 5.0: Challenges and opportunities. *Procedia Computer Science*.
- [Moerel and Timmers 2021] Moerel, L. and Timmers, P. (2021). Data sovereignty and open source governance.
- [Rahman and Wang 2024] Rahman, M. and Wang, Y. (2024). Implementation of dew-inspired matrix-mesh communication protocol. In De, D. and Roy, S., editors, *Dew Computing: The Sustainable IoT Perspectives*, pages 105–115. Springer Nature Singapore.
- [Synapse 2025] Synapse (2025). Synapse: Welcome to the documentation repository for synapse, a matrix homeserver implementation developed by element. Accessed: 2025-01-09.

Capítulo

2

Inteligência Artificial para a educação

Laura O. Moraes, Carla A. D. M. Delgado

Abstract

The belief that new Artificial Intelligence (AI) technologies are important resources for development and improving quality of life is widely spread. Nowadays, much is being said about this topic, specifically regarding the threats and opportunities it brings to education. The impacts of accessible AI on learning practices and resource management in formal education are already noticeable. Furthermore, students' impressions, expectations, and experiences increasingly demand that AI be present in their learning practices. In this Chapter, we will explore some pathways for designing intelligent solutions for the educational field. Examples of AI techniques and their potential applications to well-known educational challenges will be examined, such as personalized feedback generation, the prevention of failure and disengagement, data protection and ethical use of AI in educational contexts.

Resumo

A crença de que as novas tecnologias de Inteligência Artificial (IA) são recursos importantes para o desenvolvimento e a melhoria da qualidade de vida está bastante difundida. Atualmente, muito se fala sobre isso, especificamente sobre as ameaças e oportunidades para a educação. Já são perceptíveis os impactos da IA acessível nas práticas de aprendizagem e de gestão de recursos na educação formal. Além disso, as impressões, expectativas e experiências dos estudantes demandam que a IA esteja presente em suas práticas formativas. Neste Capítulo vamos abordar alguns caminhos para o desenho de soluções inteligentes para a área da educação. Serão exploradas algumas técnicas de IA e suas potenciais aplicações em problemas reconhecidos do contexto educacional - construção de feedback personalizado, prevenção de situações de fracasso e desengajamento, proteção de dados e uso ético de IA no contexto educativo.

2.1. Introdução

A inteligência artificial (IA) tem se consolidado como uma das mais influentes e disruptivas inovações do século XXI, transformando diversos setores da sociedade, dentre eles

a educação. Porém, o uso da IA no contexto educacional não é um fenômeno recente. Desde as primeiras tentativas de automatizar o ensino por meio de sistemas tutoriais inteligentes nas décadas de 1960 e 1970, a aplicação da IA na educação teve momentos de mais ou menos interesse por parte dos atores desse contexto, mas nunca parou de evoluir. Inicialmente voltada para a instrução individual e o acompanhamento do desempenho dos estudantes, a IA tem ampliado seu escopo de aplicação na educação, principalmente em cenários onde há intermediação computacional da situação de aprendizagem ou nos quais grande parte dos dados da situação educativa já está digitalizada. Essa trajetória reflete uma combinação dos avanços no campo tecnológico e das mudanças nos paradigmas educacionais, que passaram a valorizar cada vez mais as pessoas e as situações de aprendizagem, abrindo caminho para uma educação mais equitativa, colaborativa, empoderadora e inclusiva.

Além das especificidades dos contextos tecnológicos e educacionais, outro aspecto a ser considerado quando se fala em educação é a implementação das trajetórias formativas, desde a infância até a plena atuação profissional dos indivíduos, e para além desta. No Brasil, como em vários outros países, a educação básica está amplamente a cargo do Estado e é por ele organizada. A pressão por um processo formativo eficiente em termos de tempo e de custos é também uma demanda válida, porém ortogonal ao entendimento de que a educação deve valorizar o indivíduo e sua integração livre e crítica na sociedade e no mundo. A qualidade requer investimento, e investimento demanda vontade política.

Com a ascensão de tecnologias como aprendizado de máquina, processamento de linguagem natural e, mais recentemente, as inteligências artificiais generativas, novos desafios e possibilidades emergem. A IA passa a ser vista como um dos instrumentos que pode contribuir para atender a essa ampla gama de demandas e expectativas consideradas relevantes atualmente, mas traz riscos que precisam ser entendidos e mitigados. Compreender a história da IA na educação é importante para que possamos refletir criticamente sobre seus usos atuais e futuros, suas implicações pedagógicas e seus impactos. É a partir desse entendimento que nos preparamos para esboçar os futuros projetos robustos de uso de IA na educação.

O restante dessa seção aborda a história da IA na educação. O capítulo segue abordando temáticas relevantes atualmente para viabilizar o uso de IA em contextos educativos. A Seção 2.2 apresenta a metodologia para a criação de modelos preditivos. A Seção 2.3 apresenta os fundamentos da IA Generativa e versa sobre a sua utilização no contexto educacional. Por fim, a Seção 2.4 conclui com os aspectos éticos relacionados ao uso de IA na educação.

2.1.1. Inteligência Artificial na Educação ao longo do tempo

Embora o tema “IA na educação” tenha se tornado muito popular nos últimos anos, devido à popularização do uso das inteligências artificiais generativas (IAs generativas) em meados de 2023, há iniciativas do uso de IA na educação desde os anos 50, quando o foco eram os tutores inteligentes.

A evolução das aplicações da IA na educação é o resultado do contexto de cada tempo, influenciado pela cultura vigente e suas demandas, com expectativas e propostas

calçadas nos recursos tecnológicos então disponíveis. A seguir, um panorama simplificado dos anos 50 até os dias atuais, buscando evidenciar como cada avanço da IA educacional dialoga com os desafios, expectativas e valores da política educacional vigente em cada época. Ao final do capítulo, discutimos as oportunidades e desafios contemporâneos e como esse entendimento nos ajuda a desenhar as futuras aplicações de IA na educação.

2.1.1.1. Anos 50 a 70

A pedagogia ainda seguia modelos tradicionais e autoritários, calcados na distribuição de conteúdo [Cunha 1980, Libâneo 1994]. A proposta tecnológica cunhada na época foi a “instrução programada” – uma espécie de programação da distribuição do conteúdo, baseada na teoria comportamental de Skinner [Skinner 1953], que enfatiza que o comportamento é aprendido e moldado pelas consequências de nossas ações, através, por exemplo, de reforço das ações consideradas “certas” e repressão das “erradas”. Apresentava-se ao aluno um pequeno módulo de conteúdo seguido de um teste. O próximo conteúdo era escolhido de acordo com o desempenho do aluno nesse teste, permitindo uma adequação à velocidade do aluno. A situação de aprendizagem era individual e solitária. O material era cuidadosamente estruturado para minimizar os erros, garantindo que o aluno mantivesse um alto índice de acertos — o que aumentava a motivação e a confiança.

Nos anos 60 surgem movimentos de contestação da ordem política e social que impactam a educação. Em 1970, Paulo Freire publica a obra “Pedagogia do Oprimido” [Freire 1970], influenciando toda a América Latina. Defende-se uma pedagogia dialógica, crítica e emancipadora, com foco na realidade dos estudantes e no papel da educação como transformação social. A visão tecnicista do processo educativo passa a ser criticada. No panorama tecnológico destacam-se os primeiros avanços da IA, incluindo a Conferência de Dartmouth (1956) [McCarthy et al. 1955], onde o termo “Inteligência Artificial” foi cunhado, o desenvolvimento do “Logic Theorist” [Newell et al. 1956], considerado o primeiro programa de IA, e o *chatbot* ELIZA [Weizenbaum 1966]. As expectativas tornaram-se muito altas e, por não serem correspondidas, seguiu-se o “inverno da IA” – período de desânimo e declínio no investimento e interesse na pesquisa em IA, causado por promessas não cumpridas e resultados limitados frente ao entusiasmo inicial.

2.1.1.2. Anos 80 e 90

As práticas pedagógicas são fortemente influenciadas pela Psicologia Histórico-Cultural de Vygotsky [Vygotsky 1984] e pela Epistemologia Genética de Piaget [Piaget 1973], em reação à corrente comportamental dos anos anteriores. Reforça-se o papel da escola como espaço de construção do conhecimento. Destacam-se os modelos cognitivos – representações computacionais da maneira como os seres humanos aprendem, pensam e resolvem problemas. Surgem corretores ortográficos baseados em Processamento de Linguagem Natural. No contexto tecnológico, surgem os sistemas especialistas marcando a “IA prática” (ou *practical AI*, em inglês) – termo que se refere ao uso aplicado da inteligência artificial para resolver problemas concretos do dia a dia, em ambientes como empresas, escolas, hospitais, etc. Nas escolas, aparecem os ambientes de aprendizagem – sistemas

que integram diferentes tecnologias, como gráficos, animação e simulação, criando espaços de aprendizagem interativos. As expectativas sobre a velocidade do aumento do poder computacional sobem, influenciadas pela repercussão do *Deep Blue* – supercomputador desenvolvido pela IBM, famoso por ser a primeira máquina a derrotar um campeão mundial de xadrez [Campbell et al. 2000]. Os jogos eletrônicos educativos ganham popularidade com a chegada do computador pessoal (PC) às residências. No final da década de 90 é criado o primeiro curso de Licenciatura em Computação no Brasil, na Universidade de Brasília.

2.1.1.3. Anos 2000 e 2010

O final dos anos 90 e a transição para os anos 2000 foram tempos muito influenciados pelo pensamento neoliberal. Houve o avanço de políticas educacionais de cunho gerencialista – estratégias e diretrizes adotadas na educação que se inspiram em modelos de gestão do setor privado. Um marco é a criação de sistemas de avaliação em larga escala [Luckesi 2011] como o SAEB [INEP 2023], a Prova Brasil e o teste PISA, o que justificou investimentos em mineração de dados educacionais. No contexto tecnológico, destacam-se as pesquisas na internet e o avanço dos algoritmos de busca. Pairava a crença de que, com a internet, o acesso à informação de qualidade seria gradativamente universalizado. A personalização da aprendizagem volta à cena com o conceito de “aprendizagem adaptativa” – ajuste inteligente, através de algoritmos, do conteúdo do curso de acordo com o desempenho do aluno. Aumenta o interesse em robótica educacional. No contexto tecnológico, é importante ressaltar os avanços de técnicas de reconhecimento de imagens, a popularização das redes sociais e os primeiros passos em carros autônomos e assistentes virtuais como a Siri.

2.1.1.4. Anos 2020 e expectativas para 2030

Questões de equidade e inclusão ganham força na pauta educativa. Ocorre a expansão da pedagogia inclusiva, da educação antirracista, decolonial e dos debates sobre gênero e diversidade. O apelo ao uso de tecnologias digitais nos contextos educativos cresce, já que essas tecnologias estão muito presentes em praticamente todos os contextos sociais urbanos. Destacam-se os ambientes e recursos de aprendizagem distribuídos e ubíquos, o ensino a distância e híbrido, cursos online abertos e gratuitos (MOOCs), gamificação, preditores de desempenho escolar. Emergem discussões sobre o uso ético de dados.

Na contemporaneidade, a pedagogia é marcada por uma pluralidade de correntes de pensamento que dialogam entre si e com os desafios sociais, culturais, políticos e tecnológicos da atualidade. Em particular, a corrente pedagógica das competências tem forte presença em políticas curriculares como a Base Nacional Comum Curricular (BNCC) [Ministério da Educação do Brasil 2017]. A autonomia, o autorrespeito, a ética e o protagonismo do estudante estão valorizados nas correntes pedagógicas contemporâneas, assim como as competências socioemocionais, fortalecendo a relação entre o indivíduo e o seu entorno, incluindo o contexto tecnológico, que passa a suscitar muitos questionamentos e incertezas quanto ao futuro do trabalho e da convivência.

Destacam-se a IA generativa e as discussões sobre sua interferência nas práticas didáticas e na construção de um comportamento acadêmico e profissional ético. As expectativas para o uso das novas tecnologias na educação envolvem tutores dialógicos, assistentes de avaliação de aprendizagem, gestão de padrões de aprendizagem e ecossistemas educacionais – sistemas compostos por diferentes atores, instituições, recursos e tecnologias que interagem de forma integrada para promover o aprendizado ao longo da vida. O conceito parte de uma visão mais ampla e sistêmica da educação, que vai além da escola tradicional.

2.1.2. Desafios atuais e desenho de futuras soluções

As aplicações de Inteligência Artificial (IA) na educação, ao longo do tempo, acompanham as pautas educacionais predominantes em cada período histórico. Recentemente, o advento das IAs generativas — como o ChatGPT — ampliou ainda mais as possibilidades de uso da IA na educação, permitindo a geração automática de textos, resolução de problemas complexos, correções automatizadas e a simulação de diálogos educacionais. Essas tecnologias têm potencial para apoiar o ensino personalizado em larga escala, tornando a aprendizagem mais acessível e interativa. A chegada dessa nova tecnologia na esfera escolar, porém, ameaça as práticas didáticas vigentes. Isso não é novidade no âmbito de tecnologias educacionais ou tecnologias que tiveram impactos na educação, como a calculadora ou a internet.

Compreender as tensões que envolvem a incorporação da inteligência artificial na educação é essencial para o desenvolvimento de aplicações educativas que sejam coerentes, contextualizadas e socialmente relevantes. Esse processo de concepção começa pela análise crítica das demandas educacionais contemporâneas, que se concentram, sobretudo, em três grandes eixos: a melhoria da qualidade da educação, com foco na aprendizagem significativa e no desenvolvimento de competências para o século XXI; o aumento da eficiência dos processos de ensino, visando otimizar o tempo e os recursos disponíveis; e o fortalecimento do engajamento dos estudantes, especialmente em um cenário de alta evasão, exclusão e desmotivação¹. Paralelamente, é fundamental reconhecer as oportunidades que as tecnologias baseadas em IA oferecem atualmente, como a personalização do aprendizado, permitindo trajetórias adaptadas ao ritmo e estilo de cada aluno; a promoção da dialogicidade, por meio de interações mais ricas e responsivas com os sistemas educativos; e o uso de modelos preditivos, capazes de identificar padrões de comportamento e desempenho que podem antecipar dificuldades, apoiando intervenções pedagógicas mais precisas e oportunas por parte dos professores e até dos próprios alunos.

Enquanto esses três pilares – tensões da comunidade escolar, necessidades da educação e potencialidades da IA – são o ponto de partida para o projeto de soluções educativas, é também indispensável a análise dos riscos envolvidos, como dependência excessiva de suporte tecnológico, a perpetuação de desigualdades, vieses algorítmicos e questões éticas relacionadas à privacidade e segurança dos dados dos alunos. Um plano de mitigação desses riscos deve fazer parte de qualquer projeto tecnológico educativo relevante. Além disso, a universalização do ensino e a busca por maior eficiência da educação formal permanecem como metas centrais, sobretudo em contextos de alta vulnerabilidade

¹ Vale notar que a universalização do ensino é uma demanda latente a esses 3 eixos.

social.

A trajetória da inteligência artificial na educação evidencia avanços significativos, mas também demonstra que a tecnologia, por si só, não é a protagonista das soluções para os desafios do campo educacional. Para que seu potencial se concretize de forma efetiva, é imprescindível que esteja alinhado a princípios éticos, amparado por políticas públicas inclusivas e sustentado por investimentos consistentes em infraestrutura e formação docente. Para que seus efeitos sejam significativos e duradouros, a IA deve integrar soluções educacionais amplas, concebidas de forma intersetorial, com foco no ser humano, na equidade e na promoção da justiça social.

2.2. Análise dados educacionais e sucesso escolar - Modelos estatísticos preditivos na educação

Esta seção apresenta a metodologia para a criação de modelos preditivos, trazendo uma abordagem que combina a inferência estatística, visando a interpretabilidade das variáveis envolvidas, e técnicas avançadas de aprendizado de máquina para alcançar o melhor desempenho dos modelos. Os principais recursos bibliográficos desta seção serão os livros [de Carvalho et al. 2024], [James et al. 2023] e [Provost and Fawcett 2016].

2.2.1. Contextualização

A utilização de recursos e sistemas virtuais para apoio ao aprendizado é uma realidade comum em aulas remotas e presenciais. Com isso, é possível criar grandes repositórios de dados educacionais, contendo informações das interações dos alunos com essas plataformas. Além disso, esses repositórios podem ser enriquecidos pela integração com dados provenientes de sistemas acadêmicos, como informações sociodemográficas, histórico de disciplinas cursadas, notas, frequência, e participação dos alunos em atividades acadêmicas, como bolsas e eventos, entre outros dados disponíveis. Esses sistemas produzem informações valiosas para o meio educacional, mas a quantidade de dados gerada torna inviável a análise manual. Um dos maiores desafios das instituições de ensino atualmente é transformar essa multitude de dados em conhecimento e ações que beneficiem alunos, professores e administradores.

A análise de dados educacionais é uma área de pesquisa que abrange desde a coleta dos dados, passando pela análise, até a formulação de conclusões e recomendações de ações para melhorar o aprendizado dos alunos, a distribuição de recursos institucionais, o desenvolvimento de atividades didáticas, a organização dos recursos de aprendizado em sala de aula, a promoção da colaboração entre alunos, entre outros desafios educacionais. É uma área que resulta da interseção de três grandes campos: ciência da computação, estatística e educação. A Figura 2.1 ilustra o ecossistema de criação e utilização dos modelos resultantes da análise de dados educacionais. Neste ecossistema, alunos, professores e gestores acadêmicos geram os dados a serem processados, normalmente através da utilização de ambientes virtuais educacionais (como o Moodle, Google Classroom e Sistemas Acadêmicos internos). Esses dados passam pelo processo de análise, a ser detalhado na Seção 2.2.2. Por fim, o conhecimento adquirido nesse processo é reincorporado ao sistema através de gráficos, dashboards e alertas exibidos aos usuários do ecossistema, que se traduzem em ações e intervenções.

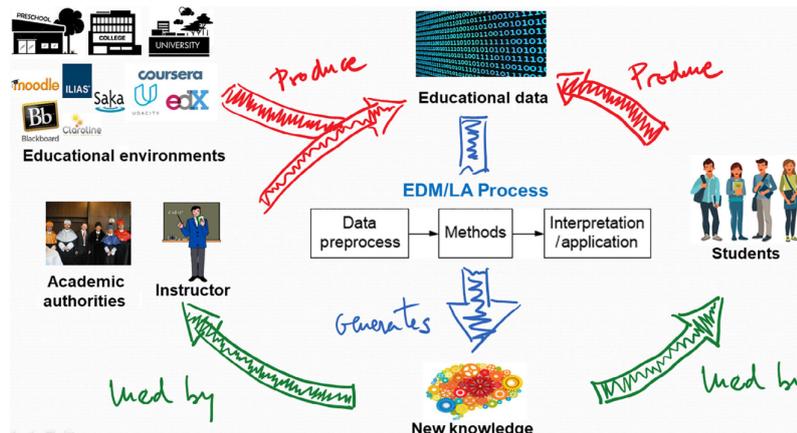


Figura 2.1. Ecossistema da análise de dados educacional. Retirado de: [Romero and Ventura 2020].

A Tabela 2.1 descreve as principais aplicações encontradas nesta área, seguidas de uma breve descrição e as ações e conhecimentos que podem vir a resultar das análises [Romero and Ventura 2020, Ahmad et al. 2024].

Existem duas grandes categorias [Ahmad et al. 2024] de sistemas de apoio à decisão educacional: 1) representacionais ou baseados em conhecimento e 2) baseados em dados. Os métodos baseados em conhecimento utilizam o conhecimento de especialistas para sugerir ações e apoiar a tomada de decisão, sendo mais comuns anteriormente, quando os dados provenientes de sistemas educacionais não estavam tão amplamente disponíveis. Atualmente, os métodos mais utilizados para apoio à decisão envolvem a análise de dados por meio de técnicas de inteligência artificial.

Dentro da área de inteligência artificial, e mais especificamente no campo do aprendizado de máquina (em que modelos são desenvolvidos a partir de dados), os tipos de aprendizado são tradicionalmente classificados em três categorias: aprendizado supervisionado, aprendizado não supervisionado e aprendizado por reforço [Murphy 2012, Sutton and Barto 2018]. Devido às características distintas da inteligência artificial generativa, suas aplicações são, muitas vezes, agrupadas em uma categoria à parte, ressaltando seu caráter disruptivo [Ahmad et al. 2024, Vasconcelos and Clua 2017].

1. **Aprendizado supervisionado:** este tipo de aprendizado predomina na maioria dos trabalhos publicados na área. Os algoritmos dessa categoria necessitam de pares de entrada associados a uma ou mais saídas e têm como objetivo aproximar uma função desconhecida, aprendendo o mapeamento entre as entradas e suas respectivas saídas. Uma vez que a função aproximada é obtida, torna-se possível prever as saídas para novos valores de entrada. No entanto, no contexto educacional, o acesso aos dados são limitados, sendo uma dificuldade consegui-los fora da instituição ao qual o pesquisador está vinculado. Exemplos de aplicações nessa categoria incluem a previsão de notas, a identificação de reprovações e a detecção de evasão escolar.
2. **Aprendizado não supervisionado:** neste tipo de aprendizado, não há pares correspondentes entre entrada e saída, ou seja, os dados utilizados não possuem rótulos

Tabela 2.1. Aplicações, ações e conhecimentos adquiridos em contextos educacionais

Aplicação	Ações e conhecimento adquirido
Análise de teorias educacionais ou estratégias pedagógicas: Entender como determinada teoria educacional ou estratégia pedagógica impacta o aprendizado.	Dinâmicas diferenciadas em sala de aula, reestruturação de currículo e reforma de material pedagógico.
Aprendizado colaborativo e trabalhos em grupos: Entender como o aprendizado colaborativo ou os trabalhos em grupos impactam o aprendizado.	Dinâmicas diferenciadas em sala de aula, estruturação da sala de aula, tipos de exercícios, orquestração de tecnologia em sala de aula.
Análise de currículos acadêmicos: Analisar a estrutura dos programas e os dados curriculares administrativos, verificando a dependência entre as ementas e conhecimentos necessários para melhor aproveitamento das disciplinas.	Aprimoramento do currículo, da qualidade dos cursos e na criação de um panorama do que é ensinado na área.
Sistema de alerta de abandono ou reprovação precoce: Prever o desempenho dos alunos ou alunos em risco de evasão.	Intervenções precoces que permitam a permanência e/ou sucesso do aluno, fatores e comportamentos de risco, feedback personalizado.
Análise de sentimentos e emoções: Entender os sentimentos dos alunos em relação às aulas ou algum recurso didático.	Reforma de material pedagógico, dinâmicas diferenciadas em sala de aula, itens de aprendizado personalizados, intervenção psicológica.
Análise de dados de gamificação: Aplicar técnicas de mineração de dados e visualização às interações dos jogadores em jogos sérios.	Avaliar o conhecimento adquirido dos alunos através de métodos de avaliação não convencionais, identificar dificuldades de aprendizado.
Análise de escrita: Aplicar ferramentas de mineração de texto e análise a dados textuais provenientes de fóruns, chats, redes sociais, avaliações, redações, etc.	Entender o sentimento e dificuldades dos alunos, feedback.
Análise de dados de orquestração: Estudar o efeito da adoção de tecnologias em sala de aula.	Maneiras de inserir a tecnologia em sala de aula de modo a maximizar o aprendizado dos alunos e otimização dos recursos.
Sistemas de recomendação: Recomendação de material de estudo ou material similar.	Estudo personalizado a partir de materiais adaptados à necessidade do aluno, pareamento de alunos que podem se ajudar.
Transfer learning (aprendizado por transferência): Desenvolver modelos que possam ser transferíveis ou aplicados a outros cursos, instituições, etc.	Desenvolver modelos mais robustos a variações, generalizações de teoria de aprendizagem e práticas pedagógicas e permitir o compartilhamento do conhecimento adquirido entre instituições, mitigando problemas de poucos dados ou dados de somente uma instituição, comumente encontrados no contexto educacional.

indicando os resultados esperados. O objetivo principal é identificar padrões em um conjunto de dados, normalmente agrupando observações semelhantes (clustering) ou extraindo os atributos (ou combinações de atributos) mais relevantes para representar esses padrões (redução de dimensionalidade). Exemplos de aplicações nessa categoria incluem a descoberta de perfis de alunos e a recomendação de materiais educacionais.

3. **Aprendizado por reforço:** frequentemente confundido com o aprendizado supervisionado, o aprendizado por reforço se diferencia por não utilizar pares de entrada e saída rotulados. Nesse paradigma, um agente autônomo aprende a tomar decisões ao interagir com um ambiente, buscando maximizar uma função de recompensa ao longo do tempo. Exemplos de aplicações nessa categoria incluem sistemas de aprendizado adaptativo e modelagem de estilos de aprendizagem.
4. **Inteligência artificial generativa:** A IA generativa é um campo recente da inteligência artificial que se dedica ao desenvolvimento de modelos capazes de criar novos conteúdos, como textos, imagens e até músicas a partir de padrões aprendidos nos dados de treinamento. Utiliza arquiteturas de redes neurais avançadas para criação dos modelos. Exemplos de aplicações nessa categoria incluem *feedbacks* personalizados, revisões de texto e *chatbots* para elucidar dúvidas.

As próximas subseções abordam conceitos, definições e aplicações do aprendizado supervisionado no contexto educacional. Os tópicos de aprendizado não supervisionado e aprendizado por reforço não serão discutidos. Devido às suas características inovadoras e ao seu impacto nas aplicações, a inteligência artificial generativa será tratada separadamente em uma seção específica (Seção 2.3).

2.2.2. Metodologia para análise de dados

Nessa seção apresentaremos a metodologia CRISP-DM (*Cross Industry Standard Process for Data Mining*, do inglês, Padrão de Process Inter-Indústrias em Mineração de Dados) [Provost and Fawcett 2016] para desenvolvimento de projetos em Ciência de Dados. A Figura 2.2 detalha cada etapa da metodologia.

1. **Entendimento do problema:** identificação, definição e motivação do problema a ser resolvido. Responde às perguntas:
 - (a) Como uma solução de mineração de dados resolverá este problema?
 - (b) O que se visa alcançar ao realizar essa análise?
 - (c) Quais métricas conseguirão medir uma solução satisfatória para o problema?

Nesta etapa, pesquisa-se na literatura existente, extraindo dos artigos relacionados os modelos existentes e como seu impacto foi medido. É preciso pesquisar métricas quantitativas e é preciso pesquisar também as métricas qualitativas, principalmente o impacto causado pelos modelos em situações autênticas (por exemplo, quando utilizado durante um curso).

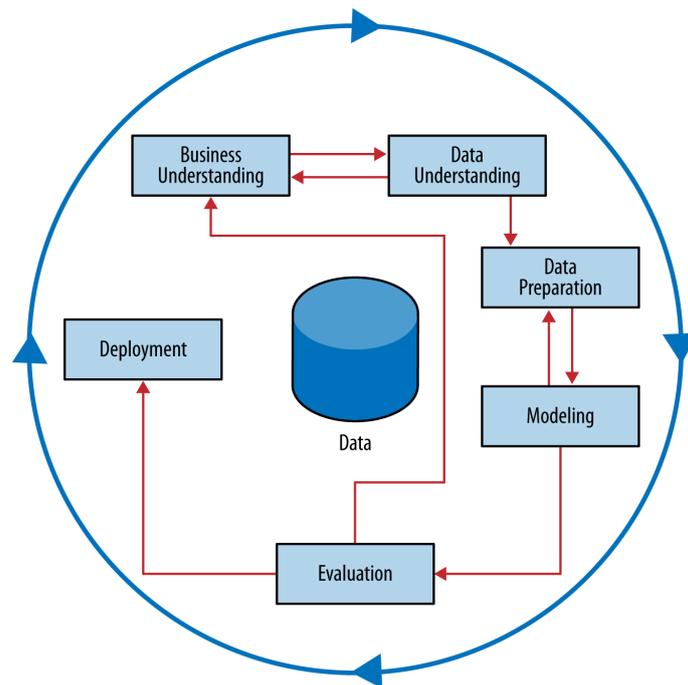


Figura 2.2. Etapas da metodologia CRISP-DM para projetos de Ciência de Dados. Retirado de [Provost and Fawcett 2016].

2. **Entendimento dos dados:** identificação dos dados necessários para resolução do problema proposto. Essa etapa consiste em avaliar o banco de dados, entender o que existe nele, suas limitações e indicar outros dados de interesse que complementem a análise. Responde às perguntas:
- Há dados faltantes? Se sim, é preciso remover os dados faltantes, utilizar alguma técnica de imputação ou optar por deixá-los no banco de dados, utilizando um modelo que saiba lidar com esse tipo de dado.
 - Há dados duplicados? Se sim, é preciso removê-los caso suspeite que os dados sejam errôneos.
 - Os dados se encontram dentro dos limites esperados? É possível verificar para cada atributo numérico os valores mínimos e máximos, verificando se se encontram dentro dos limites conceituais esperados. Por exemplo, se uma variável é a nota, levanta-se uma suspeita se essa nota for maior que 10 vindo de instituto educacional brasileiro.
 - Qual a distribuição dos dados? Para verificar essa pergunta, utiliza-se um histograma para cada variável de interesse. Por exemplo, é possível entender se há uma desigualdade de gênero em determinado curso.
 - Há *outliers*? Outliers são chamados os casos anômalos ou atípicos que podem ser acontecimentos reais no banco de dados ou resultado de erros de medição, entrada ou processamento. Há a possibilidade de remover essas observações uma vez que podem não representar padrão nenhum, distorcer estatísticas e mascarar tendências gerais. No entanto, é preciso ter cuidado ao lidar com esses casos atípicos: essas observações podem representar o comportamento

legítimo de uma minoria (e por serem minoria não constar um volume de dados significativo no banco a ponto de ser considerado um padrão) ou serem relevantes se a análise se refere a encontrar comportamentos fora do comum. Portanto, a decisão de remover, transformar ou analisar separadamente os *outliers* deve ser baseada no entendimento do contexto dos dados e nos objetivos da análise, para evitar a perda de informações relevantes ou a introdução de vieses nos resultados.

- (f) Há dependência entre os atributos? Essa é uma das análises mais importantes no entendimento dos dados. É importante verificar as variáveis com maior poder de predição da variável-alvo (variável que se deseja prever) e quais variáveis estão relacionadas entre si. Isso influencia a escolha dos modelos e a interpretação dos resultados. Por exemplo, se há duas variáveis muito correlacionadas e deseja-se prever uma terceira, um modelo de regressão pode apresentar somente uma das variáveis como relevantes para a previsão. No entanto, no momento de análise dos resultados, é preciso lembrar que ambas as variáveis estão correlacionadas e, portanto, é preciso entender mais a fundo o que está influenciando a previsão. Essa análise é feita com um gráfico de dispersão e métricas de correlação.
3. **Preparação dos dados:** nesta etapa os dados são avaliados através da análise exploratória e formatados para serem utilizados nos modelos. Aqui é verificada a completude dos dados, possíveis inconsistências, além de realizar algumas transformações como limpeza, imputação, normalização, seleção e agrupamento de atributos, redução de dimensionalidade, entre outras. Um exemplo prático será disponibilizado na Seção 2.2.4.
4. **Modelagem:** nesta etapa são criados e aplicados os modelos preditivos que auxiliam as partes interessadas a alcançarem os objetivos determinados na etapa 1 (entendimento do problema). Por isso, é importante a escolha de modelos alinhados com a pergunta que se deseja responder. Como diferentes modelos possuem diferentes hiperparâmetros que influenciam diretamente nos resultados, é necessário identificar os hiperparâmetros existentes em cada modelo e avaliar como sua alteração afeta o resultado. Responde às perguntas:
- (a) Quais modelos testar? A escolha dos métodos e algoritmos deve ser feita de acordo com a pergunta que se deseja responder. Por exemplo, se o objetivo é prever um valor contínuo, como uma nota, é necessário selecionar um modelo adequado para tarefas de regressão.
 - (b) Como testar os modelos? Para avaliar a eficiência e mitigar riscos, a preferência inicial é pela construção de modelos preditivos de forma *offline*, ou seja, utilizando dados históricos, sem interação em tempo real com alunos ou outros participantes. O método *offline* não envolve intervenções, ou seja, não altera os resultados já ocorridos, servindo apenas para simular e avaliar o desempenho dos modelos. Somente após a seleção de um modelo *offline* adequado (definida na etapa de avaliação), ele passa a ser utilizado de forma *online* (em tempo real) na fase de implantação.

5. **Avaliação:** após a criação dos modelos, é preciso avaliar se os resultados estão conforme o esperado. Neste ponto, as métricas e seus valores aceitáveis já foram definidos (etapa 1, entendimento do problema). Portanto, nesta etapa, avaliamos se os modelos definidos na etapa anterior cumprem os requisitos e onde é possível melhorá-los. É também possível reavaliar as métricas definidas anteriormente. Responde às perguntas:
 - (a) Quais métricas são apropriadas para o problema?
 - (b) Como simular um conjunto de dados não vistos para avaliar a generalização do modelo?
6. **Implantação:** nesta etapa final, o modelo é colocado em produção para ser utilizado em tempo real, coletando novos dados e validando o modelo criado.

2.2.3. Exemplo ilustrativo de aplicação da metodologia: Prevenção da evasão escolar

1. **Entendimento do problema:** Uma escola pública busca reduzir os índices de evasão escolar no ensino médio. A gestão escolar suspeita que alguns fatores registrados em sua base de dados podem antecipar comportamentos de risco. O objetivo da análise é prever quais alunos têm maior probabilidade de abandonar os estudos ainda no primeiro semestre, de modo a permitir intervenções pedagógicas antecipadas. *Métricas definidas:* acurácia geral, *recall* para a classe de risco (evadidos) e precisão.
2. **Entendimento dos dados:** A base de dados da escola contém informações históricas dos estudantes: frequência mensal (em %), notas por disciplina, histórico de reprovações, dados socioeconômicos (renda familiar, ocupação dos responsáveis), participação em programas sociais (ex: Bolsa Família), abandonou ou não a escola. Análises exploratórias indicaram:
 - Ausência de dados em cerca de 10% das entradas na variável “renda familiar”.
 - Notas com outliers (alunos com notas consistentemente muito baixas).
 - Correlação moderada entre alta frequência e permanência na escola (scatter plot).
 - Distribuição das faltas com concentração no segundo semestre (histograma).
3. **Preparação dos dados:**

Foram realizadas as seguintes etapas de preparação: remoção de registros duplicados, imputação da “renda familiar” com a mediana segmentada por bairro, normalização das variáveis contínuas (frequência, notas) e codificação de variáveis categóricas (ocupação dos responsáveis).
4. **Modelagem:** Modelos preditivos testados: regressão logística (*baseline*), Random Forest e XGBoost.

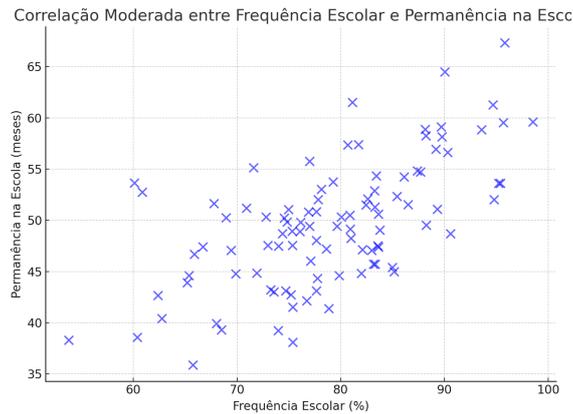


Figura 2.3. Exemplo fictício de correlação moderada entre alta frequência e permanência na escola.

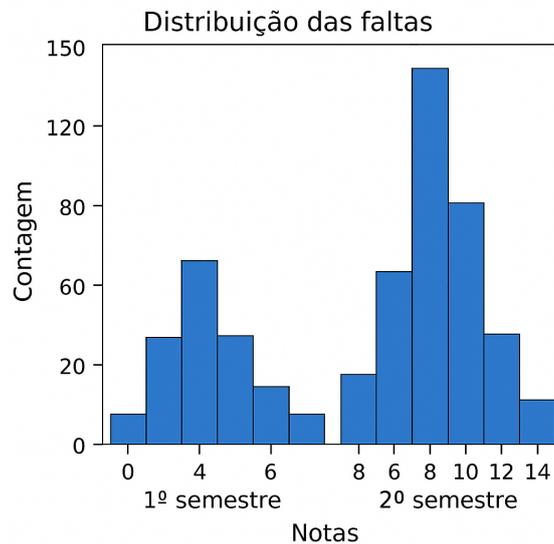


Figura 2.4. Distribuição das faltas com concentração no segundo semestre.

Hiperparâmetros como a regularização da regressão logística, profundidade de árvore e número de estimadores foram ajustados por busca em grade. A avaliação dos modelos foi feita por validação cruzada com *k-fold* ($k=5$).

5. **Avaliação:** Os resultados foram:

- Random Forest: recall de 87% para evasão, acurácia de 83%.
- Regressão logística: menor recall (74%), porém fácil interpretabilidade.
- XGBoost: melhor acurácia (85%), mas maior tempo de execução.

Com base nos objetivos da escola, o modelo Random Forest foi selecionado.

6. **Implantação:** A escola optou por adotar o modelo como ferramenta de alerta precoce, incorporando-o ao planejamento pedagógico semestral.

2.2.4. Análise exploratória de dados

Esta seção apresenta os conceitos de estatística básica necessários para o entendimento dos modelos preditivos e conduz uma análise exploratória, mostrando ao leitor as principais análises a serem realizadas e os pontos de atenção ao realizá-las.

Terminologia Os dados coletados em sistemas virtuais de aprendizado, acadêmicos ou entrevistas são, em sua maioria, organizados em formato de tabela. Essa organização pode ocorrer tanto em bancos de dados relacionais quanto em arquivos de texto estruturados, como aqueles com extensão .CSV (*Comma-Separated Values*).

Nessa estrutura tabular, cada linha representa uma observação individual, que pode ser um caso, um evento ou qualquer outra unidade de análise relevante para o estudo. Por exemplo, em uma base de dados escolar, cada linha pode corresponder a um estudante específico. As colunas da tabela, por sua vez, representam variáveis ou atributos que descrevem diferentes características das observações. No exemplo da base escolar, as colunas poderiam conter informações como frequência, nota em matemática, número de reprovações, renda familiar, entre outros. Assim, a tabela pode ser vista como uma matriz de dados, onde as linhas são chamadas de vetores. Cada vetor contém os valores dos atributos para uma única observação.

Em modelos preditivos, geralmente existe um atributo principal chamado de atributo-alvo (ou variável-alvo), embora em alguns casos possa haver mais de um. O atributo-alvo é aquele que desejamos prever utilizando as demais informações disponíveis no conjunto de dados. Por exemplo, considere o modelo que busca prever o abandono escolar, exemplificado na Seção 2.2.3. Nesse caso, o atributo-alvo é “Abandono”, e o objetivo é estimar se um aluno irá ou não abandonar a escola com base em outras variáveis, como suas notas e o número de reprovações.

Tabela 2.2. Exemplo resumido de dados estudantis

Aluno	Freq. (%)	Nota Mat.	Reprov.	Renda (R\$)	Bolsa Fam.	Abandono
Ana	95	8,5	0	1800	Sim	Não
Bruno	87	6,0	1	2500	Não	Não
Carla	92	7,0	0	1200	Sim	Não
Diego	78	5,5	2	900	Sim	Sim
Eduarda	99	9,5	0	3200	Não	Não

Medidas descritivas:

Medidas de tendência/posição central - média, mediana e moda: representam o meio do conjunto de dados. Como se fosse a escolha de um representante para aquele conjunto.

Medidas de dispersão - desvio padrão, coeficiente de variação, quartis, percentis: indica o quanto os valores variam entre eles. O quão dispersos eles estão ao redor de uma medida central?

Outras sumarizações: mínimo e máximo.

Começaremos essa explicação apresentando um problema. Anotamos a idade de um grupo de pessoas e obtemos: 4, 1 e 10. Qual valor melhor representa esse grupo? O que a gente gostaria é que o erro/diferença entre valor representante e os dados fosse o menor possível, correto? Como calcular? Definindo o representante como \bar{x} , podemos utilizar dois tipos de erros para calcular essa diferença. Qual o melhor representante?

$$Erro = |4 - \bar{x}| + |1 - \bar{x}| + |10 - \bar{x}| \quad \text{ou} \quad Erro = (4 - \bar{x})^2 + (1 - \bar{x})^2 + (10 - \bar{x})^2$$

Medidas clássicas: média, desvio padrão e variação Se escolhermos minimizar o erro quadrático, o melhor representante \bar{x} é a média aritmética, definida por:

$$\bar{x} = \frac{\sum_{i=1}^N x_i}{N}$$

Considere o exemplo da Tabela 2.3, retirado de [Barbetta et al. 2010], em que temos as notas finais relativas a alunos de três turmas. Nos três casos, a média 6,00 é o valor que minimiza o somatório do erro quadrático entre o valor do representante e os valores de cada nota. No entanto, apesar de apresentarem a mesma média, a distribuição das notas em cada turma é bastante diferente. Ao comparar as turmas A e B, por exemplo, observa-se que a turma B possui notas mais extremas, ou seja, há alunos com notas muito baixas e outros com notas muito altas, distantes da média. Isso indica que a turma B é mais heterogênea, com maior dispersão dos dados em relação à média, enquanto a turma A apresenta uma distribuição mais concentrada em torno desse valor.

Já na turma C, a situação é diferente: a maioria dos alunos obteve notas próximas a 7, mas há um único aluno com uma nota extremamente baixa (zero). Esse valor discrepante, chamado de *outlier*, exerce grande influência sobre a média, puxando-a para baixo. Como resultado, a média de 6,00 não representa adequadamente a maioria dos alunos dessa turma, pois a maior parte das notas está acima desse valor. Dessa forma, podemos concluir que a presença de valores muito discrepantes em um conjunto de dados pode distorcer a média, tornando-a um representante inadequado da tendência central dos dados. Por isso, é importante analisar não apenas a média, mas também a dispersão e a existência de possíveis *outliers* ao interpretar conjuntos de dados.

Tabela 2.3. Notas e média para três turmas hipotéticas. Retirado de: [Barbetta et al. 2010]

Turma	Notas dos alunos								Média da turma
A	4	5	5	6	6	7	7	8	6,00
B	1	2	4	6	6	9	10	10	6,00
C	0	6	7	7	8	8			6,00

Além da média, é importante apresentar uma medida complementar que indique o grau de dispersão dos dados em relação a esse representante. Essa medida, chamada

de desvio padrão, mostra o quanto os dados variam em torno da média e, portanto, o quão representativa a média é para o conjunto analisado. O desvio padrão é calculado em alguns passos: primeiro, determina-se a média aritmética dos dados. Em seguida, calcula-se a diferença entre cada valor e a média, elevando cada uma dessas diferenças ao quadrado. Depois, soma-se todos esses valores quadrados e divide-se o resultado pelo número de dados. Por fim, tira-se a raiz quadrada desse resultado para se obter o desvio padrão. Em resumo, o desvio padrão é definido por:

$$\sigma = \sqrt{\frac{\sum_{i=1}^N (x_i - \bar{x})^2}{N}}$$

Ao calcularmos a dispersão dos dados em relação à média, utilizamos o somatório dos erros ao quadrado, ou seja, elevamos ao quadrado a diferença entre cada valor e a média. O resultado desse cálculo é chamado de variância. No entanto, como estamos somando erros quadráticos, a unidade da variância é o quadrado da unidade original dos dados. Por exemplo, se os dados estão em metros, a variância estará em metros ao quadrado. Para que a medida de dispersão volte à mesma unidade dos dados originais e seja mais facilmente interpretada, definimos o desvio padrão como a raiz quadrada da variância. Assim, o desvio padrão expressa a variabilidade dos dados na mesma unidade de medida utilizada inicialmente, facilitando a comparação e a compreensão dos resultados.

Medidas baseadas na ordenação dos dados: mediana, quartis e percentis Se escolhermos minimizar o erro com valor absoluto, o melhor representante \bar{x} é a mediana. Para calculá-la, é necessário, primeiramente, organizar os dados em ordem crescente (ou decrescente). Em seguida, identifica-se o ponto central dessa sequência, de modo que o conjunto seja dividido em duas partes com a mesma quantidade de elementos em cada lado. Se o número de dados for ímpar, a mediana será o valor que ocupa exatamente a posição central da lista ordenada. Nesse caso, haverá o mesmo número de elementos à esquerda e à direita desse valor. Por outro lado, se o número de dados for par, não existe um único valor central. Nessa situação, a mediana é calculada fazendo-se a média aritmética dos dois valores que ocupam as posições centrais da sequência ordenada. Assim, garante-se que o conjunto de dados seja dividido em duas partes iguais, cada uma contendo metade dos elementos. Ainda utilizando como exemplo a turma A da Tabela 2.3, por essa turma ter um número par de elementos (oito), a mediana será a média aritmética dos dois elementos centrais (no caso, 6). Calculando para as três turmas, o valor entre a média aritmética e a mediana serão diferentes somente na turma C, onde a mediana é 7.

De forma semelhante ao cálculo da mediana, os quartis e os percentis também servem para dividir o conjunto de dados em partes iguais, facilitando a análise da distribuição dos valores. Os quartis segmentam os dados em quatro partes iguais, cada uma correspondendo a 25% do total. O primeiro quartil (Q1) indica o valor abaixo do qual estão 25% dos dados, o segundo quartil (Q2) corresponde à mediana (50%), e o terceiro quartil (Q3) marca o ponto abaixo do qual estão 75% dos dados. Já os percentis dividem o conjunto de dados em cem partes iguais, cada uma representando 1% da amostra. Por exemplo, o 30º percentil indica o valor abaixo do qual estão 30% dos dados. Para avaliar o grau de dispersão dos dados em relação à mediana, costuma-se calcular os quartis. Dessa

forma, é possível visualizar como os dados estão distribuídos em intervalos de 25%, identificando concentrações, dispersões e possíveis assimetrias. Essas medidas de posição são frequentemente representadas de forma gráfica por meio do boxplot (ou diagrama de caixa), que mostra, de maneira visual, a distribuição dos dados, destacando os quartis, a mediana e possíveis valores discrepantes (*outliers*).

2.2.4.1. Entendimento e preparação dos dados

De acordo com [Ahmad et al. 2024], as principais atividades desenvolvidas na área de inteligência artificial aplicada à educação envolvem a avaliação do desempenho dos estudantes, a previsão de retenção e a identificação de possíveis casos de evasão escolar. Nesse contexto, os modelos de previsão utilizam, em sua maioria, dados relacionados a fatores socioeconômicos (como gênero, idade e situação profissional), desempenho acadêmico prévio (por exemplo, média geral das notas e resultados em exames de admissão), atividades na internet (como o uso de ambientes virtuais de aprendizagem - AVA) e fatores psicológicos ou de saúde (incluindo qualidade do sono e vida social). Além desses, alguns estudos também consideram avaliações específicas em disciplinas, hábitos de estudo e indicadores de saúde mental dos estudantes [Rodrigues et al. 2025].

Como explicado no item 2 da Metodologia (Seção 2.2.2), a análise de dependência dos atributos entre si e dos atributos com a variável-alvo é uma das mais importantes, pois já pode fornecer *insights* sobre os dados e guiar as próximas etapas da metodologia. Nesta etapa, o método mais utilizado é o índice de correlação com o apoio visual de um gráfico de dispersão.

Correlação Dizemos que duas variáveis estão positivamente correlacionadas quando o aumento em uma delas tende a ser acompanhado pelo aumento na outra. Por outro lado, a correlação negativa ocorre quando o aumento em uma variável está associado à diminuição da outra. É importante ressaltar que o conceito de correlação refere-se apenas à existência de uma relação numérica entre duas variáveis, sem implicar, necessariamente, em uma relação de causa e efeito ou na existência de alguma estrutura subjacente [Barbetta et al. 2010]. Cabe ao pesquisador analisar os fundamentos teóricos que poderiam explicar a correlação observada e investigá-los por meio de hipóteses. Assim, a correlação funciona como uma ferramenta auxiliar na formulação e verificação de hipóteses baseadas em fundamentos teóricos, sendo normalmente utilizada em estudos exploratórios.

Existem diferentes maneiras de calcular a correlação, sendo as três mais utilizadas as de Pearson, Spearman e Kendall [Okoye and Hosseini 2024]. A correlação de Pearson caracteriza-se principalmente por medir a relação linear entre duas variáveis. Isso significa que, se um aumento x em uma variável corresponde a um aumento y em outra, espera-se que esses incrementos sejam proporcionais e constantes. Por exemplo, se a primeira variável aumenta em $2x$, a segunda deveria aumentar em $2y$. A correlação de Pearson pode ser calculada pela equação 1 e sua relação linear pode ser visualizada em um gráfico de dispersão, onde os pontos tendem a se alinhar formando uma reta. Detalhes sobre a construção da equação 1 podem ser encontrados em [Barbetta et al. 2010].

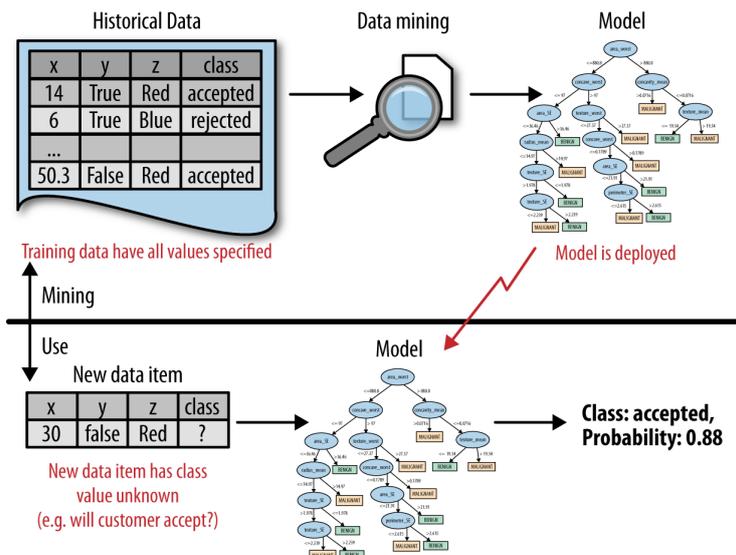


Figura 2.5. Comparação entre o processo de criação de um modelo preditivo e utilização de um modelo implantado. Retirado de: [Provost and Fawcett 2016]

$$r = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^n (x_i - \bar{x})^2} \sqrt{\sum_{i=1}^n (y_i - \bar{y})^2}} \quad (1)$$

Durante o minicurso haverá uma parte prática em que a análise exploratória será conduzida, compartilhada com os participantes através do repositório do GitHub: <https://github.com/MachineTeachingEdu/JAI2025-IA-Educacao.git>.

2.2.5. Aprendizagem estatística e modelagem

Após entendimento dos dados nas seções anteriores, essa seção introduz o leitor à criação de modelos preditivos. Para isso, são apresentados e definidos os conceitos principais para a sua criação e utilização correta.

2.2.5.1. Conjunto de treino, teste, sobreajuste e generalização

A criação de um modelo preditivo normalmente visa sua utilização no futuro, conforme novos dados são obtidos. Por exemplo, ao criar um modelo preditivo para prever a nota ou a evasão de alunos com dados históricos, visa-se utilizar esse modelo com novos alunos, que ainda estão cursando e, portanto, não temos o resultado final ainda. O processo de criação e utilização de um modelo preditivo está ilustrado na Figura 2.5. Na parte superior da figura, dados históricos são explorados para produzir o modelo. Note que os dados históricos possuem um atributo-alvo chamado “class” que será utilizado para criar o modelo que melhor se adapta aos dados (menor erro fora da amostra). Na parte inferior da figura, o modelo é aplicado um dado novo que ainda não possui o atributo-alvo e, portanto, gostaríamos de prevê-lo. O modelo prevê o valor da classe e a sua probabilidade de ocorrência.

No entanto, se criarmos um modelo e verificarmos seu funcionamento utilizando

somente um conjunto de dados, muito provavelmente criaremos um modelo que não tenha aprendido um padrão sobre o comportamento dos alunos e sim aprendido sobre aqueles dados. Uma analogia comum é pensar em um aluno estudando para uma prova nova a partir de uma prova antiga. O aluno pode aprender o conteúdo ou somente aprender as respostas da prova. No primeiro caso, em uma prova nova, espera-se que ele acerte as questões. No segundo, como não terá aprendido o conteúdo, ele provavelmente não saberá responder caso a pergunta mude. O mesmo acontece nos modelos preditivos de aprendizado de máquina. O modelo pode aprender o padrão (conteúdo) ou aprender somente o que acontece naqueles dados.

Idealmente, usaríamos um conjunto de dados só para o treinamento e criação do modelo e outro diferente para verificar seu funcionamento, similar a utilização de uma prova antiga e uma prova nova para estudo. No entanto, frequentemente só temos um conjunto de dados, como podemos então maximizar nossas chances de criarmos um modelo que aprenda o padrão e não somente o que acontece nos dados? Separaremos o conjunto único que temos em dois: o primeiro fará o papel da prova antiga, sendo utilizado na treinamento e ajuste do modelo. Chamaremos esse de conjunto de treino. O segundo, por mais que tenha vindo da mesma prova, será escondido e não será utilizado em momento algum do processo, somente no final, onde seu propósito será somente para avaliar se o modelo é capaz de aprender o padrão, simulando um prova nova. Este será chamado de conjunto de teste. A Tabela 2.4 apresenta as definições dos conceitos fundamentais em aprendizado de máquina.

Mas como saberemos que o modelo criado não generaliza e possui sobreajuste? Existem técnicas para identificar e evitar o sobreajuste. A Tabela 2.5 resume de maneira simplificada os principais sinais de alerta para identificação do sobreajuste e do subajuste a partir dos resultados de erro dentro e fora da amostra e da característica do modelo. Sobre as técnicas para evitar, elas, em sua maioria, dependem do tipo de modelo escolhido.

No contexto educacional, frequentemente lidamos com dados que possuem uma dependência temporal. Nesses casos, a divisão entre treino e teste deve respeitar a ordem cronológica dos eventos, evitando o vazamento de informações do “futuro” para o “passado”. Por exemplo, suponha uma coleta de dados realizada entre 2020 e 2022 ao longo de cinco semestres. O modelo pode ser treinado e testado utilizando cinco conjuntos de dados distintos, onde a cada rodada o conjunto de treinamento é incrementado com mais um semestre e o semestre subsequente é utilizado como conjunto de teste, conforme ilustrado na Figura 2.6. Esse procedimento, conhecido como validação cruzada para séries temporais, garante que, em cada iteração, o conjunto de teste sempre contenha dados posteriores aos do conjunto de treinamento, respeitando a sequência temporal e evitando avaliações tendenciosas do desempenho do modelo. Dessa forma, o modelo que apresentar melhor desempenho nas quatro situações é considerado o mais adequado, pois demonstra robustez e capacidade de generalização para diferentes cenários ao longo do tempo.

Tabela 2.4. Definições de conceitos fundamentais em aprendizado de máquina.

Termo	Definição
Conjunto de treino	Conjunto de dados que será utilizado para realizar o treino e ajuste do modelo.
Conjunto de teste	Conjunto de dados que não será utilizado durante o processo de construção do modelo, somente no final, para avaliar se o modelo aprendeu o padrão.
Erro dentro da amostra	Erro medido no conjunto de treino. Utilizaremos essa métrica para avaliar se o modelo conseguiu aprender dentro dos próprios dados. Continuando nossa analogia, seria verificar se o aluno consegue acertar as questões na prova antiga, na mesma prova que ele estudou.
Erro fora da amostra	Erro medido no conjunto de teste. Essa métrica é a mais importante, visto que verifica se o modelo é capaz de aprender um padrão (conteúdo) e não somente o que aconteceu em um conjunto particular de dados. Continuando a analogia, seria verificar na simulação de uma prova nova, se o aluno é capaz de acertar as novas questões.
Generalização	Capacidade do modelo de identificar o padrão dos dados e acertar a resposta em dados não vistos no treinamento do modelo. Só conseguimos ter certeza da generalização do modelo na prática, mas utilizamos o erro fora da amostra para fazer essa estimativa.
Sobreajuste	Um ditado popular nesta área diz que “se você torturar os dados por tempo suficiente, eles vão confessar” [Abu-Mostafa et al. 2012, Provost and Fawcett 2016]. O sobreajuste acontece quando “torturamos” demais os dados, ou seja, encontramos não somente um padrão geral, mas padrões falsos (que não generalizam) que se aplicam somente ao conjunto de dados em particular.
Subajuste	O subajuste é quando não encontramos padrão nenhum nos dados. Normalmente indica que não estamos usando um modelo apropriado (depende da complexidade do padrão a ser encontrado e da quantidade de dados que temos para encontrar esse padrão) ou não o configuramos corretamente. É mais fácil de identificar e contornar que o sobreajuste.

2.2.5.2. Modelos

Nesta subseção, explicaremos superficialmente três diferentes modelos comumente utilizados no contexto educacional. Detalhes sobre o funcionamento desses modelos podem ser encontrados em [James et al. 2023, Abu-Mostafa et al. 2012]:

1. Regressão linear: modelos lineares convencionais assumem que a variável dependente segue uma distribuição Normal e é modelada diretamente como uma combi-

Tabela 2.5. Sinais de alerta para identificação de sobreajuste e subajuste

Característica	Sobreajuste	Subajuste
Erro no treinamento (dentro da amostra)	Baixo	Alto
Erro no teste (fora da amostra)	Alto	Alto
Diferença do erro no treino e no teste	Grande	Pequena
Complexidade do modelo	Alta (modelo muito complexo, normalmente com muitos parâmetros)	Baixa (modelo muito simples, normalmente com poucos parâmetros)
Capacidade de generalização	Ruim	Ruim

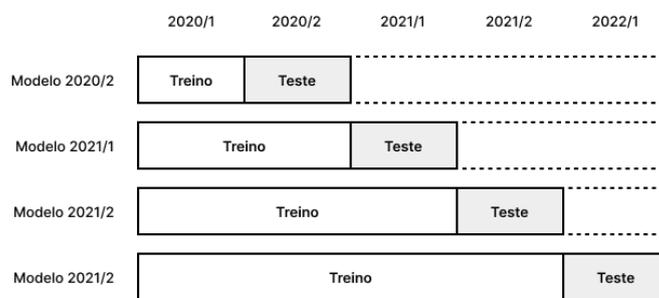


Figura 2.6. Divisões entre dados de treinamento e teste em cada modelo

nação das variáveis preditoras. Essa relação pode ser escrita na forma:

$$E(Y_i) = \mu_i = x_i^T \beta = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \dots + \beta_n x_{in} \quad (2)$$

Para estimar os valores de β , normalmente utiliza-se o método dos mínimos quadrados. Este método consiste em achar os valores de β que formam a reta (quando trabalha-se com muitas variáveis, essa reta é chamada de hiperplano) cuja diferença entre os pontos da reta (hiperplano) e os pontos observados possua o menor erro quadrático possível. Por ser um modelo simples, as principais maneiras de se evitar o sobreajuste são através da regularização e da seleção de variáveis [James et al. 2023].

2. Regressão logística: em situações em que a variável resposta é binária e, portanto, não podemos assumir a normalidade dos dados — como no caso de abandono ou não de um aluno em uma disciplina — é recomendada a utilização de uma importante extensão dos modelos lineares tradicionais: os modelos lineares generalizados (GLMs). Os GLMs permitem modelar variáveis resposta que seguem distribuições diferentes da normal, como distribuições binomial, de contagem e de proporções, ampliando consideravelmente as possibilidades de análise estatística [Dobson and Barnett 2008]. No caso específico de uma variável resposta binária

ria, o modelo mais adequado dentro do *framework* dos GLMs é a regressão logística [James et al. 2023], que utiliza a função de ligação *logit* para relacionar o preditor linear à probabilidade do evento de interesse. No caso da regressão logística, em vez de modelarmos diretamente a variável resposta, modelamos a probabilidade de que a resposta pertença a uma determinada categoria (por exemplo, a categoria “abandono”). Tal relação pode ser escrita como:

$$Pr(Y = abandono|X) = p(X) = \frac{e^{\beta_0 + \beta_1 X}}{1 + e^{\beta_0 + \beta_1 X}} \quad (3)$$

onde β representa os coeficientes da função a serem estimados e X as variáveis preditoras. Após alguma manipulação, temos que a função logística é dada por

$$\log\left(\frac{p(X)}{1 - p(X)}\right) = \beta_0 + \beta_1 X \quad (4)$$

e tem uma interpretação natural como o logaritmo da razão entre a probabilidade de um evento ocorrer e a probabilidade de não ocorrer. Assim como na regressão linear, as principais maneiras de se evitar o sobreajuste são através da regularização e da seleção de variáveis [James et al. 2023].

3. **Árvore de Decisão:** é um método sistemático para segmentar dados, particionando o espaço em regiões menores de forma sequencial, utilizando um atributo por vez. A escolha do atributo em cada divisão prioriza aquele que proporciona a melhor separação, tornando as regiões resultantes mais homogêneas em relação à variável-alvo (ou seja, com menor variância em relação a essa variável) [James et al. 2023, Provost and Fawcett 2016]. Esse método pode ser aplicado tanto em tarefas de classificação quanto de regressão. Por sua natureza, a árvore de decisão pode continuar dividindo o espaço até que não restem regiões com erro, o que a torna suscetível ao sobreajuste. Para evitar esse problema, os principais mecanismos utilizados são a poda e a parada precoce. Na poda, a árvore é inicialmente construída até sua máxima profundidade (ficando altamente especializada e, muitas vezes, sem erros nos dados de treinamento). Em seguida, são removidos ramos que não contribuem significativamente para a capacidade de generalização do modelo, ou seja, cuja remoção não aumenta substancialmente o erro do modelo. Já a parada precoce consiste em definir critérios restritivos durante a construção da árvore—como um número mínimo de amostras em cada nó ou um limite máximo de erro tolerado—fazendo com que o crescimento da árvore seja interrompido antes que ela se torne excessivamente especializada [de Oliveira F. Moraes 2016]. No entanto, a construção de uma árvore de decisão é altamente sensível à forma como os dados são divididos em conjuntos de treino e teste. Pequenas alterações no conjunto de treino podem resultar em árvores significativamente diferentes [James et al. 2023]. Para contornar esse problema e aumentar a robustez dos modelos, foram propostas algumas variações do algoritmo de árvore de decisão, como Florestas Aleatórias, Bagging e Boosting. Embora não entremos em detalhes sobre as diferenças entre esses métodos, todos eles se baseiam no princípio do aprendizado em conjunto (*ensemble learning* em inglês). Em vez de construir apenas uma única árvore, esses métodos

constroem múltiplas árvores simples, seja por meio da aleatorização das observações, dos atributos ou pela construção sequencial de árvores que corrigem os erros das anteriores. Assim, o resultado final se torna mais robusto, pois não depende das decisões de uma única árvore, reduzindo a sensibilidade a pequenas variações nos dados de treino.

2.2.5.3. A complexidade dos modelos e sua interpretabilidade

Modelos que possuem uma grande variedade de parâmetros a serem ajustados são considerados mais flexíveis ou complexos e têm a tendência a produzir melhores resultados em aplicações gerais. No entanto, em educação, onde os conjuntos de dados são mais escassos, diversos e segregados entre instituições [Hellas et al. 2018], é difícil coletar um conjunto de dados com um número suficientemente grande de observações que justifique a utilização de métodos complexos (como redes neurais profundas, por exemplo) sem causar sobreajuste. Além disso, há uma discussão vigente sobre a troca entre interpretabilidade e complexidade do modelo [James et al. 2023]. Modelos complexos tendem a ser menos interpretáveis, ou seja, é mais difícil entender (ou interpretar) como as variáveis influenciam o resultado final. No contexto educacional, é fundamental que os tomadores de decisão (normalmente professores e gestores, mas podem ser também os próprios alunos) não recebam somente a previsão como resultado, mas consigam entender o processo decisório do modelo para tomar tal decisão. Por exemplo, não é informativo a um aluno ou professor simplesmente informar que um aluno está propenso a abandonar um curso ou tirar uma nota baixa, mas sim quais dos fatores observados indicam que o aluno possa ter tal desempenho para que o tomador de decisão possa tomar ações que remediem o resultado previsto.

[Rodrigues et al. 2025] apresenta os algoritmos mais utilizados e com melhores resultados para previsão de desempenho em universidades reportados por diversos artigos. A pesquisa mostrou que o algoritmo de Floresta Aleatória foi o mais utilizado, mostrando a preferência por algoritmos complexos mas que ainda oferecem interpretabilidade.

Técnicas mais recentes chamadas de Inteligência Artificial Explicável (XAI ou eXplainable Artificial Intelligence, em inglês) como LIME [Ribeiro et al. 2016] e SHAP [Lundberg and Lee 2017] estão permitindo quebrar essa troca, oferecendo explicações e interpretabilidade mesmo em modelos complexos [Belle and Papantonis 2021]. No entanto, a dificuldade no contexto educacional continua ainda na coleta de um conjunto grande de dados para justificar a utilização dessas técnicas, o que só é conseguido normalmente através de cursos online abertos e massivos (MOOC), coletas longitudinais (ao longo de muitos anos) e colaborações institucionais.

2.2.6. Avaliação e interpretação dos resultados

Após a criação dos modelos preditivos, é preciso tirar *insights* dos resultados e transformá-los em ações, apresentando-os aos diferentes interessados: gestores acadêmicos, professores e alunos. Cada interessado utilizará e interpretará o resultado de uma maneira diferente, tirando proveitos distintos. Essa seção apresenta as principais técnicas de apresentação de resultados de acordo com o tipo de interessado.

2.2.6.1. Matriz de confusão e suas métricas derivadas

A matriz de confusão, ou tabela de contingência, é o instrumento mais completo para avaliar o resultado dos modelos preditivos quando estamos trabalhando com um conjunto pequeno de classes, como por exemplo, no caso da previsão de evasão. A matriz de confusão mostra a contagem ou a proporção de previsões corretas e incorretas feitas pelo modelo preditivo, comparando-as com as classes verdadeiras dos dados. Cada coluna representa a classe verdadeira (rótulo real) de uma instância de dado e cada linha representa a classe prevista pelo classificador, conforme apresentado na Tabela 2.6.

Tabela 2.6. Matriz de confusão para o caso de classificação binária

Valor predito	Valor real	
	$Y = 0$	$Y = 1$
$c(X) = 0$	VN (verdadeiro negativo)	FN (falso negativo)
$c(X) = 1$	FP (falso positivo)	VP (verdadeiro positivo)

Note que a matriz de confusão deve ser construída a partir das classificações no conjunto de teste para que as estimativas das medidas de desempenho não sejam subestimadas. Algumas das métricas definidas com base em uma matriz de confusão são apresentadas na Tabela 2.7. Pode-se utilizar tais métricas para a escolha do melhor classificador. Em muitos casos, busca-se maximizar simultaneamente a sensibilidade e a especificidade, ou seja, um equilíbrio entre os erros de classificação entre as duas classes. Em outros casos, pode-se ter interesse em maximizar a estatística F1 ou outra métrica obtida a partir da matriz de confusão [Izbicki and dos Santos 2020].

Outra forma de analisar o desempenho das classificações de modelos é usando a curva ROC (do inglês, *Receiver Operating Characteristic*), uma representação gráfica que permite avaliar o desempenho de um classificador binário em diferentes valores de K. A curva ROC é construída plotando a taxa de verdadeiros positivos (Sensibilidade) no eixo vertical e o complementar da taxa de falsos positivos (1 - Especificidade) no eixo horizontal. Cada ponto na curva ROC representa um determinado limiar de classificação aplicado ao modelo. A curva ROC permite a comparação visual da capacidade preditiva de diferentes modelos ajustados. Curvas com maior área abaixo dela indicam melhor desempenho do modelo, o valor da área nos leva a mais uma métrica, conhecida por AUC (Area Under the Curve, em inglês), que reflete um desempenho mais elevado à medida que se aproxima do valor 1. Além disso, a curva ROC é usada para definir o ponto de corte do classificador que maximiza a sensibilidade e a especificidade [James et al. 2023]. A fim de definir um modelo preditivo para classificação, precisamos escolher um valor para o limiar de decisão K, acima do qual classificamos uma observação com a classe de interesse. Para isso, podemos testar diferentes cortes K entre 0 e 1 e selecionar aquele que tem melhor desempenho, segundo algum critério estabelecido usando os valores obtidos de uma matriz de confusão.

Tabela 2.7. Métricas de avaliação: descrição e fórmulas

Métrica	Fórmula
Acurácia: do total de observações, quantas foram classificadas corretamente.	$A = \frac{VP+VN}{VP+VN+FP+FN}$
Sensibilidade/Recall: das observações em que o valor real é 1 ($Y = 1$), quantas foram classificadas corretamente ($c(X) = 1$).	$S = \hat{P}(c(X) = 1 Y = 1) = \frac{VP}{VP+FN}$
Especificidade: das observações em que o valor real é 0 ($Y = 0$), quantas foram classificadas corretamente ($c(X) = 0$).	$E = \hat{P}(c(X) = 0 Y = 0) = \frac{VN}{VN+FP}$
Valor preditivo positivo/Precision: das observações classificadas como 1 ($c(X) = 1$), quais são de fato da classe 1 ($Y = 1$).	$VPP = \hat{P}(Y = 1 c(X) = 1) = \frac{VP}{VP+FP}$
Valor preditivo negativo: das observações classificadas como 0 ($c(X) = 0$), quais são de fato da classe 0 ($Y = 0$).	$VPN = \hat{P}(Y = 0 c(X) = 0) = \frac{VN}{VN+FN}$
Estatística F1: média harmônica entre o Recall e a Precision.	$F1 = \frac{2}{\frac{1}{S} + \frac{1}{VPP}}$

O contexto educacional, assim como outros contextos sensíveis que lidam com dados pessoais, como saúde e justiça, apresenta um desbalanceamento natural não apenas na quantidade de dados por classe, mas também em relação a quais classes são de maior interesse. Por exemplo, no caso da previsão de evasão ou reprovação, há maior interesse em identificar quem está em maior risco de evasão ou reprovação do que em identificar quem está em menor risco. Isso nos leva a refletir sobre quais tipos de erros são mais prejudiciais, pois, ao utilizar métricas simples de classificação não se faz distinção entre erros falsos positivos e falsos negativos, assumindo tacitamente que ambos os erros são igualmente importantes, quando, na prática, esse raramente é o caso [Provost and Fawcett 2016]. Por exemplo, é mais grave errar ao afirmar que um aluno com baixo risco de abandonar o curso irá abandoná-lo, ou ao afirmar que um aluno com alto risco de abandono não irá abandonar? Esses são, normalmente, tipos de erros com custos muito diferentes, porque as classificações incorretas têm consequências de diversas gravidades.

Considerando que os resultados dos modelos podem ser utilizados para orientar ações e intervenções por parte de professores e gestores, no primeiro caso, gastaríamos recursos extras com um aluno que provavelmente não precisaria de intervenção, sem causar grandes prejuízos a ele. No entanto, no segundo caso, deixaríamos de intervir em um aluno que provavelmente se beneficiaria da ação, podendo comprometer seu desempenho ou permanência. Portanto, é fundamental refletir sobre essas questões ao trabalhar com modelos preditivos na educação, ajustando as métricas e os pontos de corte de modo que reflitam esse desequilíbrio e priorizem intervenções mais eficazes.

O desafio de escrever sobre este tema é que cada situação é diferente, não havendo uma solução única ou “correta” que possa ser ditada [Provost and Fawcett 2016].

2.2.6.2. Apresentação dos resultados

As métricas são indicadores importantes do poder preditivo de um modelo. No entanto, principalmente no contexto educacional, os interessados (alunos, professores e gestores) estão interessados não somente na previsão (por exemplo, o risco de abandono ou reprovação) mas também em como mitigar esse risco. E, inspecionando alguns outros fatores dos modelos, conseguimos algumas informações sobre os comportamentos preditores do desfecho de interesse. Por exemplo, em uma regressão linear ou logística, é comum inspecionarmos os coeficientes ajustados dos atributos para entender aqueles de maior relevância, através de sua magnitude e p-valor. Já em uma árvore de decisão, a interpretação é facilitada, pois o modelo pode ser representado visualmente por um diagrama, permitindo que suas decisões sejam compreendidas como um conjunto de regras lógicas.

Ainda, para a utilização apropriada dos modelos e promover a auto-avaliação, auto-reflexão e consciência situacional nos contextos educacionais, *dashboards* (ou painéis, em português) podem ser utilizados, organizando a informação em gráficos informativos direcionados às pessoas de interesse.

2.3. Inteligência artificial generativa para alunos e professores

A inteligência artificial generativa, mais conhecida através de seus modelos de Linguagem de Larga Escala (*Large Language Models* em inglês ou LLMs) ganharam destaque nos últimos anos por sua facilidade de interação e respostas coerentes [Brown et al. 2020]. Na educação em computação, modelos como ChatGPT, Llama, Gemini, Sabiá, entre outros, têm sido alvos de pesquisa sobre a sua capacidade de responder corretamente questões [Denny et al. 2024], prover explicações de código para alunos [Denny et al. 2024, Nam et al. 2024] e professores [Hassany et al. 2024]; apoiar os alunos na escrita de código [Kazemitabaar et al. 2024, Nam et al. 2024] e atuar como um monitor, elicitando dúvidas [Liffiton et al. 2024]. Sendo um campo de pesquisa incipiente, a literatura tem pesquisado e perguntado que tipo de interações são percebidas como mais úteis pelos alunos [Denny et al. 2024, Liffiton et al. 2024], quais produzem resultados mais acurados [Liffiton et al. 2024, Nam et al. 2024] e quais conseguem uma retenção maior do conhecimento [Kazemitabaar et al. 2024]. Essas pesquisas apontam que a utilização de ferramentas baseadas em LLMs na sala de aula pode impulsionar a criação de experiências personalizadas para os alunos [Kasneci et al. 2023], atuando como um agente capaz de identificar a dificuldade de cada aluno e prover sugestões de melhorias e comentários sobre os códigos escritos.

Esse cenário torna promissora a ideia de utilizar LLMs para oferecer suporte aos alunos e professores. No entanto, há preocupações sobre a possível dependência excessiva dos estudantes em relação a esses modelos, especialmente quando são usados para gerar respostas completas [Becker et al. 2022]. Uma descrição simplificada da ameaça percebida é a seguintes: até recentemente, a tarefa de articular ideias com base em fontes selecionadas — seja para produzir textos, seja para resolver problemas matemático-científicos — era amplamente valorizada como estratégia eficaz para o desenvolvimento de competências fundamentais da educação básica. Ao interagir com diferentes fontes, os estudantes aprendiam a identificar informações relevantes, distinguir fatos de opiniões, avaliar a confiabilidade dos conteúdos e organizar o conhecimento de forma coerente.

Essa prática fomenta habilidades como pensamento crítico, argumentação e letramento informacional — competências essenciais para a formação de cidadãos autônomos e conscientes no século XXI. A possibilidade de que um estudante delegue essas tarefas a uma LLM, abrindo mão de todo esse aprendizado, levanta preocupações legítimas. No entanto, o cerne dessa questão, evidenciado pelas novas tecnologias, já fazia parte dos debates educacionais: a corresponsabilidade do estudante por sua própria trajetória de aprendizagem e do valor percebido pelo estudante acerca da formação escolar. Apesar de percebidas como uma ameaça, as novas tecnologias e as transformações que elas impulsionam podem enriquecer esse debate e contribuir para a construção de uma experiência escolar mais significativa e alinhada aos propósitos dos estudantes, respeitando o espaço de atuação e a responsabilidade do professor.

Esta seção se propõe a apresentar como utilizar LLMs incentivando o aprendizado dos alunos e apoiando o trabalho do professor. Mais especificamente, auxiliaremos os professores a responder às seguintes perguntas:

1. Como estimular o pensamento crítico dos alunos através da avaliação das alucinações de IAs generativas?
2. Quais exercícios os alunos podem fazer com o auxílio de IA generativa?
3. Em quais tarefas a IA generativa pode ajudar o professor?
4. Como escrever *prompts* (comandos para a IA generativa) que forneçam melhores resultados?

2.3.1. Conceitos fundamentais de LLMs

Esta seção apresenta a fundamentação teórica do funcionamento das LLMs, dando uma visão geral e definindo conceitos como *redes neurais*, *embedding* e *tokens* de modo que o leitor entenda a natureza probabilísticas das respostas fornecidas por um modelo de IA generativa.

Redes neurais são modelos computacionais inspirados no funcionamento do cérebro humano, compostos por unidades chamadas neurônios artificiais ou perceptrons. O perceptron é a unidade fundamental dessas redes: ele recebe um conjunto de sinais de entrada, cada um multiplicado por um peso, soma esses valores e processa o resultado por meio de uma função de ativação, geralmente não linear [Vasconcelos and Clua 2017]. As redes neurais são formadas por vários perceptrons organizados em camadas: uma camada de entrada, uma ou mais camadas ocultas e uma camada de saída. A arquitetura da rede refere-se à forma como essas camadas e seus neurônios estão conectados, podendo variar conforme a aplicação e o tipo de problema a ser resolvido.

Antes de serem utilizados no treinamento do modelo, os dados passam por várias etapas de preparação, como limpeza, normalização e, principalmente, a *tokenização*. Nesse processo, a entrada é dividida em pequenas partes chamadas *tokens*. No caso de textos, esses *tokens* geralmente correspondem a palavras, subpalavras ou combinações frequentes de caracteres. Em outros tipos de dados, como imagens ou áudios, os *tokens* podem ser pequenos trechos da imagem ou fragmentos do som.

Cada um desses *tokens* é então associado a um vetor — uma lista de números — que busca capturar o significado daquele fragmento. Esses vetores, conhecidos como *embeddings*, posicionam cada *token* em um espaço de alta dimensionalidade, de forma que *tokens* com significados semelhantes fiquem próximos uns dos outros nesse espaço. Assim, o modelo pode representar o texto (ou outros dados) como uma sequência de vetores, facilitando a identificação de relações complexas entre palavras, frases ou até mesmo elementos visuais e sonoros. O Exemplo 1 ilustra como uma frase pode ser segmentada em *tokens*, e como cada *token* recebe um identificador numérico correspondente ao seu *embedding*. Os IDs são fictícios, apenas para ilustrar como cada *token* é associado a um índice numérico no vocabulário do modelo.

Exemplo 1, adaptado de [de Sousa 2025]

Texto: “Estou com dificuldades para entender estruturas de repetição”

Tokens: “Est”, “ou”, “com”, “di”, “fic”, “uldades”, “para”, “en”, “tender”, “es”, “tru”, “turas”, “de”, “re”, “pe”, “ti”, “ção”]

IDs: [501, 208, 305, 612, 713, 499, 402, 311, 718, 256, 337, 805, 109, 410, 527, 634, 999]

Após o pré-processamento, os dados — agora representados como uma sequência de vetores — são processados por arquiteturas baseadas em *transformers*. Esses modelos combinam camadas que realizam diferentes operações para analisar os *tokens* e aprender as relações entre eles. Entre as camadas mais utilizadas estão os mecanismos de atenção, redes neurais *feed forward* e *encoders* e *decoders*.

As camadas de atenção revolucionaram a IA Generativa, pois ela é que permite que os vetores “conversem” entre si, trocando informações para atualizar seus valores de acordo com o contexto. Por exemplo, a palavra “modelo” em “modelo de aprendizado de máquina” tem um significado diferente de “modelo de moda”. As camadas de atenção são responsáveis por identificar quais palavras, dentro do contexto, são relevantes para atualizar o significado das demais, ajustando as representações de acordo com a relação entre os *tokens*. Assim, mesmo que dois *tokens* estejam distantes na sequência, o mecanismo de atenção pode estabelecer conexões importantes entre eles, refinando a compreensão contextual do texto [de Sousa 2025].

A geração do texto final ocorre por meio de um processo sequencial, no qual o modelo prevê os *tokens* um a um. A cada etapa, a escolha do próximo *token* é baseada no histórico dos *tokens* já gerados e nas representações contextuais (vetores ou *embeddings*) construídas. Esse processo envolve repetidas passagens pelos blocos de atenção e camadas de rede *feed forward*, permitindo que o modelo refine continuamente a compreensão do contexto [Barros et al. 2025].

Ao final dessas iterações, espera-se que todo o significado essencial do trecho esteja incorporado no último vetor da sequência. Sobre esse vetor final, o modelo realiza uma operação que gera uma distribuição de probabilidade sobre todos os possíveis *tokens* seguintes — ou seja, sobre todas as pequenas unidades de texto que podem ser geradas na sequência. Dessa forma, o modelo seleciona o próximo *token* mais provável e repete o processo até concluir a geração do texto.

2.3.2. Técnicas de criação de *prompts*

Os *prompts* são instruções fornecidas aos LLMs com o objetivo de orientar a geração de texto e obter respostas específicas, sem a necessidade de alterar os parâmetros do modelo. Eles constituem o principal meio de interação com o sistema, estabelecendo o contexto, a tarefa a ser realizada e até mesmo o tom desejado para a resposta. Por isso, a elaboração cuidadosa do *prompt* é fundamental para garantir que o modelo forneça respostas adequadas à tarefa proposta [Sahoo et al. 2024]. Quando os modelos recebem instruções vagas ou pouco detalhadas, acabam considerando uma ampla variedade de possibilidades. Isso geralmente resulta em respostas genéricas e superficiais, semelhantes ao que se encontraria em uma simples pesquisa na internet. Embora essas respostas não sejam erradas ou inúteis, muitas vezes não atendem às necessidades específicas de determinadas tarefas [Chen et al. 2024]. No Exemplo 2, observa-se um caso em que um *prompt* genérico levou o modelo a gerar uma resposta ampla. Como o tema não foi claramente especificado, o modelo apresentou uma lista que pode não contemplar exatamente o interesse do usuário.

Uma das estratégias mais eficazes ao utilizar LLMs é elaborar instruções claras e específicas, evitando qualquer tipo de ambiguidade. Esses modelos, treinados com grandes volumes de dados textuais provenientes de diversas fontes, conseguem sintetizar informações variadas. No entanto, quando recebem *prompts* pouco detalhados, tendem a gerar respostas genéricas, o que pode comprometer a relevância em situações que exigem precisão. Em contrapartida, *prompts* bem estruturados e detalhados ajudam a reduzir a incerteza do modelo, direcionando suas respostas para atender exatamente às necessidades da tarefa. Isso resulta em conteúdos mais precisos e relevantes, aprimorando o desempenho do modelo [Sahoo et al. 2024].

Exemplo 2, retirado de [de Sousa 2025]

Entrada: Eu gostaria de saber sobre o assunto do momento no campo da tecnologia.

Saída:

As principais tendências tecnológicas de 2025 incluem:

- **Computação Quântica:** Oferece capacidades de cálculo revolucionárias para aplicações em ciência dos materiais e descoberta de drogas
- **IA Generativa:** Transforma setores através da criação automática de conteúdo complexo, melhorando a produtividade e o engajamento do cliente
- **Geminação Digital:** Utiliza IoT para replicar e analisar sistemas e processos do mundo real em ambientes virtuais, beneficiando saúde e manufatura
- **Bioprinting:** Avança na medicina com a impressão de tecidos e órgãos para transplantes, promovendo a medicina personalizada

Outra prática recomendada é atribuir ao modelo um papel específico, como o de professor ou especialista em determinado tema. Essa estratégia orienta a forma como o modelo responde, garantindo maior alinhamento com o contexto desejado. Ao definir

um papel claro, não só se direciona a produção do conteúdo, mas também se fortalece a coerência entre o *prompt* e a resposta gerada. No Exemplo 3, por exemplo, o modelo foi instruído a agir como professor de um grupo de alunos de determinada faixa etária, o que resultou em uma resposta mais adequada ao interesse do usuário.

Essas estratégias apresentam como benefícios a diminuição da necessidade de reunir grandes quantidades de dados específicos para cada tarefa, além de reduzirem o risco de o modelo limitar suas respostas apenas a perguntas previamente conhecidas. Dessa forma, o modelo se torna mais flexível e capaz de lidar com uma variedade maior de solicitações.

Exemplo 3, retirado de [de Sousa 2025]

Entrada: Você é um professor de introdução a programação para alunos do ensino médio com idades entre 15 e 18 anos. Quais projetos você poderia criar com os seus alunos usando Python?

Saída: Como professor de programação para alunos do ensino médio, você pode desenvolver projetos interessantes usando Python para ajudar a ensinar conceitos básicos e estimular o interesse dos alunos. Algumas ideias incluem:

- Calculadora Simples: Para praticar operações matemáticas básicas.
- Jogos Simples: Como “Adivinhe o Número” ou “Tic-Tac-Toe”.
- Gerador de Histórias Aleatórias: Combinando partes de frases para criar histórias.

2.3.2.1. Prompts básicos: *zero-shot*, *few-shot* e *chain-of-thought*

Entre as principais estratégias para a criação de *prompts* estão o *zero-shot*, *one-shot* e *few-shot prompting*. Essas abordagens se diferenciam pela quantidade de exemplos incluídos no *prompt*, o que impacta diretamente o desempenho do modelo em diferentes tipos de tarefas. No *zero-shot prompting*, por exemplo, o modelo é solicitado a executar uma tarefa sem receber nenhum exemplo concreto. Nesse caso, o *prompt* apenas descreve a tarefa e o contexto, sem fornecer modelos de resposta. Embora seja uma estratégia rápida e simples, ela pode não garantir alta precisão em situações em que o contexto ou o formato esperado da resposta não estejam claramente definidos [Brown et al. 2020]. No Exemplo 4 o usuário pediu que o modelo criasse um código para dividir dois números de forma genérica. Contudo, se fosse necessário um formato específico, como a inclusão de uma descrição detalhada (*docstring*), essa informação não seria contemplada automaticamente pelo modelo em sua resposta.

No *one-shot prompting*, é fornecido ao modelo um exemplo de resposta para a tarefa, o que facilita a compreensão do formato e do tipo de resposta esperado. Normalmente, o *prompt* apresenta uma instrução seguida de um exemplo, e então pede-se ao modelo que produza uma resposta semelhante. No Exemplo 5, por exemplo, o usuário

Exemplo 4, retirado de [de Sousa 2025]

Entrada: Faça um código python que calcule a divisão de dois números.

Saída:

```
def divisao( Numerador, divisor):  
    try:  
        resultado = Numerador / divisor  
        return resultado  
    except ZeroDivisionError:  
        return "Não é possível dividir por zero"
```

incluiu um código de referência que continha tanto uma *docstring* quanto um comando de impressão (*print*). O modelo, por sua vez, utilizou esse exemplo como guia para gerar o novo código solicitado, seguindo o formato especificado pelo usuário.

Visando quebrar o problema proposto em problemas menores e diminuir a chance de alucinações e respostas incorretas, as técnicas *Chain-of-Thought* [Wei et al. 2022] e *Plan-and-Solve* [Wang et al. 2023] buscam inserir etapas intermediárias de processamento nos modelos. No entanto, não há garantia de que as etapas intermediárias de processamento assegurem a correção das respostas, podendo resultar tanto em respostas incorretas quanto em respostas certas baseadas em processos falhos [Peres 2023]. A maior parte da LLMs disponíveis em formato de *chatbot* atualmente utiliza uma mistura dessas técnicas, já incorporando as etapas intermediárias em seu raciocínio. O Exemplo 6 apresenta a resposta fornecida pelo ChatGPT-4o a uma pergunta de raciocínio matemático, indicando o processamento por etapas.

Por fim, existem os *prompts* iterativos (ou de múltiplos estágios), nos quais o processamento é dividido em várias entradas enviadas ao LLM, de modo que cada nova entrada depende da resposta obtida à etapa anterior [Qiao et al. 2023]. Combinando essa abordagem com a especialização do modelo para contextos ou bases de dados específicas, essas técnicas têm demonstrado resultados superiores [Sun et al. 2024, Gao et al. 2024]. Isso ocorre porque o usuário ganha o poder de avaliar e direcionar as respostas do modelo a cada etapa, promovendo maior precisão e adequação ao objetivo desejado.

2.3.3. Exemplos práticos no contexto educacional

Esta seção traz exemplos de utilização de IA generativa no contexto educacional. São apresentados casos de uso para alunos e professores, além de fomentada a discussão sobre a dependência excessiva, as alucinações e o pensamento crítico na utilização desta tecnologia.

Para os professores da educação básica, compreender tanto as potencialidades quanto os riscos associados às tecnologias baseadas em Inteligência Artificial é essencial para integrá-las de forma crítica, ética e pedagógica ao cotidiano escolar. A construção desse entendimento requer processos formativos que desenvolvam o senso crítico e favoreçam a apropriação consciente dos mecanismos, benefícios e limitações dessas

Exemplo 5, retirado de [de Sousa 2025]

Entrada: Com base no código de exemplo abaixo, faça um código python que calcule a divisão de dois números.

```
def somar(num_a, num_b):  
    """Soma dois números e retorna o resultado.  
    Entradas:  
        num_a -> int, float  
        num_b -> int, float  
    Retorna:  
        int, float: A soma de num_a e num_b."""  
    resultado = num_a + num_b  
    print("O resultado da soma dos números é",  
          resultado)  
    return resultado
```

Saída:

```
def dividir(num_a, num_b):  
    """Divide o primeiro número pelo segundo e retorna  
    o resultado.  
    Entradas:  
        num_a -> int, float  
        num_b -> int, float  
    Retorna:  
        float: O resultado da divisão de num_a por  
        num_b."""  
    if num_b == 0:  
        print("Não é possível dividir por zero.")  
        return None  
    resultado = num_a / num_b  
    print("O resultado da divisão dos números é",  
          resultado)  
    return resultado
```

Exemplo 6, adaptado de [Kojima et al. 2022]

Entrada: Um malabarista consegue fazer malabarismo com 16 bolas. Metade das bolas são bolas de golfe e metade das bolas de golfe são azuis. Quantas bolas de golfe azuis há?

Saída:

Vamos analisar passo a passo:

1. O malabarista faz malabarismo com 16 bolas.
2. Metade das bolas são bolas de golfe $\rightarrow \frac{16}{2} = 8$ bolas de golfe.
3. Metade das bolas de golfe são azuis $\rightarrow \frac{8}{2} = 4$ bolas de golfe azuis.

Resposta: há 4 bolas de golfe azuis.

ferramentas — com destaque para sistemas generativos como o ChatGPT.

Uma introdução reflexiva e prática para docentes pode ser feita em oficinas de curta duração, abordando inicialmente os principais riscos associados ao uso da IA, como vieses algorítmicos, desinformação e questões éticas. Após essas primeiras considerações, é indicada a organização de discussões sobre os riscos e ameaças percebidos e a contraposição destes com outros riscos e ameaças já presentes no cotidiano escolar. É importante lembrar que o contexto escolar é parte da vida social e que toda implicação à vida social é uma implicação também para o contexto escolar. A IA chega na escola e cabe ao professor analisar criticamente a bagagem que ela traz.

2.3.4. Identificação e reflexão sobre os riscos da IA

A partir de uma pesquisa simples na internet sobre os riscos do uso do ChatGPT, os professores são convidados a identificar e analisar criticamente informações provenientes de diferentes fontes. Essa etapa busca revelar questões pouco percebidas previamente, promovendo o desenvolvimento do pensamento crítico e a sensibilização para a necessidade da mitigação responsável desses riscos em contextos educacionais.

Também é interessante que os professores utilizem experimentalmente o ChatGPT para gerar descrições visuais de diferentes perfis humanos (pessoas bem-sucedidas, boas, simpáticas) e avaliem se existem indícios de vieses culturais nas respostas. Tal exercício favorece a compreensão de que sistemas de IA refletem valores e limitações de seus dados de treinamento, estimulando a reflexão sobre os desafios éticos inerentes à adoção dessas tecnologias.

2.3.5. Utilizando IA para desenvolver atividades didáticas e avaliações

A proposta prática de elaboração de atividades didáticas mediadas por IA oferece aos professores a oportunidade de experimentar o potencial das ferramentas na construção de planos de aula contextualizados, alinhados a demandas curriculares (como a BNCC) e metodologias ativas. A avaliação crítica da satisfação com os resultados gerados ajuda a balizar a efetividade e os limites da tecnologia no cotidiano escolar.

É importante ressaltar a importância de fornecer detalhes como segmento/ano, matéria, e qualquer outra informação que achar pertinente. Por exemplo: Sou professora do quinto ano do Ensino Fundamental em uma escola pública brasileira e preciso planejar uma aula de 50 minutos sobre o sistema solar. Utilizando metodologias ativas e se baseando nas habilidades da Base Nacional Comum Curricular, elabore três propostas de planos de aula detalhando cada etapa. Inclua informações sobre evidências de aprendizagem e uma proposta de avaliação.

2.3.5.1. Propondo atividades didáticas que envolvam o uso de LLMs pelos alunos

Para desenhar atividades que tragam o uso de LLMs para os estudantes, é importante pensar nas vantagens do uso dessa tecnologia. A maior delas é a possibilidade de interagir, criando seus próprios prompts. Como em qualquer atividade didática, tudo começa com a definição do objetivo de aprendizagem. Esse objetivo pode estar relacionado a habilidades como a produção escrita, a análise crítica, ou a resolução de problemas, entre outros.

Por exemplo, ao pedir que os alunos dialoguem com o ChatGPT para explorar ideias, revisar textos ou simular situações, o professor estimula a autonomia e o pensamento crítico. Essas interações devem ser acompanhadas de momentos de reflexão: “O que você aprendeu?”, “Você concorda com a resposta da IA?”, “O que mudaria?”.

Exemplo: Língua Portuguesa. Competências da BNCC: Desenvolver a oralidade e a escuta ativa; Identificar elementos estruturais da história (personagem, espaço, tempo e enredo): Recontar histórias com criatividade e autonomia. Relacionar textos ouvidos com experiências pessoais.

Com a ajuda da IA, criar uma versão de contos de fadas clássicos. A IA pode ser usada para fazer imagens ou sugerir textos, porém os alunos têm que partir de ideias próprias para criar o prompt. Há um amplo universo de questões que podem ser trabalhadas, além da construção de narrativas. Alguns exemplos mais diretos são questões de gênero, representatividade e regionalidades.

Outro tipo de atividade que atende a qualquer área do conhecimento é o uso da LLM para questionamento Socrático. A técnica do questionamento socrático, originada na prática filosófica de Sócrates, baseia-se em provocar o pensamento crítico por meio de perguntas que estimulam a reflexão, o raciocínio lógico e a construção do conhecimento pelo próprio sujeito. Em vez de fornecer respostas prontas, o método busca orientar o estudante a descobrir por si mesmo os fundamentos, contradições e implicações de suas crenças, fornecendo, ao invés de respostas, outras perguntas.

Como a IA é capaz de sustentar diálogos e responder a perguntas em linguagem natural, o professor ou o próprio aluno pode utilizar o ChatGPT para simular uma conversa socrática, formulando questões abertas, exploratórias e reflexivas.

Exemplos de comando:

- “Acredito que as redes sociais fazem mais mal do que bem. Faça 5 perguntas que desafiem meu ponto de vista.”

- “Estou com dificuldades para entender o Teorema de Pitágoras. Faça perguntas que me ajudem a descobrir sozinho como ele funciona.”
- “Aqui está um problema: ‘Um trem sai de uma cidade a 60 km/h...’ — Me ajude a resolver esse problema fazendo apenas perguntas.”
- “Estou tentando entender por que um objeto em queda livre acelera. Faça perguntas que me ajudem a entender isso sozinho.”
- “Aqui está um problema: Um carro de 1000 kg é puxado com uma força de 2000 N. Me ajude a entender o que está acontecendo fazendo apenas perguntas.”
- “Por que vemos um objeto de uma certa cor? Me guie por meio de perguntas.”

2.4. Questões éticas relacionadas ao uso de IA na educação

Os avanços tecnológicos, a Inteligência Artificial (IA) e o uso crescente de dados em ambientes educacionais ampliaram as possibilidades de personalização da aprendizagem e de aprimoramento dos processos pedagógicos. No entanto, essa transformação também traz desafios significativos relacionados ao uso ético, seguro e transparente dessas informações, bem como os impactos que essas novas no ambiente educacional.

É bastante perceptível a aceleração do uso de IA no setor educacional ocorrida nos últimos anos, especialmente após a pandemia de COVID-19, que impulsionou a digitalização do ensino. Como essas tecnologias têm alto potencial para interferir e até mesmo transformar o ambiente educacional, organizações internacionais que se ocupam de questões de impacto global produziram documentos que discutem a IA e até especificamente o uso de IA na educação.

A consideração dos riscos associados à aplicação da Inteligência Artificial (IA) na educação é fundamental para garantir que essas tecnologias sejam utilizadas de forma alinhada aos princípios educacionais democráticos, contribuindo para o objetivo primordial da educação – a formação do indivíduo livre, crítico, autônomo e apto a promover o bem-estar próprio e do seu entorno.

Sistemas de IA aplicados à educação processam grandes volumes de dados pessoais e sensíveis de estudantes, como desempenho acadêmico, histórico de comportamento, de condições de saúde física, emocional e psicológica, e algumas vezes até mesmo dados biométricos. O uso indevido de informações dessa natureza tão pessoal pode violar legislações como a Lei Geral de Proteção de Dados (LGPD) no Brasil e o Regulamento Geral sobre a Proteção de Dados (GDPR) na União Europeia, comprometendo a privacidade e a segurança dos alunos [UNESCO 2021a].

Algoritmos treinados com dados enviesados, muitas vezes por questões históricas e/ou estruturais, podem reproduzir e até intensificar desigualdades já existentes no sistema educacional, impactando estudantes de grupos vulneráveis em processos de recomendação de conteúdos e ações pedagógicas, e reforçando as situações estruturais de fracasso escolar.

Ferramentas de IA frequentemente operam como “caixas-pretas” para os usuários finais, dificultando a compreensão sobre o resultado produzido. A falta de transparência

compromete a possibilidade de responsabilização dos tomadores de decisão e impede que professores, estudantes e famílias entendam, ponderem ou contestem decisões automatizadas [UNESCO 2023].

A delegação de tarefas pedagógicas complexas à IA pode reduzir a autonomia dos educadores, ocultar nuances dos processos pessoais de aprendizagem dos alunos, coibir o diálogo e as oportunidades de acolhimento e suporte, comprometendo o papel da mediação humana no ensino-aprendizagem. A utilização de sistemas de monitoramento e personalização baseados em IA pode gerar estresse e sensação de vigilância constante, afetando o ambiente escolar e a saúde mental dos estudantes.

Há ainda a dependência de soluções tecnológicas desenvolvidas por grandes corporações estrangeiras, que pode comprometer a soberania educacional e tecnológica dos países. Isso levanta preocupações quanto à autonomia dos sistemas educacionais e à governança dos dados coletados.

2.4.1. Mapeamento dos riscos do uso de IA na educação

O relatório *Governing AI for Humanity* [High-Level Advisory Body on AI 2024], elaborado pelo Alto Comitê Consultivo das Nações Unidas (ONU) e publicado em 2024, propõe princípios e recomendações globais para a governança ética e segura da IA, com foco na promoção dos direitos humanos, na equidade e na sustentabilidade. Neste relatório, é apresentada uma lista de riscos relacionados à IA, categorizados com base na vulnerabilidade existente ou potencial. Esses riscos abrangem aspectos éticos, sociais, econômicos e ambientais, incluindo:

1. **Viés algorítmico e discriminação:** Sistemas de IA podem perpetuar ou amplificar preconceitos existentes, resultando em decisões injustas, especialmente em áreas como saúde, justiça e educação.
2. **Privacidade e segurança de dados:** O uso extensivo de dados pessoais por sistemas de IA levanta preocupações sobre a proteção da privacidade e a segurança das informações sensíveis.
3. **Desinformação e manipulação:** A IA pode ser utilizada para criar e disseminar informações falsas ou enganosas, impactando negativamente a opinião pública e os processos democráticos.
4. **Armas autônomas e uso malicioso:** O desenvolvimento de sistemas de IA para fins militares ou por atores mal-intencionados representa uma ameaça significativa à segurança global.
5. **Impactos socioeconômicos:** A automação impulsionada pela IA pode levar à perda de empregos e ao aumento das desigualdades econômicas, afetando especialmente comunidades vulneráveis.
6. **Consumo energético e sustentabilidade:** O treinamento e operação de modelos de IA de grande escala demandam altos níveis de energia, contribuindo para impactos ambientais adversos.

7. **Concentração de poder:** O domínio do desenvolvimento de IA por um número limitado de empresas e países pode resultar em desequilíbrios de poder e acesso desigual aos benefícios da tecnologia.

O relatório enfatiza a necessidade de uma governança global inclusiva e colaborativa para mitigar esses riscos e garantir que a IA seja desenvolvida e utilizada de maneira ética, segura e benéfica para toda a humanidade.

A Organização das Nações Unidas para a Educação, a Ciência e a Cultura – UNESCO publicou diversos relatórios que abordam os riscos associados ao uso da Inteligência Artificial (IA). Em 2021, a UNESCO publicou a Recomendação sobre a Ética da Inteligência Artificial [UNESCO 2021b], na qual reconhece que a IA pode trazer benefícios significativos para a sociedade; contudo, alcançar esses benefícios pode intensificar tensões já existentes, relacionadas à transformação das relações sociais e ao acesso desigual ao conhecimento e às tecnologias. Entre os desafios estão: Déficit de **alfabetização digital e cívica**, que limita a capacidade da população de se envolver criticamente com os temas ligados à IA; Lacunas em capacidades institucionais para mitigar questões como **transferência de responsabilidades**; Deficiências em **infraestrutura física e digital**; Ausência de **marcos regulatórios** adequados no que se refere à governança de dados, podendo ameaçar o respeito e a proteção à privacidade e à diversidade cultural e agravar situações de vulnerabilidade.

A publicação *Artificial Intelligence and Education: Guidance for Policy-makers* [UNESCO 2021a] oferece orientações para formuladores de políticas sobre como melhor aproveitar as oportunidades e enfrentar os riscos apresentados pela crescente conexão entre IA e educação. Os riscos já apontados nas referências anteriores foram analisados à luz das especificidades do contexto educacional. De forma resumida: **Ampliação das desigualdades educacionais:** a rápida adoção de tecnologias de IA pode acentuar disparidades existentes, especialmente em contextos com acesso limitado a recursos tecnológicos, infraestrutura e conectividade. **Viés algorítmico e discriminação:** sistemas de IA podem perpetuar ou amplificar preconceitos existentes, resultando em decisões injustas, especialmente em áreas como saúde, justiça e educação. **Privacidade e segurança de dados:** o uso extensivo de dados pessoais por sistemas de IA levanta preocupações sobre a proteção da privacidade e a segurança das informações sensíveis. **Dependência excessiva da IA:** a confiança excessiva em sistemas de IA pode levar à redução do papel dos educadores e ao enfraquecimento das interações humanas essenciais para o processo de ensino-aprendizagem. **Desinformação e manipulação:** a IA pode ser utilizada para criar e disseminar informações falsas ou enganosas, impactando negativamente a opinião pública e os processos democráticos.

Especificamente sobre o uso de IA generativa na educação, a UNESCO alerta para riscos como a distorção de fatos históricos e a propagação de discursos de ódio, conforme destacado no relatório *Generation AI: Navigating the opportunities and risks of artificial intelligence in education* [UNESCO 2024]. Para mitigar esses riscos, a UNESCO recomenda o desenvolvimento de estruturas normativas robustas, políticas educacionais inclusivas e estratégias de capacitação que promovam o uso ético e equitativo da IA na educação. Profissionais da computação e especialistas em informática na educação tem um lugar de fala importante para essa discussão.

Além dos documentos produzidos por estas organizações mundiais, voltadas primordialmente ao desenvolvimento sustentável e a propósitos humanitários e socioambientais, vale mencionar também o *AI Risk Repository*² do Instituto de Tecnologia de Massachusetts (MIT), lançado em 2024 e atualizado em abril de 2025. O *AI Risk Repository* é uma base de dados pública e dinâmica que reúne mais de 1.600 riscos associados à IA, extraídos de 65 estruturas e classificações existentes. Desenvolvido pelo *MIT FutureTech*, o repositório visa consolidar e esclarecer como os riscos da IA são categorizados nas literaturas acadêmica, industrial e política, oferecendo uma estrutura comum para monitoramento e supervisão desses riscos [Slattery et al. 2025].

2.4.2. Gestão de Dados Educacionais

O contexto educacional gera uma ampla gama de informações que se qualificam como dados educacionais. Isso inclui avaliações, informações pessoais dos estudantes, como endereço, idade, histórico médico e dados de contato, bem como registros administrativos, como horários de entrada e saída, frequência, relatórios descritivos sobre comportamento ou desempenho em atividades não avaliativas, e interações gerais no âmbito da comunidade escolar. Esses dados desempenham um papel crucial na gestão educacional, contribuindo para o monitoramento do bem-estar dos estudantes, a organização administrativa e o planejamento pedagógico.

Informações detalhadas sobre o desempenho acadêmico dos estudantes permitem que professores e gestores desenvolvam estratégias pedagógicas individualizadas, atendendo às necessidades específicas de cada aluno e promovendo um progresso educacional mais eficaz [Hernández-de Menéndez et al. 2022]. O uso desses dados é comum em aplicações de IA [Ahmad et al. 2024]. Eles podem apoiar a identificação precoce de dificuldades de aprendizagem, permitindo intervenções direcionadas que ajudam a reduzir as taxas de evasão escolar e a melhorar os indicadores de permanência dos alunos. Dados consolidados podem orientar políticas educacionais e permitir uma alocação mais eficiente de recursos, contribuindo para a redução das desigualdades no acesso à educação. Porém, o uso indevido de dados educacionais pode gerar consequências negativas. Um dos riscos mais significativos é a violação da privacidade dos estudantes, especialmente em contextos onde medidas de proteção ou anonimização são insuficientes. Tal exposição pode comprometer a segurança dos alunos e aumentar sua vulnerabilidade a abusos, discriminação ou práticas de vigilância excessiva [Gašević et al. 2015]. Outro impacto negativo potencial é o reforço das desigualdades educacionais preexistentes. Quando o uso de algoritmos ou ferramentas de análise de dados não é acompanhado de supervisão ética e técnica, pode-se perpetuar vieses sistêmicos, categorizando ou limitando oportunidades com base em padrões históricos ou distorcidos, e desprezando o poder que ações educativas têm de transformar indivíduos.

A gestão de dados educacionais exige uma análise cuidadosa dos parâmetros legais e regulatórios que regem a geração, o armazenamento e a utilização dessas informações por instituições educacionais no Brasil. No entanto, os marcos legais frequentemente não acompanham a velocidade dos avanços tecnológicos, o que gera lacunas que dificultam sua aplicação prática [Almeida and Soares 2022]. A proteção de dados

²<https://airisk.mit.edu>, acessado em maio de 2025

sensíveis representa um dos principais desafios da LGPD, especialmente devido ao seu potencial discriminatório e aos riscos associados ao manuseio inadequado dessas informações [Sarlet and Ruaro 2021]. No contexto educacional, essa preocupação se torna ainda mais premente, dada a vulnerabilidade dos estudantes e a necessidade de resguardar sua privacidade e dignidade.

Embora a LGPD forneça uma estrutura inicial para a proteção de dados pessoais, ainda há ausência de regulamentações específicas voltadas para o setor educacional. Além disso, a Autoridade Nacional de Proteção de Dados (ANPD) ainda não emitiu diretrizes específicas para instituições educacionais, contribuindo para um cenário de incerteza quanto ao uso adequado dos dados educacionais. Essa lacuna regulatória cria um ambiente de insegurança jurídica, no qual as instituições educacionais enfrentam dificuldades para alinhar suas práticas às disposições legais existentes.

A implementação da LGPD nas instituições de ensino exige a adoção de medidas de governança, segurança e transparência de dados, em conformidade com recomendações de adaptação já identificadas em outros contextos institucionais. Estudos sobre a aplicação dessa legislação em diferentes tipos de instituições, como repositórios institucionais [Nascimento and Silva 2023] e escolas de educação infantil [Wendling et al. 2023], demonstram que o cumprimento da LGPD demanda não apenas ajustes técnicos, mas também transformações organizacionais. Esse processo envolve a sensibilização dos atores institucionais, o estabelecimento de protocolos claros de gestão de dados e a oferta de formação contínua aos responsáveis pelo tratamento das informações.

Entre os princípios gerais da LGPD aplicáveis ao contexto educacional, destaca-se o princípio da *transparência*, que exige das instituições a comunicação clara sobre como os dados são coletados, processados e armazenados. Além disso, é exigida a publicação de políticas de privacidade que detalhem os procedimentos e finalidades relacionados ao uso dos dados. A legislação também estabelece a obrigatoriedade da indicação de um Encarregado pelo Tratamento de Dados Pessoais (DPO), responsável por assegurar o cumprimento legal, bem como pela formação contínua dos envolvidos na aplicação das normas de proteção de dados. A constituição de equipes de conformidade também é essencial para a identificação de riscos e o cumprimento dos requisitos legais. Por fim, a LGPD enfatiza a importância da *anonimização* dos dados sempre que possível, como medida de mitigação de riscos e prevenção à exposição de informações pessoais. No entanto, permanecem indefinições, como a classificação de dados sensíveis no contexto educacional e os métodos eficazes de anonimização, especialmente em avaliações de estudantes.

Em contrapartida, países como o Reino Unido e os Estados Unidos possuem estruturas regulatórias mais detalhadas e específicas, que podem servir de referência para o aprimoramento da legislação brasileira. No Reino Unido, o tratamento de dados educacionais é regido pelo UK GDPR e pela *Data Protection Act 2018* [Kingdom 2018], que adaptam o Regulamento Geral de Proteção de Dados da União Europeia (GDPR) [Parliament and Council 2016] ao contexto britânico. Nesse modelo, as escolas devem coletar dados exclusivamente para finalidades educacionais claras e justificáveis, minimizar a coleta de dados e garantir a transparência no uso dessas informações. Além disso, o consentimento é requisito obrigatório para o compartilhamento de dados com terceiros, exceto em casos com base legal adequada.

Nos Estados Unidos, o arcabouço regulatório é definido pelo *Family Educational Rights and Privacy Act* (FERPA) [of Education 1974], que protege a privacidade dos dados educacionais e garante aos pais o direito de acesso e correção dos registros escolares de seus filhos. A FERPA também estabelece limitações rigorosas quanto ao compartilhamento de informações, permitindo acesso apenas a indivíduos autorizados e para finalidades legítimas, como auditorias ou análises pedagógicas. Em casos de violação, a legislação prevê sanções federais às instituições educacionais, reforçando a importância da conformidade e da proteção da privacidade dos dados.

No entanto, mesmo em países com legislações robustas de proteção de dados, como os Estados Unidos e nações europeias, persistem desafios relacionados ao uso indevido dessas informações. Estudos apontam que, nos Estados Unidos, a implementação de políticas de responsabilização escolar baseadas em dados nem sempre resultou em melhoria do desempenho estudantil. Em alguns casos, essas políticas levaram a práticas indesejáveis, como a manipulação de resultados para alcançar metas predefinidas, sem, de fato, melhorar a qualidade da educação [Núñez et al. 2019].

É importante ressaltar que dados produzidos por ferramentas de IA a partir de dados educacionais — como em previsões de desempenho ou recomendações — também devem ser tratados com o mesmo rigor ético e em conformidade com a legislação que se aplica a outros tipos de dados educacionais, garantindo o alinhamento com os marcos regulatórios e as boas práticas de governança de dados. A falta deste conhecimento expõe não apenas as instituições e os profissionais a riscos, mas também os próprios estudantes e suas famílias. Inseguranças desses atores quanto à privacidade e proteção de dados fazem com que instituições relutem em adotar novas tecnologias educacionais [van Haastrecht et al. 2024]. Quanto menor a confiança das instituições em sua capacidade de coletar, armazenar e usar informações pessoais de forma segura, respeitando a legislação vigente, menor será a probabilidade de investimento nas tecnologias que dependem dessas informações.

2.4.2.1. Métodos e técnicas para mitigação de vies

A discussão sobre justiça e responsabilidades no uso da inteligência artificial (IA) tem se intensificado à medida que os sistemas algorítmicos assumem papéis cada vez mais decisivos em processos sociais, educacionais e governamentais. O reconhecimento dos potenciais vieses algorítmicos e de sua reprodução de desigualdades históricas tem sido alvo de críticas e análises profundas por diversos pesquisadores.

A mitigação de injustiças e vieses no uso de IA é um dos grandes desafios contemporâneos. Algoritmos de IA são treinados com dados históricos que muitas vezes refletem desigualdades sociais, raciais e econômicas, podendo reproduzi-las ou mesmo agravá-las [Noble 2018, Eubanks 2018]. Por esse motivo, diversas abordagens técnicas, institucionais e regulatórias têm sido propostas para promover maior justiça e equidade.

Do ponto de vista técnico, estratégias podem ser adotadas para tentar garantir dados diversos e representativos para alimentar os algoritmos de Inteligência Artificial. Técnicas como *pré-processamento* (*pre-processing*), *in-processing* e *post-processing* são amplamente utilizadas para mitigar vieses algorítmicos.

O pré-processamento atua diretamente na preparação dos dados, antes do treinamento dos modelos, com o objetivo de reduzir ou eliminar padrões discriminatórios que possam levar a decisões enviesadas [Barocas et al. 2019]. Pode envolver diferentes estratégias, como a *remoção de atributos sensíveis* (por exemplo, raça, gênero ou religião), um *balanceamento de classes* para igualar distribuições de grupos sub-representados, e a *transformação de variáveis correlacionadas* a atributos sensíveis, conhecidas como *proxies discriminatórios*. A exclusão direta de atributos sensíveis nem sempre é suficiente, uma vez que outras variáveis — como código postal, histórico educacional, profissão (própria ou dos pais) — podem indiretamente refletir essas características e perpetuar preconceitos ou situações desvantajosas [Zemel et al. 2013].

A técnica *Learning Fair Representations*, proposta por Zemel et al. (2013), transforma os dados em representações que retêm a informação necessária para a predição, mas minimizam a dependência de atributos sensíveis. Outra abordagem é o *Reweighting*, que ajusta os pesos das instâncias durante o treinamento para compensar disparidades entre diferentes grupos [Kamiran and Calders 2012]. Além disso, algoritmos de pré-processamento podem utilizar métodos estatísticos para verificar a presença de viés nos dados, como *disparate impact* e *equalized odds*, permitindo a identificação de fontes de discriminação antes do uso do modelo preditivo. O uso dessas técnicas contribui para o desenvolvimento de sistemas mais éticos e inclusivos, alinhando-se a princípios como justiça, responsabilidade e respeito aos direitos humanos [Russell and et al. 2022]. Contudo, a aplicação do pré-processamento requer cautela, pois alterações excessivas podem comprometer a acurácia e a utilidade dos modelos.

O *in-processing* inclui modificações no próprio algoritmo de aprendizado de máquina [Barocas et al. 2019], no intuito de promover equidade durante o treinamento dos modelos. Diferentemente das abordagens de pré-processamento, que modificam os dados antes do treinamento, o *in-processing* altera a função de otimização ou o próprio algoritmo de aprendizado para que ele produza previsões mais justas. Isso geralmente é feito por meio da introdução de restrições ou penalidades associadas a medidas de injustiça na função de custo do modelo [Hardt et al. 2016], ou seja, ajustes que penalizam decisões injustas.

Um exemplo é o método *Adversarial Debiasing*, no qual um modelo preditivo principal é treinado em conjunto com um adversário que tenta inferir atributos sensíveis a partir das previsões do modelo [Zhang et al. 2018]. O objetivo é minimizar simultaneamente o erro de predição e a capacidade do adversário de identificar os atributos sensíveis, forçando o modelo a fazer previsões menos enviesadas.

Outra técnica é a incorporação de métricas de equidade diretamente na função de penalidade do modelo, como a *Equalized Odds*, que busca garantir que a taxa de verdadeiros positivos e a taxa de falsos positivos sejam semelhantes entre diferentes grupos [Hardt et al. 2016]. Tais abordagens são especialmente úteis quando há interesse em promover justiça de resultados, e não apenas a justiça no tratamento de cada caso.

O *in-processing* tem como vantagem não requerer alterações nos dados nem etapas de correção posterior do resultado, o que pode preservar melhor a acurácia do modelo. No entanto, pode acarretar aumento da complexidade computacional e a necessidade de algoritmos de treinamento customizados.

O *post-processing* consiste na modificação das saídas do modelo após o processo de treinamento, com o objetivo de torná-las mais justas segundo critérios pré-estabelecidos de equidade [Barocas et al. 2019]. As técnicas de *post-processing* atuam exclusivamente sobre resultados do modelo já treinado. Por isso, são particularmente atrativas em contextos nos quais o modelo original não pode ser modificado, como em sistemas de terceiros.

O método *Equalized Odds Postprocessing* [Hardt et al. 2016], de forma análoga ao *Equalized Odds* do *in-processing*, ajusta as previsões positivas e negativas de um classificador de forma a garantir que a taxa de verdadeiros positivos e a taxa de falsos positivos sejam semelhantes entre diferentes grupos (por exemplo, raça ou gênero). O método utiliza programação linear para redistribuir as saídas sem comprometer significativamente a acurácia. Na técnica *Reject Option Classification* as decisões mais próximas ao limiar de decisão são reavaliadas com base em critérios de equidade. Assim, em casos onde a decisão já teria um alto grau de incerteza, o sistema pode favorecer o grupo historicamente desfavorecido, contribuindo para a redução de discriminação indireta [Kamiran and Calders 2012].

Embora sejam menos invasivas e mais fáceis de implementar, as técnicas de *post-processing* podem gerar inconsistências entre a previsão e a pontuação original do modelo, o que pode ser problemático em contextos sensíveis à rastreabilidade das decisões. Essa abordagem é uma opção valiosa quando não é possível reestruturar os dados ou o algoritmo.

2.4.2.2. Ferramentas para Avaliação e Mitigação de Viés

Ferramentas como o *AI Fairness 360* (AIF360), da IBM, e os *Fairness Indicators*, do Google, têm sido reconhecidas como recursos interessantes para identificar e mitigar vieses em sistemas de IA.

O *AI Fairness 360*³ é uma biblioteca de código aberto desenvolvida pela IBM, projetada para detectar, entender e mitigar vieses em modelos de aprendizado de máquina. Disponível em Python e R, esse *toolkit* oferece: mais de 70 métricas de equidade, como *Statistical Parity Difference*, *Equal Opportunity Difference* e *Disparate Impact*; dez algoritmos de mitigação de viés, abrangendo técnicas de pré-processamento; tutoriais interativos e documentação, aplicáveis a diversos domínios como saúde, educação e finanças.

Os *Fairness Indicators*⁴ desenvolvidos pelo Google, são utilizados para avaliar a equidade de modelos de aprendizado de máquina. Integrados ao TensorFlow, permitem: avaliar métricas de desempenho como *precision* e *recall* em diferentes subgrupos definidos por atributos sensíveis; visualizar disparidades entre grupos e facilitar a identificação de vieses; integração com outras ferramentas.

Embora eficazes para identificação de desigualdades, os *Fairness Indicators* não fornecem mecanismos de mitigação, sendo frequentemente usados em conjunto com ou-

³<https://ai-fairness-360.org/>, acessado em maio de 2025.

⁴<https://github.com/tensorflow/fairness-indicators>, acessado em maio de 2025.

tras bibliotecas como o AIF360.

A *What-If Tool*⁵ é uma ferramenta interativa de visualização desenvolvida pela equipe PAIR (People + AI Research) do Google [Wexler et al. 2020], projetada para facilitar a análise de modelos de aprendizado de máquina sem a necessidade de programação adicional. Integrada ao *TensorBoard* e compatível com o TensorFlow e outros frameworks via API, a ferramenta oferece suporte direto à avaliação de justiça, explicabilidade e desempenho dos modelos.

A *What-If Tool* permite aos usuários explorar o comportamento de modelos preditivos de forma interativa, destacando-se pelas seguintes funcionalidades: **Análise Contra-factual:** Permite modificar manualmente atributos de uma instância de entrada e observar como a previsão do modelo muda, ajudando a identificar variáveis sensíveis. **Comparação entre modelos:** Torna possível comparar o desempenho e as previsões de múltiplos modelos aplicados ao mesmo conjunto de dados, identificando inconsistências. **Visualização de Métricas de Justiça:** Oferece métricas como *Statistical Parity*, *Equal Opportunity* e *Disparate Impact*, facilitando a avaliação de justiça algorítmica entre subgrupos. **Agrupamento por atributos sensíveis:** O desempenho do modelo pode ser analisado por subgrupos definidos por atributos como raça, gênero ou idade, promovendo uma visão granular do comportamento preditivo.

Apesar de ser uma excelente ferramenta interativa de visualização, o que ajuda a perceber vieses e analisar suas possíveis causas, a *What-If Tool* não implementa algoritmos de mitigação. Uma vantagem dessa ferramenta é a possibilidade de uso das funcionalidades interativas e visuais por profissionais de diversas áreas, contribuindo para a democratização da análise crítica de modelos de aprendizado de máquina.

As ferramentas apresentadas, bem como outras já existentes ou ainda em construção, são importantes peças para a formação de um acervo de recursos para o desenvolvimento ético de sistemas inteligentes, possibilitando uma abordagem mais sistemática à mitigação de vieses. Contudo, é imprescindível ressaltar que ferramentas técnicas não enfrentam sozinhas as causas sociais e históricas dos vieses [Birhane 2021]. Decisões sobre o que é justo dependem de contextos socioculturais, e exigem participação ativa de grupos afetados. Uma abordagem sociotécnica é indicada, combinando análises técnicas com consultas públicas, auditorias participativas e governança inclusiva [Cath 2018].

A necessidade de atribuição de responsabilidades legais em casos de falhas ou discriminação algorítmica é um ponto central. Documentos internacionais como as diretrizes da [UNESCO 2021b] e o *White Paper* da [European Commission 2020] delineiam princípios éticos fundamentais, incluindo justiça, responsabilidade, transparência e não discriminação.

A implementação de mecanismos de responsabilização, como a nomeação de agentes de proteção de dados (DPOs), auditorias independentes e canais para denúncias de danos algorítmicos, reforça a responsabilidade das instituições pelo uso ético da IA. É importante também garantir diversidade nas equipes de desenvolvimento, incluindo diferentes origens sociais, de gênero e culturais, como forma de reduzir cegueiras epistêmicas e ampliar a compreensão dos impactos sociais dos sistemas de IA

⁵<https://pair-code.github.io/what-if-tool/>, acessado em maio de 2025.

[Buolamwini and Gebru 2018]. Portanto, a mitigação de injustiças no uso da IA não depende apenas de soluções técnicas, mas de um ecossistema que promova responsabilidade, participação democrática e justiça social.

Referências

- [Abu-Mostafa et al. 2012] Abu-Mostafa, Y. S., Magdon-Ismail, M., and Lin, H.-T. (2012). *Learning From Data: A Short Course*. AMLBook.
- [Ahmad et al. 2024] Ahmad, K., Iqbal, W., El-Hassan, A., Qadir, J., Benhaddou, D., Ayyash, M., and Al-Fuqaha, A. (2024). Data-Driven Artificial Intelligence in Education: A Comprehensive Review. *IEEE Transactions on Learning Technologies*, 17:12–31.
- [Almeida and Soares 2022] Almeida, S. D. C. D. D. and Soares, T. A. (2022). Os impactos da lei geral de proteção de dados-igpd no cenário digital. *Perspectivas em Ciência da Informação*, 27:26–45.
- [Barbetta et al. 2010] Barbetta, P. A., Reis, M. M., and Bornia, A. C. (2010). *Estatística Para Cursos de Engenharia e Informática*. Atlas, São Paulo, 3 edition.
- [Barocas et al. 2019] Barocas, S., Hardt, M., and Narayanan, A. (2019). *Fairness and Machine Learning: Limitations and Opportunities*. fairmlbook.org. Disponível em: <https://fairmlbook.org>.
- [Barros et al. 2025] Barros, J., Moraes, L. O., Oliveira, F., and Delgado, C. A. D. M. (2025). Large language models generating feedback for students of introductory programming courses. In *Proceedings of the Artificial Intelligence in Education Conference*, Palermo, Italy. Springer. A ser publicado.
- [Becker et al. 2022] Becker, B. A., Denny, P., Finnie-Ansley, J., Luxton-Reilly, A., Prather, J., and Santos, E. A. (2022). Programming is hard - or at least it used to be: Educational opportunities and challenges of ai code generation. *Arxiv*. Available at <https://arxiv.org/abs/2212.01020>.
- [Belle and Papantonis 2021] Belle, V. and Papantonis, I. (2021). Principles and practice of explainable machine learning. *Frontiers in Big Data*, 4.
- [Birhane 2021] Birhane, A. (2021). Algorithmic injustice: a relational ethics approach. *Patterns*, 2(2).
- [Brown et al. 2020] Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D. (2020). Language models are few-shot learners. *arXiv*. Available at <https://arxiv.org/abs/2005.14165v4>.

- [Buolamwini and Gebru 2018] Buolamwini, J. and Gebru, T. (2018). Gender shades: Intersectional accuracy disparities in commercial gender classification. In *Proceedings of the 1st Conference on Fairness, Accountability and Transparency (FAT* 2018)*, pages 77–91. PMLR.
- [Campbell et al. 2000] Campbell, M., Jr., A. J. H., and Hsiung Hsu, F. (2000). Deep blue. *IBM Journal of Research and Development*, 44(3):213–228.
- [Cath 2018] Cath, C. (2018). Governing artificial intelligence: Ethical, legal and technical opportunities and challenges. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 376(2133):20180080.
- [Chen et al. 2024] Chen, B., Zhang, Z., Langrené, N., and Zhu, S. (2024). Unleashing the potential of prompt engineering in large language models: A comprehensive review. *arXiv*. Available at <https://arxiv.org/abs/2310.14735v5>.
- [Cunha 1980] Cunha, L. A. (1980). *Educação e desenvolvimento social no Brasil*. Francisco Alves, Rio de Janeiro.
- [de Carvalho et al. 2024] de Carvalho, A. C. P. L. F., Menezes, A. G., and Bonidia, R. P. (2024). *Ciência de Dados - Fundamentos e Aplicações*. LTC.
- [de Oliveira F. Moraes 2016] de Oliveira F. Moraes, L. (2016). Classificação de linfomas utilizando uma abordagem baseada em Árvores de decisão. Mestrado, Universidade Federal do Rio de Janeiro (UFRJ).
- [de Sousa 2025] de Sousa, J. B. (2025). Modelos de linguagem em larga escala na geração de feedbacks para alunos de cursos introdutórios de programação da ufrj. Trabalho de conclusão de curso, Universidade Federal do Rio de Janeiro (UFRJ), Escola Politécnica, Rio de Janeiro.
- [Denny et al. 2024] Denny, P., Prather, J., Becker, B. A., Finnie-Ansley, J., Hellas, A., Leinonen, J., Luxton-Reilly, A., Reeves, B. N., Santos, E. A., and Sarsa, S. (2024). Computing education in the era of generative ai. *Communications of the ACM*, 67(2):56–67.
- [Dobson and Barnett 2008] Dobson, A. and Barnett, A. (2008). *An Introduction to Generalized Linear Models*. Chapman & Hall/CRC Texts in Statistical Science. Chapman and Hall/CRC.
- [Eubanks 2018] Eubanks, V. (2018). *Automating Inequality: How High-Tech Tools Profile, Police, and Punish the Poor*. St. Martin’s Press.
- [European Commission 2020] European Commission (2020). White paper on artificial intelligence: A european approach to excellence and trust.
- [Freire 1970] Freire, P. (1970). *Pedagogia do oprimido*. Paz e Terra, Rio de Janeiro.
- [Gao et al. 2024] Gao, Y., Xiong, Y., Gao, X., Jia, K., Pan, J., Bi, Y., Dai, Y., Sun, J., Wang, M., and Wang, H. (2024). Retrieval-augmented generation for large language models: A survey.

- [Gašević et al. 2015] Gašević, D., Dawson, S., and Siemens, G. (2015). Let’s not forget: Learning analytics are about learning. *TechTrends*, 59(1):64–71.
- [Hardt et al. 2016] Hardt, M., Price, E., and Srebro, N. (2016). Equality of opportunity in supervised learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 3315–3323.
- [Hassany et al. 2024] Hassany, M., Brusilovsky, P., Jiaze, K., Akhsueyinoglu, K., and Narayanan, A. B. L. (2024). Human-ai co-creation of worked examples for programming classes. In *Workshops at the International Conference on Intelligent User Interfaces (IUI) 2024*.
- [Hellas et al. 2018] Hellas, A., Ihantola, P., Petersen, A., Ajanovski, V. V., Gutica, M., Hynninen, T., Knutas, A., Leinonen, J., Messom, C., and Liao, S. N. (2018). Predicting academic performance: a systematic literature review. In *Proceedings Companion of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education, ITiCSE 2018 Companion*, page 175–199, New York, NY, USA. ACM.
- [Hernández-de Menéndez et al. 2022] Hernández-de Menéndez, M., Morales-Menendez, R., Escobar, C. A., and Ramírez Mendoza, R. A. (2022). Learning analytics: state of the art. *International Journal on Interactive Design and Manufacturing (IJIDeM)*, 16(3):1209–1230.
- [High-Level Advisory Body on AI 2024] High-Level Advisory Body on AI (2024). Governing ai for humanity: Report of the high-level advisory body on artificial intelligence. Acesso em: 22 maio 2025.
- [INEP 2023] INEP (2023). Sistema de avaliação da educação básica (saeb). <https://www.gov.br/inep/pt-br/areas-de-atuacao/avaliacao-e-exames-educacionais/saeb>. Acesso em: 2 jun. 2025.
- [Izbicki and dos Santos 2020] Izbicki, R. and dos Santos, T. M. (2020). *Aprendizado de máquina: uma abordagem estatística*.
- [James et al. 2023] James, G., Witten, D., Hastie, T., Tibshirani, R., and Taylor, J. (2023). *An Introduction to Statistical Learning: with Applications in Python*. Springer, 1st edition.
- [Kamiran and Calders 2012] Kamiran, F. and Calders, T. (2012). Decision theory for discrimination-aware classification. *IEEE Transactions on Knowledge and Data Engineering*, 25(5):833–846.
- [Kasneji et al. 2023] Kasneji, E., Sessler, K., Küchemann, S., Bannert, M., Dementieva, D., Fischer, F., Gasser, U., Groh, G., Günemann, S., Hüllermeier, E., Krusche, S., Kutyniok, G., Michaeli, T., Nerdel, C., Pfeffer, J., Poquet, O., Sailer, M., Schmidt, A., Seidel, T., Stadler, M., Weller, J., Kuhn, J., and Kasneji, G. (2023). Chatgpt for good? on opportunities and challenges of large language models for education. *Learning and Individual Differences*, 103:102274.

- [Kazemitabaar et al. 2024] Kazemitabaar, M., Hou, X., Henley, A., Ericson, B. J., Weintrop, D., and Grossman, T. (2024). How novices use llm-based code generators to solve cs1 coding tasks in a self-paced learning environment. In *Proceedings of the 23rd Koli Calling International Conference on Computing Education Research*, Koli Calling '23, New York, NY, USA. Association for Computing Machinery.
- [Kingdom 2018] Kingdom, U. (2018). Data protection act 2018. Accessed: 2025-01-20.
- [Kojima et al. 2022] Kojima, T., Gu, S. S., Reid, M., Matsuo, Y., and Iwasawa, Y. (2022). Large language models are zero-shot reasoners. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NIPS '22, Red Hook, NY, USA. Curran Associates Inc.
- [Libâneo 1994] Libâneo, J. C. (1994). *Pedagogia e pedagogos, para quê?* Cortez, São Paulo.
- [Liffiton et al. 2024] Liffiton, M., Sheese, B. E., Savelka, J., and Denny, P. (2024). Co-dehelp: Using large language models with guardrails for scalable support in programming classes. In *Proceedings of the 23rd Koli Calling International Conference on Computing Education Research*, Koli Calling '23, New York, NY, USA. ACM.
- [Luckesi 2011] Luckesi, C. C. (2011). *Avaliação da aprendizagem escolar: estudos e proposições*. Cortez, São Paulo, 22 edition. Capítulos sobre políticas públicas de avaliação, incluindo o SAEB.
- [Lundberg and Lee 2017] Lundberg, S. M. and Lee, S.-I. (2017). A unified approach to interpreting model predictions. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.
- [McCarthy et al. 1955] McCarthy, J., Minsky, M. L., Rochester, N., and Shannon, C. E. (1955). A proposal for the dartmouth summer research project on artificial intelligence. Memorandum, Dartmouth College. Disponível em: <https://www.jmc.stanford.edu/articles/dartmouth/dartmouth.pdf>.
- [Ministério da Educação do Brasil 2017] Ministério da Educação do Brasil (2017). Base nacional comum curricular. Acesso em: 22 maio 2025.
- [Murphy 2012] Murphy, K. P. (2012). *Machine Learning: A Probabilistic Perspective*. MIT Press, Cambridge, Massachusetts, illustrated edition.
- [Nam et al. 2024] Nam, D., Macvean, A., Hellendoorn, V., Vasilescu, B., and Myers, B. (2024). Using an llm to help with code understanding. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, ICSE '24, New York, NY, USA. Association for Computing Machinery.
- [Nascimento and Silva 2023] Nascimento, B. L. C. and Silva, E. M. (2023). Lei geral de proteção de dados (lgpd) e repositórios institucionais: reflexões e adequações. *Em Questão*, 29:e-127314.

- [Newell et al. 1956] Newell, A., Shaw, J. C., and Simon, H. A. (1956). The logic theory machine – a complex information processing system. In *Proceedings of the Western Joint Computer Conference*, pages 218–230. IRE-AIEE-ACM.
- [Noble 2018] Noble, S. U. (2018). *Algorithms of Oppression: How Search Engines Reinforce Racism*. NYU Press.
- [Núñez et al. 2019] Núñez, C. P., Koslinski, M. C., and Fernández, S. J. (2019). Políticas de incentivo ao uso de dados educacionais: experiências no contexto internacional e brasileiro. *Jornal de Políticas Educacionais*, 13.
- [of Education 1974] of Education, U. D. (1974). Family educational rights and privacy act (ferpa). 20 U.S.C. 1232g; 34 CFR Part 99.
- [Okoye and Hosseini 2024] Okoye, K. and Hosseini, S. (2024). *Correlation Tests in R: Pearson Cor, Kendall's Tau, and Spearman's Rho*, pages 247–277. Springer Nature Singapore, Singapore.
- [Parliament and Council 2016] Parliament, E. and Council (2016). Regulation (eu) 2016/679 of the european parliament and of the council of 27 april 2016 – general data protection regulation. Accessed: 2025-01-20.
- [Peres 2023] Peres, R. S. (2023). Grandes modelos de linguagem na resolução de questões de vestibular: O caso dos institutos militares brasileiros. Mestrado, Universidade Federal do Estado do Rio de Janeiro (UNIRIO).
- [Piaget 1973] Piaget, J. (1973). *A epistemologia genética*. Bertrand Brasil, Rio de Janeiro. Tradução de E. F. Souza. Obra original: *L'épistémologie génétique*, 1950.
- [Provost and Fawcett 2016] Provost, F. and Fawcett, T. (2016). *Data Science Para Negócios*. Editora Alta Books, edição: 1 edition.
- [Qiao et al. 2023] Qiao, S., Ou, Y., Zhang, N., Chen, X., Yao, Y., Deng, S., Tan, C., Huang, F., and Chen, H. (2023). Reasoning with language model prompting: A survey. In Rogers, A., Boyd-Graber, J., and Okazaki, N., editors, *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 5368–5393, Toronto, Canada. Association for Computational Linguistics.
- [Ribeiro et al. 2016] Ribeiro, M. T., Singh, S., and Guestrin, C. (2016). “Why Should I Trust You?”: Explaining the Predictions of Any Classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1135–1144.
- [Rodrigues et al. 2025] Rodrigues, H. S., Moraes, L. O., Garcia, A. C. B., Alvares, R. V., dos Santos, R. P., and Delgado, C. (2025). Investigating artificial intelligence algorithms to predict college students’ academic performance: A systematic mapping study. In *Anais do Workshop de Educação em Computação*, Maceió, Brasil. A ser publicado.

- [Romero and Ventura 2020] Romero, C. and Ventura, S. (2020). Educational data mining and learning analytics: An updated survey. *WIREs Data Mining and Knowledge Discovery*, 10(3):e1355.
- [Russell and et al. 2022] Russell, S. and et al. (2022). Human rights in the age of ai. *UNESCO Report*.
- [Sahoo et al. 2024] Sahoo, P., Singh, A. K., Saha, S., Jain, V., Mondal, S., and Chadha, A. (2024). A systematic survey of prompt engineering in large language models: Techniques and applications. *arXiv*. Available at <https://arxiv.org/abs/2402.07927v1>.
- [Sarlet and Ruaro 2021] Sarlet, G. B. S. and Ruaro, R. L. (2021). A proteção de dados sensíveis no sistema normativo brasileiro sob o enfoque da lei geral de proteção de dados (lgpd) – l. 13.709/2018. *Revista de Direitos Fundamentais & Democracia*, 26(2):81–106.
- [Skinner 1953] Skinner, B. F. (1953). *Science and Human Behavior*. Macmillan, New York.
- [Slattery et al. 2025] Slattery, P., Saeri, A. K., Grundy, E. A. C., Graham, J., Noetel, M., Uuk, R., Dao, J., Pour, S., Casper, S., and Thompson, N. (2025). The ai risk repository: A comprehensive meta-review, database, and taxonomy of risks from artificial intelligence.
- [Sun et al. 2024] Sun, S., Yuan, R., Cao, Z., Li, W., and Liu, P. (2024). Prompt chaining or stepwise prompt? refinement in text summarization. In Ku, L.-W., Martins, A., and Srikumar, V., editors, *Findings of the Association for Computational Linguistics: ACL 2024*, pages 7551–7558, Bangkok, Thailand. Association for Computational Linguistics.
- [Sutton and Barto 2018] Sutton, R. S. and Barto, A. G. (2018). *Reinforcement Learning: An Introduction*. Adaptive Computation and Machine Learning. The MIT Press, Cambridge, Massachusetts, 2nd edition.
- [UNESCO 2021a] UNESCO (2021a). *Artificial Intelligence and Education: Guidance for Policy-makers*. United Nations Educational, Scientific and Cultural Organization, Paris.
- [UNESCO 2021b] UNESCO (2021b). Recommendation on the ethics of artificial intelligence. Disponível em: <https://unesdoc.unesco.org/ark:/48223/pf0000381137>.
- [UNESCO 2023] UNESCO (2023). *Governing Artificial Intelligence in Education: From Principles to Practice*. United Nations Educational, Scientific and Cultural Organization, Paris.
- [UNESCO 2024] UNESCO (2024). Generation ai: Navigating the opportunities and risks of artificial intelligence in education.

- [van Haastrecht et al. 2024] van Haastrecht, M., Brinkhuis, M., and Spruit, M. (2024). Federated learning analytics: Investigating the privacy-performance trade-off in machine learning for educational analytics. In Olney, A. M., Chounta, I.-A., Liu, Z., Santos, O. C., and Bittencourt, I. I., editors, *Artificial Intelligence in Education*, pages 62–74, Cham. Springer Nature Switzerland.
- [Vasconcelos and Clua 2017] Vasconcelos, C. N. and Clua, E. W. G. (2017). Deep learning - teoria e prática. In Delicato, F. C., Pires, P. F., and Silveira, I. F., editors, *Jornadas de Atualização em Informática 2017*, chapter 6. Sociedade Brasileira de Computação.
- [Vygotsky 1984] Vygotsky, L. S. (1984). *A formação social da mente: o desenvolvimento dos processos psicológicos superiores*. Livros Técnicos e Científicos, São Paulo. Tradução de José Cipolla Neto, Luiz Silveira Menna Barreto e Solange Castro Afeche. Obra original de 1930-1934.
- [Wang et al. 2023] Wang, L., Xu, W., Lan, Y., Hu, Z., Lan, Y., Lee, R. K.-W., and Lim, E.-P. (2023). Plan-and-solve prompting: Improving zero-shot chain-of-thought reasoning by large language models. In Rogers, A., Boyd-Graber, J., and Okazaki, N., editors, *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2609–2634, Toronto, Canada. Association for Computational Linguistics.
- [Wei et al. 2022] Wei, J., Wang, X., Schuurmans, D., Bosma, M., Ichter, B., Xia, F., Chi, E. H., Le, Q. V., and Zhou, D. (2022). Chain-of-thought prompting elicits reasoning in large language models. In *Proceedings of the 36th International Conference on Neural Information Processing Systems, NIPS '22*, Red Hook, NY, USA. Curran Associates Inc.
- [Weizenbaum 1966] Weizenbaum, J. (1966). Eliza – a computer program for the study of natural language communication between man and machine. *Communications of the ACM*, 9(1):36–45.
- [Wendling et al. 2023] Wendling, M. I. et al. (2023). Diagnóstico do nível de maturidade da aplicação da Igpd nas escolas de educação infantil da rede municipal de passo fundo. *Revista Brasileira de Políticas Públicas e Direito da Educação*, 15(3):45–62.
- [Wexler et al. 2020] Wexler, J., Pushkarna, M., Bolukbasi, T., Wattenberg, M., Viégas, F., and Wilson, J. (2020). The what-if tool: Interactive probing of machine learning models. *IEEE Transactions on Visualization and Computer Graphics*, 26(1):56–65.
- [Zemel et al. 2013] Zemel, R., Wu, Y., Swersky, K., Pitassi, T., and Dwork, C. (2013). Learning fair representations. In *International Conference on Machine Learning*, pages 325–333. PMLR.
- [Zhang et al. 2018] Zhang, B. H., Lemoine, B., and Mitchell, M. (2018). Mitigating unfairness in machine learning with adversarial learning. In *Proceedings of the 2018 AAAI/ACM Conference on AI, Ethics, and Society*, pages 335–340.

Capítulo

3

Programação Multithread: Modelos e Abstrações em Linguagens Contemporâneas

Gerson Geraldo H. Cavalheiro, Alexandro Baldassin, André Rauber Du Bois

Resumo

Este capítulo oferece uma introdução à programação concorrente, com foco em linguagens que possuem suporte nativo a multithreading. São apresentados os fundamentos da concorrência, os principais modelos de implementação de threads e, em seguida, analisadas as abordagens adotadas por C++, Rust, Go e Elixir. O estudo destaca como cada linguagem organiza suas construções para programação concorrente e inclui dois estudos de caso que ilustram a aplicação prática dos conceitos. O conteúdo serve como referência inicial para aprofundar o domínio técnico e explorar criticamente as potencialidades das linguagens abordadas.

Abstract

This chapter provides an introduction to concurrent programming, focusing on languages with native support for multithreading. It presents the fundamentals of concurrency, the main thread implementation models, and analyzes the approaches adopted by C++, Rust, Go, and Elixir. The study highlights how each language structures its constructs for concurrent programming and includes two case studies that illustrate the practical application of the discussed concepts. The content serves as an initial reference for deepening technical knowledge and critically exploring the capabilities of the featured languages.

3.1. Apresentação

Este texto se apresenta como um material ao apoio ao estudo da programação concorrente e multithread utilizando ferramentas de programação contemporâneas. A proposta é combinar uma abordagem teórica com exemplos práticos, oferecendo uma visão abrangente dos fundamentos, modelos e estratégias adotados para explorar o multithreading em C++, Rust, Go e Elixir.

Este material cobre os conceitos e mecanismos disponíveis nas linguagens escolhidas, destacando seus principais recursos para programação concorrente. O texto apresentado, portanto, não se propõe a realizar comparações de desempenho entre as linguagens nem a discutir, de forma aprofundada, critérios para a escolha de uma linguagem em detrimento de outra, considerando diferentes cenários de aplicação.

Nas seções iniciais, Seção 3.2 e Seção 3.3, são apresentados os fundamentos da programação multithread e as principais estratégias de implementação, com o objetivo de uniformizar a terminologia adotada ao longo do texto. As seções seguintes abordam, em sequência, os modelos de execução, as formas de comunicação entre tarefas e os mecanismos de sincronização disponíveis nas quatro linguagens de programação contemporâneas selecionadas para compor este material: C++ (Seção 3.4), com seu modelo de threads nativos e controle explícito por mutexes; Rust (Seção 3.5), que combina segurança de memória com mecanismos de sincronização seguros por construção; Go (Seção 3.6), cujo modelo baseado em goroutines e canais simplifica a programação concorrente; e Elixir (Seção 3.7), que segue o modelo de atores, promovendo isolamento entre processos e tolerância a falhas. A Seção 3.8 discute a implementação de dois casos de estudo nas diferentes linguagens, e a Seção 3.9 apresenta as considerações finais.

3.2. Fundamentos do Multithreading

A crescente complexidade dos sistemas computacionais e a evolução das arquiteturas de hardware, especialmente com a ubiquidade dos processadores multicore, tornaram a programação concorrente um elemento central no desenvolvimento de software moderno [Sutter and Larus 2005]. Como reflexo, linguagens tradicionais passaram a incorporar mecanismos de concorrência de forma nativa, e as linguagens recentes têm sido concebidas para operar em contextos de programação paralela [McCool et al. 2012, Rauber and Runger 2010].

3.2.1. Motivações e Contexto

A programação concorrente tornou-se indispensável para utilizar efetivamente as arquiteturas multicore, que emergiram como substitutas das arquiteturas monoprocessadas diante das limitações físicas no aumento da frequência dos processadores e da demanda por maior poder computacional [Gupta et al. 2021]. Atualmente, qualquer equipamento computacional, de laptops a smartphones, dispõe de múltiplos núcleos de processamento. No entanto, a simples presença desse recurso não garante ganhos de desempenho: cabe ao software explorar esse paralelismo. Apesar dos avanços no desenvolvimento de novas abstrações, a maioria das linguagens de programação ainda exige que o programador expresse explicitamente o uso de múltiplos fluxos. Isso pode ocorrer por meio de threads, tarefas assíncronas ou outras abstrações [Skillicorn and Talia 1998]. Além disso, é necessário empregar mecanismos para coordenar esses fluxos na cooperação por dados.

Neste contexto, é importante entender com clareza os conceitos de concorrência e paralelismo. Ainda que bastante relacionados, esses conceitos expressam ideias distintas, e é fato que os autores os empreguem de formas diferentes [Pacheco 2022]. A concorrência pode ser entendida pela presença de múltiplas tarefas em progresso em um programa, independentemente de serem executadas no mesmo instante de tempo real (tempo de re-

lógico). Um sistema multitarefa é um exemplo típico: mesmo em uma máquina com apenas um núcleo, diversas tarefas, como leitura de entrada, atualização de tela e execução de processos em segundo plano, podem estar em andamento por meio de revezamento no acesso aos recursos de processamento. Esse comportamento caracteriza um sistema concorrente. Por outro lado, o paralelismo implica a execução simultânea dessas tarefas, com múltiplos núcleos cooperando diretamente na resolução de um problema. Assim, todo programa paralelo é também concorrente [Pacheco 2022].

É também usual associar *concorrência* a uma propriedade da aplicação (um programa descreve atividades concorrentes entre si) e *paralelismo* a uma propriedade do hardware (uma arquitetura possui recursos suficientes para suportar a execução de duas ou mais de atividades simultâneas) [Pacheco 2022][Herlihy et al. 2020]. Assim, é comum que a expressão “programação concorrente” se refira à programação voltada à exposição da concorrência da aplicação, enquanto “programação paralela” seja relacionada à programação focada em extrair o máximo de desempenho de uma determinada arquitetura.

Neste texto, a distinção entre concorrência e paralelismo, assim como entre programação concorrente e paralela não é destacada. De toda forma, o leitor deve observar que o enfoque é dado para a exploração da programação multithread pela exposição da concorrência de atividades concorrentes nos programas e expectativa de execução paralela sobre o hardware multiprocessado disponível.

As aplicações que se beneficiam de múltiplos fluxos são numerosas e crescentes. Sistemas embarcados, como os de controle automotivo, precisam lidar com sensores, atuadores e lógica de controle em tempo real. Servidores web modernos atendem centenas de requisições simultâneas, exigindo alto grau de concorrência. Ambientes de computação científica e aprendizado de máquina exploram paralelismo para reduzir o tempo de programação de tarefas computacionalmente intensivas. Em todos esses contextos, o domínio de técnicas de programação multithread é um diferencial essencial.

3.2.2. Infraestrutura de Suporte ao Multithreading

O estudo da programação multithread segue-se aos estudos das arquiteturas de computadores e dos sistemas operacionais. Embora a disponibilidade de uma arquitetura paralela não seja um requisito essencial para a programação multithread, a onipresença de processadores multicore torna praticamente inconcebível o desenvolvimento de um programa em que não seja prevista sua programação sobre uma arquitetura multiprocessada. Da mesma forma, os sistemas operacionais atuais são capazes de gerenciar centenas, e até milhares, de fluxos de programação nas diversas unidades de processamento providas por tais arquiteturas [Tanenbaum and Bos 2022][Silberschatz et al. 2018].

As arquiteturas paralelas consideradas neste texto tratam-se de multiprocessadores [Culler et al. 1999], compostos por múltiplos núcleos de processamento compartilhando um espaço de endereçamento comum. Este modelo se apresenta em duas organizações arquiteturais: UMA (Uniform Memory Access) e NUMA (Non-Uniform Memory Access). No modelo UMA, presente em equipamentos como notebooks e desktops, todos os núcleos compartilham uma única memória principal, na qual o acesso a qualquer posição de memória a partir de qualquer núcleo se dá em tempo constante. Nas arquiteturas NUMA, típicas em servidores de maior porte, o acesso à memória é segmentado em regiões, cada

uma mais próxima a determinados núcleos, o que reduz a contenção no acesso à memória e permite uma escalabilidade mais eficiente em termos de número de núcleos disponíveis. Embora mecanismos de escalonamento e estratégias de programação possam explorar de forma eficiente as características de uma arquitetura NUMA, este texto se concentra, de forma geral, no modelo básico de programação sobre uma arquitetura multiprocessada.

No sistema operacional, a abstração fundamental para a programação de programas é o processo. Cada processo possui seu próprio espaço de endereçamento e contexto de programação, sendo representados no sistema operacional por uma estrutura de dados chamada PCB (Process Control Block). Um processo sequencial convencional possui um único fluxo de execução, denominado neste texto como thread. Um processo, ao lançar múltiplos fluxos de execução é dito multithread. Os threads, por sua vez, são considerados processos leves em contraste aos processos convencionais, mais pesados para serem manipulados pelo sistema operacional. Threads compartilham o espaço de endereçamento e outras estruturas com os demais threads do mesmo processo, o que torna sua criação e escalonamento mais eficientes. O padrão POSIX Threads (Pthreads) é implementado nativamente nos principais sistemas operacionais modernos, como Linux, macOS e Windows, e serve de base para diversas linguagens e bibliotecas de programação concorrente.

O suporte ao multithreading também se estende ao hardware. Processadores modernos incluem mecanismos que garantem a atomicidade de certas operações, evitando interferências indesejadas entre threads que compartilham dados. Instruções atômicas, como `compare-and-swap` ou `test-and-set`, são fundamentais para a implementação de estruturas de sincronização como mutexes e semáforos. Além disso, recursos como registradores de controle, caches com políticas de coerência e instruções de barreira de memória contribuem para a programação eficiente de programas concorrentes em ambientes com múltiplos núcleos.

3.2.3. Princípios da Programação Concorrente

A programação concorrente lida com a coordenação de múltiplas unidades de programação com o objetivo de aproveitar os recursos disponíveis em sistemas computacionais com múltiplos núcleos. Para compreender as diferentes formas de concorrência, é importante distinguir os conceitos de processo, thread e tarefa, utilizados para determinar o encapsulamento de uma unidade de trabalho nos programas. Um processo é uma instância isolada de execução, com seu próprio espaço de endereçamento e recursos de sistema, como descritores de arquivos e registradores. Os threads, por sua vez, são unidades de programação mais leves que compartilham o espaço de memória e recursos do processo ao qual pertencem. Essa leveza torna o uso de threads mais eficiente em termos de criação, comunicação e escalonamento, ainda que traga a necessidade de mecanismos de sincronização para lidar com o acesso simultâneo a dados compartilhados na memória do processo. Já as tarefas representam unidades lógicas de trabalho, independentes em termos conceituais, mas que podem ser implementadas como threads, processos ou até corrotinas, dependendo do ambiente de programação adotado.

Além da especificação das unidades de trabalho, a sincronização é um dos principais desafios na programação concorrente [Herlihy et al. 2020]. Como múltiplos threads podem acessar simultaneamente variáveis ou estruturas de dados compartilhadas, é neces-

sário coordenar esses acessos para evitar inconsistências. A exclusão mútua garante que apenas um thread acesse determinada região crítica de código por vez. Mecanismos como mutexes, semáforos e barreiras são amplamente utilizados para esse fim, sendo estes os principais recursos para coordenação de atividades em ferramentas multithread clássicas, como Pthreads [Kleiman et al. 1996] e OpenMP [Chandra et al. 2001], e também em C++ e Rust. Quando a sincronização é negligenciada, surgem problemas como condições de corrida, em que o comportamento do programa depende da ordem de execução dos threads e pode levar a resultados incorretos ou imprevisíveis.

Uma alternativa à sincronização por exclusão mútua é a comunicação por troca de mensagens, na qual threads ou tarefas interagem por meio de canais de comunicação. Nesse modelo, em vez de múltiplas unidades de programação acessarem diretamente uma região compartilhada da memória, os dados são explicitamente enviados e recebidos entre elas. Isso favorece um estilo de programação em que a sincronização ocorre como parte da comunicação, reduzindo os riscos de condições de corrida e simplificando o raciocínio sobre a concorrência. Linguagens como Go e Elixir adotam esse modelo como principal mecanismo de interação entre tarefas, mas muitas bibliotecas em linguagens tradicionais também oferecem suporte a canais.

Outro aspecto importante diz respeito à granularidade, tanto da unidade de trabalho quanto das seções críticas. A granularidade em relação a unidade de trabalho refere-se ao comprimento, em termos de número de operações, que as unidades de trabalho descrevem [McCool et al. 2012] [Rauber and Rünger 2010]. Unidades de trabalho com poucas operações, sobre-expondo a concorrência da aplicação, podem gerar sobrecarga, devido ao custo de gerenciamento do grande número de atividades geradas. Por outro lado, unidades muito longas podem limitar o aproveitamento do paralelismo disponível, serializando a execução do programa e, potencialmente, não explorando eficientemente os recursos de hardware disponíveis. Essa limitação está relacionada à Lei de Amdahl [Amdahl 1967], segundo a qual o ganho de desempenho com paralelismo é limitado pela fração sequencial do programa.

A granularidade das seções críticas [Herlihy et al. 2020][McCool et al. 2012] possui outra natureza e diz respeito à quantidade de recursos (dados) adquiridos para execução de uma determinada seção crítica, influenciando diretamente o tempo de bloqueio. Neste caso, uma abordagem de granularidade grossa utiliza um único ponto de sincronização, um mutex, por exemplo, para proteger diversos dados compartilhados. Esta estratégia simplifica o código, embora aumente a possibilidade de serialização indesejada, uma vez que pode forçar a exclusividade ao acesso a um dado não utilizado em determinada seção crítica, mas necessário em outra. Por outro lado, uma granularidade fina, por exemplo, individualizando cada dado compartilhado a um mutex distintos, tem potencial para ampliar o paralelismo que pode ser explorado, mas acaba por elevar a complexidade do programa ao mesmo tempo em que eleva também o custo computacional de manipulação dos diversos recursos de sincronização.

Por fim, a programação concorrente está sujeita a problemas clássicos de implementação [McCool et al. 2012][Herlihy et al. 2020], como condição de corrida, deadlock e starvation. O deadlock ocorre quando dois ou mais threads ficam bloqueados indefinidamente esperando por recursos que nunca serão liberados. O starvation se dá quando

um thread é repetidamente preterido no acesso a recursos, não conseguindo progredir. A prevenção e mitigação desses problemas requer estratégias específicas, como ordenação consistente de aquisição de recursos, uso de algoritmos de escalonamento justos e aplicação de técnicas de detecção e recuperação.

3.2.4. Panorama das Linguagens Abordadas

A concepção de multithreading surgiu no contexto de sistemas operacionais, como uma estratégia para permitir a existência de múltiplos fluxos de controle dentro de um mesmo processo, visando melhorar a responsividade e o aproveitamento dos recursos disponíveis. A biblioteca POSIX Threads (pthreads) [Kleiman et al. 1996] tornou-se uma representação canônica dessa abordagem, fornecendo uma API padronizada e portátil para criação, sincronização e gerenciamento de threads no nível do sistema. Ainda que ferramentas oferecidas na forma de uma biblioteca não sejam as mais convenientes ao desenvolvimento de programas multithread [Boehm 2005], é fato que ela se popularizou neste uso. A linguagem Ada [Burns and Wellings 1998] nos anos 1970, por sua vez, introduziu conceitos de concorrência no nível da linguagem de programação, oferecendo suporte a tarefas e ao mecanismo de sincronização por *rendezvous* (mecanismo semelhante à troca de mensagens ou canais, que não utiliza dados em memória para comunicação). Já no final do século 20, emerge o padrão OpenMP, resultado de um esforço coordenado para fornecer uma interface de programação de alto nível voltada à exploração eficiente de paralelismo em arquiteturas multiprocessadas, especialmente em aplicações científicas.

Pthreads, OpenMP e Ada, entre outras, são as ferramentas de programação frequentemente utilizadas para o ensino da programação concorrente [Silva et al. 2022] [Pacheco 2022] [Andrews 1999] [Cavaleiro and Santos 2007] [Cavalheiro 2009] [Pilla et al. 2009][Cavaleiro and Du Bois 2014], ainda que Ada em menor escala atualmente. No entanto, o mercado está repleto de novas abordagens para programação concorrente, disponibilizadas em novas linguagens de programação ou em linguagens de programação consolidadas, revisitadas no atual contexto da ubiquidade dos multiprocessadores pela ploriferação das arquiteturas de processadores multicore. Neste texto são abordadas quatro dessas novas abordagens, oferecidas por C++, Rust, Go e Elixir. Cada uma dessas linguagens adota estratégias distintas quanto à unidade de concorrência, formas de comunicação entre tarefas e mecanismos de sincronização.

O C++, nas suas versões a partir de C++11, utiliza um modelo baseado em threads nativos, acessados por meio da biblioteca `std::thread`. A criação, o controle e a sincronização dos threads são de responsabilidade do programador, que deve recorrer a primitivas como mutexes e locks para garantir exclusão mútua em regiões críticas. A comunicação entre threads se dá por memória compartilhada, exigindo sincronização cuidadosa para evitar problemas como condições de corrida e deadlocks. As versões mais recentes de C++ (a atual especificação estável é a C++23, embora a maioria dos compiladores esteja suportando efetivamente a versão C++20) incorporaram muitas funcionalidades ao suporte originalmente oferecido por C++11.

Rust também oferece threads nativos, mas incorpora mecanismos de segurança baseados em seu sistema de tipos. As regras de ownership e borrowing, verificadas em tempo de compilação, restringem o compartilhamento de dados entre threads, prevenindo

erros comuns em modelos de memória compartilhada. A linguagem ainda oferece suporte a tarefas assíncronas escalonadas por um runtime leve, permitindo tanto concorrência baseada em threads quanto orientada a eventos.

Go adota um modelo centrado em goroutines, unidades leves de concorrência gerenciadas pela linguagem. A comunicação entre essas unidades ocorre preferencialmente por meio de canais (`chan`), promovendo o compartilhamento seguro de dados por troca de mensagens. Embora o uso de memória compartilhada e sincronização explícita com mutexes seja possível, a abordagem recomendada favorece canais, refletida no lema: “não comunique compartilhando memória; compartilhe memória comunicando”.

Elixir segue o modelo de atores, herdado da máquina virtual Erlang. Cada ator é um processo leve e isolado, com sua própria mailbox para recebimento de mensagens. Toda comunicação ocorre por meio de troca assíncrona de mensagens, sem memória compartilhada. O modelo é supervisionado, permitindo que falhas em processos sejam detectadas e tratadas por supervisores, o que favorece a criação de sistemas robustos e tolerantes a falhas.

3.3. Implementações e Modelos

Embora a ideia de ferramentas para programação multithread remeta, em geral, a modelos baseados na exploração do espaço de endereçamento compartilhado em arquiteturas multiprocessadas, diferentes ferramentas podem adotar estratégias variadas de implementação e oferecer modelos de programação distintos. Nesta seção, são discutidas tanto as abordagens de implementação dessas ferramentas quanto os principais modelos de programação multithread que elas proporcionam.

3.3.1. Modelos de Implementação de Ferramentas

A implementação de ferramentas para programação multithread envolve diferentes estratégias de gerenciamento das unidades de concorrência, tarefas ou threads, criadas pelo programador. Essas estratégias estão associadas a decisões quanto ao nível de abstração oferecido, ao grau de envolvimento com o sistema operacional e à política de escalonamento adotada. Em linhas gerais, podem ser identificados dois modelos principais [Kleiman et al. 1996]: o modelo 1×1 , no qual cada unidade de concorrência do programa corresponde diretamente a um thread do sistema operacional, e o modelo $N \times M$, no qual as N unidades de concorrência lançadas pelo programa são gerenciadas em espaço de usuário e distribuídas sobre M threads do sistema providas pelo runtime da ferramenta (normalmente, N é muito maior do que M). Até o advento dos multicores, ferramentas de programação também eram implementadas sobre o modelo $N \times 1$, em que todas as unidades de concorrência lançadas por um programa eram multiplexadas sobre um único thread. No entanto, com a virtual extinção das plataformas monoprocessadas, esse modelo deixou de ser relevante, e é raramente adotado por ferramentas contemporâneas.

No modelo 1×1 , também conhecido como modelo thread sistema, cada unidade de execução lançada pelo programa em execução é diretamente gerenciada pelo sistema operacional. Em termos de nomenclatura, usualmente as ferramentas de programação implementadas sobre este modelo adotam o termo *thread* para designar a unidade que descreve a concorrência no programa. Também é comum que os threads criados sejam

individualmente identificados e esta identificação conhecida como retorno da operação de criação de um thread.

A estrutura de implementação de ferramentas sob o modelo 1×1 não requer a introdução de estratégias de mapeamento dos threads sobre os recursos de processamento disponíveis e a gestão de sua execução, uma vez que a infraestrutura de escalonamento, sincronização e controle de recursos é delegada ao sistema operacional. Ferramentas neste modelo oferecem grande liberdade para construção de algoritmos, uma vez que, de posse do identificador de um thread, seja este representado por uma variável ou um objeto, qualquer ponto do programa pode sincronizar com o thread em execução, tendo este thread sido criado neste ponto, ou não.

Como exemplos de ferramentas 1×1 , este texto aborda as linguagens C++ e Rust que são implementadas sob este modelo, ainda que extensões de Rust promovam o uso do modelo $N \times M$ discutido a seguir. Outra ferramenta implementada neste modelo é suportada nas principais implementações do padrão POSIX threads (Pthreads) e também Java, desde sua versão 1.0.

O modelo $N \times M$ introduz uma camada de abstração que intermedia as unidades de concorrência do programa, usualmente referenciadas como *tarefas* e um conjunto de unidades de execução em nível de sistema operacional (threads sistema) mantido pela ferramenta em um núcleo de execução. Nesse caso, o runtime da linguagem é responsável tanto pelo escalonamento das tarefas sobre os threads do núcleo de execução, delegando ao sistema operacional a responsabilidade do mapeamento dos threads sobre os recursos físicos de processamento. Essa abordagem permite que operações de criação, suspensão e retomada de execução ocorram inteiramente no espaço de usuário, reduzindo significativamente o custo associado a essas operações. Como consequência, torna-se viável lidar com milhares ou até milhões de unidades de concorrência leves. Linguagens como Go e Elixir exemplificam esse modelo, assim como OpenMP e também Java a partir de sua versão 21, como recurso Virtual Threads.

Tabela 3.1: Comparação entre os modelos de implementação 1×1 e $N \times M$

Aspecto	Modelo 1×1	Modelo $N \times M$
Unidade de Execução	Thread	Tarefa
Custo (comparado)	Maior	Menor
Mapeamento	1 unidade de concorrência \rightarrow 1 thread do sistema	N unidades \rightarrow M threads do sistema (com $N > M$)
Escalonamento	Sistema operacional	Ambiente de execução (runtime)
Criação e troca de contexto	Mais custosas, dependem do sistema operacional	Mais leves, realizadas em espaço de usuário
Escalabilidade	Limitada por overhead do sistema e número de threads	Alta, viabiliza milhares ou milhões de unidades leves
Integração com ferramentas do sistema	Total (visibilidade de threads nativos)	Parcial ou ausente (threads não visíveis ao sistema)
Exemplos de linguagens	Pthreads, C++, Rust	OpenMP, Go, Elixir

Uma vez que a criação de tarefas no modelo $N \times M$ tem menor custo que a criação de threads no modelo 1×1 , é usual que ferramentas de programação implementadas neste modelo sejam a opção para aplicações que necessitem expor um grande número de atividades concorrentes. A Tabela 3.1 apresenta uma comparação entre estes dois modelos.

3.3.2. Modelos para Programação Multithread

Independente do modelo de implementação adotado pela ferramenta de programação, estas oferecem diferentes modelos de programação. Esses modelos podem ser classificados em duas grandes categorias [Rauber and Rüniger 2010]: compartilhamento de memória e troca de mensagens.

Embora seja comum que ferramentas de programação forneçam recursos de programação a ambos os modelos, memória compartilhada e troca de mensagens, as vocações das ferramentas se posicionam, majoritariamente, sobre um ou outro. Como atrativo das ferramentas baseadas em memória compartilhada, destaca-se a simplicidade da comunicação por meio de escritas e leituras em um espaço de endereçamento comum. Esse modelo, no entanto, exige que o programador gerencie a sincronização entre os fluxos de execução concorrentes. Já a troca de mensagens incorpora sincronização implícita entre as partes comunicantes, ao custo de gerenciar o endereçamento e a entrega das mensagens.

Modelos baseados em compartilhamento de memória são os mais tradicionais e consistem na execução concorrente de múltiplas threads que acessam um espaço de memória comum. Um exemplo típico é o modelo *thread + lock*, adotado por bibliotecas como POSIX Threads, Java Threads, `std::thread` em C++ e as primitivas de `std::thread` e `Mutex` em Rust. Nessa abordagem, a sincronização entre as unidades concorrentes é feita explicitamente por meio de mecanismos como mutexes, semáforos e variáveis de condição. Embora ofereça grande flexibilidade, esse modelo exige cuidado redobrado do programador para evitar condições de corrida, impasses e outros erros difíceis de depurar.

Também compõe parte dos modelos baseados em compartilhamento de memória o paralelismo de dados, no qual múltiplas threads executam o mesmo conjunto de operações sobre diferentes partes de uma estrutura de dados. Nesse paradigma, o foco não está na divisão do programa em tarefas distintas, mas na divisão dos dados a serem processados, permitindo que cada thread opere sobre uma fatia independente do conjunto de entrada. Essa abordagem é especialmente eficiente em aplicações de natureza numérica ou científica, como simulações físicas, álgebra linear e processamento de imagens, quando grandes volumes de dados homogêneos devem ser manipulados de forma iterativa.

Linguagens e bibliotecas que oferecem suporte ao paralelismo de dados geralmente fornecem construções que abstraem o particionamento dos dados e a criação das threads. Em C++, a biblioteca `Parallel STL`, introduzida no C++17, permite aplicar algoritmos padrão, como `std::sort` ou `std::transform`, com execução paralela utilizando a política `std::execution::par`. Em Java, a API `Stream` com `.parallel()` permite aplicar transformações a coleções de forma concorrente, com a divisão automática dos dados e execução em múltiplos núcleos. No contexto do OpenMP, o paralelismo de dados é frequentemente expresso com a diretiva `#pragma omp for`, que distribui as iterações de um laço entre múltiplos threads, cada um responsável por

uma fatia do intervalo de índices.

Essa forma de paralelismo tende a ser mais previsível e escalável do que modelos centrados na coordenação entre tarefas, já que minimiza a necessidade de sincronização entre as threads. No entanto, o paralelismo de dados pressupõe que as operações sobre diferentes porções do conjunto sejam independentes entre si, o que nem sempre é possível ou trivial de garantir. Uma variação desse paradigma é o modelo de regiões paralelas ou paralelismo estruturado, em que o programador especifica blocos de código a serem executados em paralelo, delegando à ferramenta a responsabilidade por criar, distribuir e sincronizar as threads. OpenMP é o principal exemplo dessa abordagem, permitindo que diretivas de compilação definam paralelismo sobre laços, seções ou tarefas sem necessidade de gerenciamento explícito de threads.

Como alternativa dentre os modelos de programação baseados em memória compartilhada, encontra-se o paralelismo de tarefas. Nessa estratégia, a ênfase recai na decomposição lógica do trabalho em tarefas independentes, sem que o programador precise controlar as threads que as executam. Ferramentas como `std::async` em C++, o módulo `Task` em Elixir, o suporte a tarefas em OpenMP e os executores em Java com `CompletableFuture` ou `ExecutorService` exemplificam essa abordagem.

Modelos de programação concorrente baseados em troca de mensagens também são largamente explorados em arquiteturas multiprocessadas. Neste modelo, apesar de fluxos de execução compartilharem um espaço de endereçamento comum, é evitado o compartilhamento de dados em memória, sendo privilegiada a comunicação explícita entre unidades de execução autônomas. O modelo de atores é uma das formas mais conhecidas, no qual cada ator encapsula seu próprio estado e responde a mensagens assíncronas. Esse paradigma é adotado pela máquina virtual Erlang, utilizada por Elixir, onde processos leves reagem a mensagens sem acesso compartilhado à memória.

Outro modelo de troca de mensagens relevante é o de processos sequenciais comunicantes (CSP), que fundamenta a programação em canais de comunicação entre unidades concorrentes. Go adota esse modelo por meio da combinação de *goroutines* e *channels*, permitindo que processos concorrentes interajam de forma sincronizada e segura por meio de trocas explícitas de mensagens.

3.3.2.1. Abordagens Funcionais

Além dos modelos tradicionais de programação multithread baseados em compartilhamento de memória ou troca de mensagens, algumas ferramentas contemporâneas oferecem suporte a abordagens funcionais [Thomas 2022]. Essas abordagens propõem uma mudança de paradigma, eliminando a necessidade de compartilhamento de estado entre unidades de execução, favorecendo o isolamento e a imutabilidade dos dados.

Um exemplo é a incorporação de construtores nas linguagens para os padrões paralelos de *pipelines* e *streams*, nos quais transformações são aplicadas de maneira encadeada a uma sequência de dados. Em vez de explicitar a divisão de trabalho entre múltiplas unidades de execução, o programador compõe funções puras, ou seja, de funções que operem apenas sobre os parâmetros recebidos e retornem exclusivamente o resultado da

computação sobre estes parâmetros. Estas funções operam sobre elementos de um espaço de dados de forma independente, permitindo a exploração do paralelismo. A `Stream` API de Java oferece essa abordagem. Com o uso do modificador `.parallel()`, pipelines declarativos como `stream.map(...).filter(...).collect(...)` podem ser automaticamente paralelizados pela plataforma.

De forma análoga, linguagens funcionais como Elixir promovem a composição de pipelines com o operadores `|>`, em que listas ou fluxos de dados são transformados por meio de funções puras. Bibliotecas como `Flow` permitem que esses pipelines sejam executados de forma concorrente e distribuída, utilizando internamente múltiplos processos leves (baseados no modelo de atores). Em cenários de processamento de eventos em larga escala, frameworks como `Broadway` oferecem uma abstração de fluxo concorrente que combina particionamento, paralelismo e tolerância a falhas de maneira declarativa.

Linguagens como C++ e Rust também têm incorporado elementos desse paradigma. No C++17, a `Parallel STL` permite aplicar algoritmos sobre coleções com políticas como `std::execution::par`, sem exigir controle direto de threads. Em Rust, bibliotecas como `rayon` oferecem iteradores paralelos que preservam a semântica funcional e facilitam a paralelização segura sobre coleções.

3.3.3. Foco nas Linguagens Analisadas

Nesta seção foram apresentados os principais modelos empregados na implementação de programas multithread, com foco nas abordagens oferecidas por C++, Rust, Go e Elixir. A Tabela 3.2 resume como essas linguagens estruturam suas abstrações para os principais aspectos da programação concorrente. Ainda que todas ofereçam meios para adotar diferentes modelos de concorrência, cada uma revela uma vocação predominante, delineada por decisões de projeto que influenciam diretamente o estilo de programação incentivado. As seções seguintes, dedicadas ao estudo individual dessas linguagens, têm por objetivo destacar essas vocações e discuti-las em maior profundidade.

Tabela 3.2: Modelos de concorrência nas linguagens abordadas.

Aspecto	C++	Rust	Go	Elixir
Unidade	Thread nativa	Thread nativa / Tarefa assíncrona	Goroutine	Processo leve (actor)
Comunicação	Memória compartilhada	Memória compartilhada (com propriedade e empréstimo)	Troca de mensagens e memória compartilhada	Troca de mensagens
Mecanismo	<code>std::thread</code>	<code>std::thread</code> / <code>thread::spawn</code>	<code>go</code> (goroutines)	<code>spawn</code> (processos leves)
Mapeamento	Manual	Automática (via tarefa)	Automática	Supervisionada
Sincronização	Seções críticas (mutexes, locks)	Exclusão mútua + garantias de compilação (propriedade e empréstimo)	Canais; mutexes opcionais	Mailbox com isolamento

Complementando a sumarização da presente seção, a Tabela 3.3 apresenta alguns dos principais recursos disponibilizados pelas ferramentas estudadas para a implementação dos diferentes modelos de concorrência discutidos ao longo do texto. Embora não exaustiva, a tabela oferece um panorama representativo das possibilidades técnicas e estilísticas favorecidas por cada linguagem. Estes recursos estão entre aqueles discutidos para caracterizar cada ferramenta.

Tabela 3.3: Abordagens para controle da concorrência em ferramentas de programação multithread.

Abordagem	Ferramenta	Forma de Implementação
Seções críticas	C++	<code>std::thread</code> , <code>std::mutex</code> , <code>std::lock_guard</code> , <code>std::shared_mutex</code>
	Rust	<code>thread::spawn</code> , <code>std::sync::Mutex</code> , <code>std::sync::RwLock</code> , <code>std::sync::Arc</code>
	Go	Pacote <code>sync</code> com <code>Mutex</code> , <code>WaitGroup</code> , <code>RWMutex</code>
Paralelismo de Tarefas	C++	<code>std::async</code> com <code>std::future</code>
	Rust	<code>thread::spawn</code> com <code>JoinHandle</code> ; uso de <code>move</code> para paralelismo leve
Modelo de Atores	Elixir	Cada processo leve atua como ator. Uso de primitivas <code>send/receive</code> , e módulos como <code>GenServer</code>
	Elixir	Supervisores organizam os processos em árvores de supervisão com tolerância a falhas
Troca de Mensagens	Go	Goroutines com canais (<code>chan</code>), síncronos ou com buffer
	Rust	Biblioteca <code>std::sync::mpsc</code> e alternativa <code>crossbeam::channel</code>
Paralelismo de Dados	Rust	Biblioteca <code>rayon</code> com <code>par_iter()</code>
Funcional	Elixir	Imutabilidade; bibliotecas <code>Flow</code> (pipelines concorrentes) e <code>Broadway</code> (fluxos concorrentes tolerantes a falhas)

3.4. C++

C++¹ [Williams 2019] é uma linguagem criada na década de 1980 e passou por sucessivas revisões que ampliaram significativamente suas capacidades expressivas e operacionais. No contexto da programação multithread, o padrão C++11 trouxe a introdução da classe `std::thread`, permitindo a criação e manipulação de threads nativas de forma mais simples e com um ciclo de vida bem definido. Além disso, C++11 também trouxe recursos como mecanismos de sincronização e operações atômicas. Em versões subsequentes, como C++14, melhorias incrementais foram feitas no suporte a paralelismo, enquanto o C++17 introduziu o conceito de algoritmos paralelos integrados à STL. Nesse novo contexto, as operações da STL passaram a permitir a paralelização direta por meio de políticas de execução oferecendo recursos para programação paralela sem a necessidade de um controle explícito sobre os threads. O C++20, por sua vez, expandiu ainda mais o suporte a paralelismo e sincronização, trazendo novas primitivas como barreiras e travas de contagem, além de aprimorar ainda mais a exploração de paralelismo na biblioteca

¹<https://isocpp.org>

padrão.

3.4.1. Modelo Básico

Em C++, o suporte nativo a threads é realizado por meio da classe `std::thread`. A criação de um thread ocorre pela construção de um objeto dessa classe, ao qual deve ser associado um objeto invocável, como uma função, um functor ou uma expressão lambda. Assim que o objeto `std::thread` é instanciado, o thread associado inicia sua execução imediatamente, sendo o objeto invocável executado sobre este thread. Esse modelo permite integrar o gerenciamento de threads à programação orientada a objetos da linguagem, conferindo ao thread o comportamento de um objeto comum, com ciclo de vida e escopo bem definidos. O Código 1 ilustra a criação com as diferentes formas de objetos invocáveis. Neste código são criados três threads, `t1`, `t2` e `t3`, cada thread sendo criado com um tipo de objeto invocável diferente: uma função, um functor (representado pelo método `operator()` na classe `Functor`, e uma expressão lambda.

O controle do thread criado se dá por meio do próprio objeto `std::thread`. É responsabilidade do programador finalizar explicitamente o thread por meio de uma chamada ao método `join()`, que faz com que o fluxo chamador aguarde o término do thread, assegurando sua conclusão antes da continuação do programa. Alternativamente, pode-se utilizar o método `detach()`, que dissocia o objeto do thread em execução, permitindo que ele continue sua programação de forma independente, sendo sua finalização gerenciada pelo sistema operacional. Caso nenhum dos dois métodos seja chamado antes da destruição do objeto `std::thread`, o programa será abruptamente encerrado com uma chamada a `std::terminate()`.

Os threads criados em C++ podem ser considerados *nomeados*, uma vez que o objeto que os encapsula possui um identificador explícito no escopo da função onde é criado; assim, todos os threads são identificados individualmente. Além disso, quando o objeto `std::thread` está em estado *joinable*, ou seja, não foi *detached*, ele pode ser submetido a uma e apenas uma sincronização por chamada a `join()`. Após essa chamada, o thread é considerado finalizado, e o objeto não é mais *joinable*.

O uso de `std::thread` segue o princípio de RAII (Resource Acquisition Is Initialization), no qual a alocação e liberação de recursos são associadas ao ciclo de vida de objetos. Nesse modelo, um thread é criado na construção do objeto `std::thread` e deve ser adequadamente finalizado antes que esse objeto seja destruído. Caso contrário, o programa será encerrado de forma abrupta. O RAII reforça a ideia de que a responsabilidade pela aquisição e liberação dos recursos do thread é do próprio objeto, permitindo que o controle sobre o ciclo de vida do thread seja exercido de forma segura e previsível. Isso reduz a propensão a erros como vazamento de recursos, como memória e mecanismos de sincronização, evitando panes no programa, condições de corrida ou terminação inesperada, e favorecendo uma estruturação mais clara do programa concorrente.

3.4.2. Sincronizações e Compartilhamento de Dados

Em C++, toda comunicação entre threads se dá por meio do compartilhamento de dados na memória. Isso implica que os escopos de vida das variáveis compartilhadas devem estar ativos enquanto as threads acessam tais dados. Como consequência, múltiplas refe-

```

void foo() {
    std::cout << "Executando uma função foo." << std::endl;
}
class Functor {
public:
    void operator() () {
        std::cout << "Executando um functor." << std::endl;
    }
};
int main() {
    Functor f;
    std::thread t1(foo);
    std::thread t2(f);
    std::thread t3([]() {
        std::cout << "Executando uma função lambda." << std::endl;
    });
    // ...
    t1.join();
    t2.join();
    t3.join();
    return 0;
}

```

Código 1: Modelo básico de criação de threads em C++

rências simultâneas a uma mesma região de memória podem ocorrer, exigindo mecanismos de controle de acesso para garantir a integridade dos dados e evitar comportamentos indeterminados.

O controle eficiente de acesso a dados compartilhados pode ser realizado por meio de operações atômicas, disponibilizadas pela na classe `std::atomic`. Esta classe oferece operações realizadas diretamente sobre os recursos de hardware para operações atômicas, garantindo que uma modificação em uma variável ocorra de forma indivisível. Isso impede que outras threads interfiram no meio de uma operação, assegurando a consistência dos dados. No entanto, as operações atômicas são aplicáveis a dados de uma única palavra de memória, como um inteiro, o que as tornam adequadas principalmente para contadores, flags e indicadores de estado. As variáveis atômicas suportam operações como incremento, troca e comparação-condicional (`compare_exchange_strong`), sendo muito eficientes, pois não envolvem bloqueios. Um exemplo de uso da atomicidade em C++ pode ser ilustrada por:

```

std::atomic<int> valor(10);
int esperado = 10, novo_valor = 20;

```

```

if (valor.compare_exchange_strong(esperado, novo_valor))
    std::cout << "Sucesso, valor = " << valor.load() << std::endl;
else
    std::cout << "Falha, valor = " << valor.load()
                << ". Esperado valor: " << esperado << std::endl;

```

Neste exemplo de uso de `atomic`, a variável `valor` é protegida contra condições de corrida por meio do uso da função atômica `compare_exchange_strong`, que realiza uma substituição condicional: ela apenas altera o valor caso ele ainda seja igual ao esperado. Essa abordagem é comum em algoritmos *lock-free*, em que o controle de concorrência é feito sem o uso de bloqueios explícitos. A classe `std::atomic` é uma classe template, cujo tipo de dado é especificado entre os sinais de menor e maior, como em `std::atomic<int>`. O tipo fornecido deve ser um tipo trivial de dados, isto é, tipos escalares como `int`, `bool`, `char`, ponteiros (`T*`) ou tipos enumerados. A biblioteca também oferece especializações para ponteiros e suporte adicional por meio da classe `std::atomic_flag`, útil para operações ainda mais básicas de sinalização.

Para seções críticas com construções mais elaboradas, envolvendo um número maior de operações ou estruturas de dados mais complexas do que as suportadas pelos templates ao `std::atomic`, é indicado o uso de `std::mutex`. Este mecanismo de sincronização permite ao programador delimitar seções de código, ditas seções críticas, a serem executadas em regime de exclusão mútua com outras seções críticas sincronizadas pelo mesmo mutex. Um mutex deve ser explicitamente adquirido, com uma chamada ao método `lock()`, antes do acesso a dados compartilhados. Após o uso deve ser explicitamente liberado, pela invocação da operação `unlock()`. A omissão do desbloqueio, por falha lógica ou desvio de controle (como exceções), pode comprometer toda a aplicação. Para evitar esse risco, é recomendável o uso de mecanismos de guarda. Em comparação com variáveis atômicas, o uso de mutex não é tão eficiente, embora ofereça maior flexibilidade e permita seções críticas complexas, envolvendo múltiplas variáveis.

O `std::lock_guard` é um objeto RAII que adquire o mutex na construção e o libera automaticamente na destruição. Sua simplicidade o torna ideal para escopos curtos e bem definidos. Já o `std::unique_lock` oferece maior flexibilidade, permitindo adiar a aquisição, liberar e reacquirir o mutex manualmente, ou combiná-lo com variáveis de condição. Ambos os mecanismos evitam vazamentos de locks mesmo em situações de exceção.

Em situações em que múltiplos mutexes precisam ser adquiridos simultaneamente, a chamada direta à operação `lock()` sobre objetos mutex em ordem arbitrária pode levar a deadlocks, especialmente se threads distintas tentarem adquirir os mesmos mutexes em ordens diferentes. Para tratar esse problema, o C++ oferece a função `std::lock`, que realiza a aquisição conjunta de dois ou mais mutexes de forma atomizada e livre de deadlocks.

3.4.3. Programação Assíncrona

A biblioteca padrão de C++, a partir de C++11, sua primeira versão com suporte à concorrência, oferece mecanismos de programação assíncrona com suporte nativo à execução de

tarefas em segundo plano. O principal recurso para esse fim é a função `std::async`, que recebe um *objeto invocável* e retorna imediatamente um objeto `std::future`. Esse objeto retornado representa o resultado eventual da computação e pode ser usado posteriormente para sincronização e acesso ao valor retornado. Um exemplo da criação assíncrona de threads, e uso de variáveis futuras, é apresentado no Código 2.

A abordagem baseada em `std::async` abstrai o controle explícito de threads, permitindo que a lógica da aplicação foque na decomposição do problema em tarefas independentes, sem lidar diretamente com criação, junção ou gerenciamento de threads. A execução da função fornecida pode ocorrer de forma imediata ou adiada, dependendo da política de lançamento especificada.

A política `std::launch::async` força o início imediato da tarefa em uma thread separada, enquanto `std::launch::deferred` adia sua execução até que o resultado seja requisitado, por meio de `future::get`. Quando nenhuma política é especificada, a decisão fica a critério da implementação. A chamada a `get()` bloqueia a thread chamadora até que o resultado esteja disponível, realizando, portanto, uma sincronização implícita.

```
int resposta() {
    return 42;
}
int main() {
    std::future<int> f = std::async(std::launch::async, resposta);
    // ...
    int resultado = f.get(); // bloqueia até computacao() terminar
    std::cout << "Resultado: " << resultado << "\n";
    return 0;
}
```

Código 2: Criação assíncrona de threads em C++

Essa forma de programação é útil para tarefas computacionalmente intensivas ou operações de I/O que podem ser realizadas em paralelo com a lógica principal. No entanto, o uso de `std::async` deve ser feito com atenção ao modelo de execução adotado pela implementação, especialmente em relação ao número de threads disponíveis e à política de escalonamento do sistema subjacente. O padrão C++ não especifica se tarefas iniciadas com `std::async` usam threads dedicados ou um thread pool. O padrão define que com `std::launch::async`, a execução deve ser assíncrona, mas a implementação é livre para criar uma nova thread ou reutilizar uma de um pool. Já com `std::launch::deferred`, a execução ocorre no próprio thread chamador quando `get()` ou `wait()` é invocado. Na prática, as implementações variam: `libstdc++` (GCC) e `libc++` (Clang) criam uma nova thread para cada tarefa iniciada com `std::launch::async`, sem utilizar thread pools. Em contraste, o MSVC (Visual

Studio) adota um thread pool interno, o que reduz o overhead associado à criação de threads e permite um gerenciamento mais eficiente das tarefas. **REFERENCIA**

3.4.4. Algoritmos Paralelos

A partir do C++17, a STL passou a contar com suporte à execução paralela de seus algoritmos por meio de políticas que indicam se a operação deve ocorrer de forma sequencial, paralela ou paralela com vetorização. Ainda que essa extensão promova o uso de paralelismo de forma declarativa, ela exige o programador da necessidade de gerenciar diretamente threads ou empregar mecanismos explícitos de sincronização.

A principal interface para esse modelo é fornecida pelas versões sobrecarregadas dos algoritmos padrão da STL, que aceitam uma política de execução como primeiro argumento: `std::execution::seq` e `std::execution::par`. A política `seq` indica que o algoritmo deve ser executado de forma sequencial, sendo equivalente, portanto, ao comportamento anterior ao C++17. A política `par` habilita execução paralela das operações, com cada invocação podendo ocorrer em threads distintos. Existe também a política `std::execution::par_unseq`. Esta política combina paralelismo oferecido com a política `par` com a exploração do potencial de execução vetorial do processador, permitindo maior grau de exploração do paralelismo de dados. O trecho a seguir exemplifica os algoritmos paralelos, aplicando uma função lambda para atualizar cada elemento de um vetor com o valor de seu quadrado:

```
std::for_each(std::execution::par, v.begin(), v.end(), [](int& x) {
    x *= x;
});
```

O uso de esqueletos para descrição da concorrência permite delegar ao compilador e à biblioteca padrão a escolha das estratégias de mapeamento da concorrência gerada pelo programa no paralelismo disponibilizado pelo hardware, melhorando a expressividade do código e facilitando a manutenção. Deve ser observado mais uma vez que o padrão não especifica como os threads devem ser criados ou gerenciados para viabilizar a execução paralela dos algoritmos. Cabe à implementação da STL decidir se utilizará criação direta de threads, um banco de threads reutilizáveis (thread pool) ou se delegará essa responsabilidade a uma biblioteca paralela subjacente, como Intel TBB ou Microsoft PPL. Na prática, a maioria das implementações modernas opta pelo uso de thread pools, tanto por eficiência quanto por escalabilidade, evitando o custo associado à criação e destruição frequente de threads. Na `stdlibc++`, a estratégia adotada é de thread pool, sendo que o número de threads no pool é igual ao número de *cores* (virtuais) disponíveis.

Embora a abordagem dos algoritmos paralelos da STL seja de aplicação simples, deve ser observado que não é garantida a ordem de execução entre as iterações. Isso inviabiliza o uso de lógicas cuja execução de diferentes passos da iteração devam respeitar uma sequência específica. Por esse motivo, funções puramente funcionais ou que operem de forma independente sobre cada elemento da faixa são as mais indicadas nesse contexto.

3.4.5. Barreiras e Travamentos

A versão C++20, além de promover aprimoramentos incrementais na interface de `std::atomic`, introduziu novos mecanismos de sincronização, como as barreiras (`std::barrier`) e os travamentos (`std::latch`). Essas novas ferramentas visam oferecer um controle mais preciso sobre o fluxo de execução de threads em cenários de paralelismo.

A `std::barrier` permite que um conjunto de threads se sincronize em um ponto específico de execução antes de continuar, garantindo que todas as threads atinjam um marco determinado antes de prosseguir. Esse mecanismo é particularmente útil em situações que envolvem fases de processamento que precisam ser concluídas de forma conjunta por todas as threads, como em pipelines de dados ou etapas de computação paralela.

Já o `std::latch` oferece uma forma de sincronizar threads até que um número pré-determinado de threads tenha alcançado uma etapa específica de execução. A diferença fundamental entre `latch` e `barrier` é que, uma vez que o valor de contagem do `latch` é alcançado, ele não pode ser reutilizado, tornando-o adequado para cenários em que a sincronização precisa ocorrer apenas uma vez. Isso é útil em casos onde se deseja garantir que um conjunto de threads complete uma tarefa antes de permitir que o programa continue, sem necessidade de reuso da mesma barreira.

3.4.6. Mais sobre sincronização

O C++ oferece ainda outros mecanismos de sincronização além do tradicionais `std::mutex` e das de `std::atomic`. Um destes recursos é o `std::shared_mutex`, que oferece uma semântica de leituras concorrentes e escrita exclusiva a dados compartilhados. Seu uso permite que múltiplas threads adquiram acesso de leitura simultaneamente, enquanto garante exclusividade para acessos de escrita. Ele é útil em situações onde operações de leitura são muito mais frequentes do que as de escrita. Para usá-lo, empregam-se as classes `std::shared_lock` (para leitura) e `std::unique_lock` (para escrita). Um exemplo simples de uso:

```
std::shared_mutex smutex;
void leitor() {
    std::shared_lock lock(smutex);
    ... // acesso somente leitura
}
void escritor() {
    std::unique_lock lock(smutex);
    ... // acesso de escrita
}
```

Outro recurso que se apresenta como uma alternativa ao uso individualizado das operações `lock()` sobre `mutex` é a classe `std::scoped_lock`, que permite travar

múltiplos mutexes simultaneamente em uma única chamada, evitando deadlocks por aquisição desordenada. Ele é particularmente útil quando diversas seções críticas devem ser protegidas de forma coordenada:

```
std::mutex a, b;
void funcao() {
    std::scoped_lock lock(a, b);
    ... // código protegido
}
```

Outro mecanismo importante para coordenação entre threads é a `std::condition_variable`, que permite a uma thread esperar por uma determinada condição antes de prosseguir. Ela é normalmente usada em conjunto com `std::unique_lock` e `std::mutex` e permite que threads sejam notificados por meio das funções `notify_one()` ou `notify_all()`. Essa abordagem é útil, por exemplo, para implementar filas de trabalho, buffers compartilhados ou qualquer estrutura que dependa de condições de disponibilidade de dados ou recursos. A verificação da condição deve sempre ser feita dentro de um laço, pois sinais espúrios podem ocorrer. O exemplo a seguir ilustra uma situação simples em que um thread produtor acorda uma thread consumidor após alterar uma variável de controle:

```
std::mutex m;
std::condition_variable cv;
bool pronto = false;
void produtor() {
    std::this_thread::sleep_for(std::chrono::seconds(1));
    {
        std::unique_lock lock(m);
        pronto = true;
    }
    cv.notify_one();
}
void consumidor() {
    std::unique_lock lock(m);
    cv.wait(lock, [] { return pronto; });
    std::cout << "Consumidor acordado.\n";
}
```

Neste exemplo, a variável `pronto` controla quando o consumidor pode prosseguir. A `std::condition_variable` depende de um `std::mutex` para coordenar o acesso à variável compartilhada e garantir consistência. O método `wait()` libera temporariamente o mutex enquanto a thread está bloqueada, e o `readquire` automaticamente quando a notificação ocorre e a condição é satisfeita.

```

void tarefa(std::stop_token st, int i) {
    while (!st.stop_requested()) {
        std::cout << "Trabalhando...\n";
        std::this_thread::sleep_for(std::chrono::milliseconds(500));
    }
    std::cout << "Thread " << i << " cancelada com segurança.\n";
}

int main() {
    std::jthread t1(tarefa,1);
    std::jthread t2(tarefa,2);
    std::this_thread::sleep_for(std::chrono::seconds(2));
    t1.request_stop();
    t2.request_stop();
    return 0;
}

```

Código 3: Cancelamento cooperativo seguro.

Por fim, o C++ disponibiliza a `std::condition_variable_any`, uma versão mais flexível da `std::condition_variable`. Enquanto esta última funciona exclusivamente com `std::unique_lock` e `std::mutex`, a variante `any` pode ser usada com qualquer tipo de objeto de sincronização que implemente os métodos `lock()` e `unlock()`, como `std::shared_mutex` ou mutexes personalizados. Isso permite, por exemplo, empregar variáveis de condição em cenários que envolvam acesso concorrente para leitura, algo inviável com a versão tradicional. Internamente, a semântica de uso é semelhante, envolvendo uma verificação de condição dentro de um laço e a liberação automática do bloqueio durante o tempo de espera. No entanto, o programador deve garantir que o tipo de trava utilizada ofereça o comportamento correto de exclusão mútua ou compartilhada, conforme o caso.

3.4.7. Cancelamento Cooperativo

O padrão C++20 introduziu a classe `std::jthread`, que representa uma evolução segura da tradicional `std::thread`. A principal vantagem deste recurso é que sua estrutura interna gerencia automaticamente a finalização da thread, realizando uma chamada a `join()` em seu destrutor, eliminando a necessidade de fazê-lo manualmente. Essa implementação evita encerramentos abruptos por destruição de threads ainda ativos, reduzindo o risco de erros comuns associados à gestão manual do seu ciclo de vidas.

`std::jthread` introduz ainda suporte embutido ao cancelamento cooperativo, por meio de travas de contagem implementadas por mecanismos `std::stop_token` e `std::stop_source`. O cancelamento é iniciado pelo emissor de um `stop_source`, e as threads associadas recebem um `stop_token` que pode ser consultado periodicamente para verificar se a execução deve ser encerrada.

O Código 3 ilustra o cancelamento coletivo com uso de travas de contagem. No exemplo, cada thread recebe um `std::stop_token` como primeiro argumento da função executada. Esse token permite à função detectar, de forma não intrusiva, se foi solicitado seu encerramento. No corpo da função `tarefa`, a verificação é feita com `st.stop_requested()`, que retorna `true` quando o pedido de cancelamento é emitido. A verificação ocorre dentro de um laço, permitindo que a thread finalize sua execução de maneira controlada assim que o sinal de parada for detectado.

A chamada `request_stop()` realizada no `main` aciona o mecanismo de cancelamento, comunicando a intenção de finalização aos threads. Como o cancelamento é cooperativo, cabe à função em execução respeitar o pedido e interromper seu fluxo. A estrutura do `std::jthread` garante, por meio de seu destrutor, que o método `join()` seja invocado automaticamente, assegurando que o programa principal espere o término das threads antes de prosseguir.

3.4.8. RAI e Gerenciamento de Recursos

Em alguns pontos desta seção foi mencionado o termo RAI (Resource Acquisition Is Initialization). RAI é um padrão de design em C++ que vincula a alocação e liberação de recursos ao ciclo de vida de objetos. Quando um objeto RAI é criado, ele adquire um recurso no seu construtor, como memória, arquivos ou mecanismos de sincronização e mesmo a criação de um thread. Ao sair de escopo, o destrutor do objeto é chamado, liberando o recurso de forma automática. Esse comportamento elimina a necessidade de gerenciamento manual de recursos, minimizando o risco de erros como vazamentos de memória ou falhas na liberação de recursos, especialmente em cenários com exceções ou retornos antecipados de funções.

Na programação multithread em C++, o RAI é amplamente utilizado para garantir o gerenciamento correto de threads e sincronização de dados. A classe `std::thread` segue o padrão RAI, pois o thread é iniciado na construção do objeto e precisa ser explicitamente finalizado com `join()` ou `detach()` antes da destruição do objeto. Se o thread não for finalizado corretamente, o programa chamará `std::terminate()`, forçando o término do programa. O uso de mutexes também é gerido por meio de RAI, como demonstrado pelo `std::lock_guard`, que adquire um mutex no momento da criação do objeto e o libera automaticamente quando o objeto sai de escopo. Além desses casos, o RAI pode ser empregado no gerenciamento de recursos temporários em operações assíncronas, onde objetos como `std::unique_lock` e `std::shared_lock` permitem maior flexibilidade na sincronização de múltiplos threads, assegurando a correta liberação de recursos mesmo em cenários complexos de concorrência.

3.5. Rust

Rust² [Team 2021][Troutwine 2018] teve sua primeira versão estável em 2015. Desde então tem se destacado como uma linguagem moderna voltada ao desenvolvimento de sistemas seguros e eficientes. A linguagem combina controle de baixo nível com garantias de segurança em tempo de compilação, utilizando o sistema de `ownership` e as regras de empréstimo para evitar condições de corrida e acessos inválidos a dados. As abstra-

²<https://www.rust-lang.org>

ções oferecidas por sua biblioteca padrão incluem criação e gerenciamento de threads, estruturas de sincronização, como mutexes e tipos atômicos, além de canais para troca segura de mensagens entre threads. Para aplicações que demandam alta escalabilidade, Rust também oferece suporte a programação assíncrona baseada em futures, com sintaxe `async/await` e integração com runtimes como Tokio. O ecossistema da linguagem é fortemente apoiado pelo Cargo, uma ferramenta que gerencia pacotes, dependências e builds de forma integrada, facilitando a adoção de bibliotecas especializadas para concorrência e paralelismo.

3.5.1. Modelo Básico

Rust oferece um modelo de concorrência construído a partir de três mecanismos principais: threads nativos, tarefas assíncronas e canais de comunicação. Cada um desses pilares é integrado à linguagem de forma a respeitar o sistema de ownership e garantir segurança em tempo de compilação.

3.5.1.1. Threads Nativos

A criação de um thread nativo ocorre com a chamada à função `thread::spawn`, que requer como parâmetro um fechamento (closure). O thread é criado para executar esta closure. A captura de variáveis é explicitada pelo uso da palavra-chave `move`, resultando na transferência da posse dos dados instanciados no escopo envolvente ao lançamento do thread para o escopo do novo thread. Este mecanismo de propriedade inviabiliza o compartilhamento de dados, impedindo assim referências inválidas e compartilhamento inseguro. Observe que apenas as variáveis efetivamente utilizadas no closure serão movidos para o novo thread, os demais dados do escopo envolvente não são afetados. O Código 4 mostra exemplos básicos de criação de threads com closures, incluindo o uso de `join` para aguardar sua conclusão.

```
use std::thread;
fn main() {
    let x = 8;
    let handle = thread::spawn(move || x * x);
    let r = handle.join().unwrap();
    println!("{}", r);
}
```

Código 4: Modelo básico de criação de threads em Rust

O objeto retornado por `thread::spawn` representa a handle do thread criado e permite sincronização explícita com a chamada ao método `join`. Os threads são, assim, identificados individualmente. Caso `join` não seja chamado, o thread ainda será executado normalmente, mas não há garantia de que será concluído antes do término da função

principal, podendo levar à finalização prematura do programa. Assim, todo thread criado deverá ser submetido a uma, e no máximo a uma, operação de sincronização por `join`, não sendo possível ativar um mecanismo de `detach`, como em C++. No exemplo, o thread retorna um valor, e a chamada a `join()` tem como resultado um `Result<T>`, onde `T` é o tipo do valor retornado pela closure do thread. O método `.unwrap()` é então chamado nesse `Result`. O `unwrap()` é uma forma de lidar com o possível erro que pode ocorrer durante a execução da thread. Se o thread completar sua execução com sucesso e retornar um valor, o `unwrap()` extrairá esse valor do `Result`. Por outro lado, se o thread entrar em pânico (devido a um erro irrecuperável), a chamada a `unwrap()` fará com que o thread principal também entre em pânico e o programa seja finalizado.

3.5.1.2. Programação Assíncrona

O suporte à programação assíncrona é realizado por meio dos construtores `async` e `await`. Esse suporte é implementado por meio da transformação automática de funções assíncronas em estruturas que implementam a *trait* `Future`. No entanto, a biblioteca padrão não fornece um mecanismo de escalonamento para essas *futures* . As opções recaem em desenvolver esta lógica de agendamento no próprio programa ou usar bibliotecas, ou *crates* , na terminologia Rust, consolidadas que proveem estes serviço. Dentre as opções, Tokio é bastante popular.

No modelo oferecido pelo Tokio, cada tarefa assíncrona é representada por um valor que implementa a *trait* , algo como interface no contexto Rust, `Future`. Esse valor encapsula o estado da computação e pode ser agendado em um runtime baseado em múltiplos threads, que executa as tarefas de forma cooperativa. O programador declara uma tarefa com a palavra-chave `async`, e seu corpo só é executado quando o `Future` resultante for submetido ao runtime por meio do método `spawn` ou `spawn_blocking`.

A macro `tokio::main` transforma a função principal do programa em uma tarefa assíncrona e inicializa automaticamente o runtime. A seguir, o Código 5 ilustra a criação de tarefas concorrentes com Tokio.

```
use tokio::task;
#[tokio::main]
async fn main() {
    let t = task::spawn(async { println!("Executando uma tarefa"); });
    t.await.unwrap();
}
```

Código 5: Criação de tarefas assíncronas com Tokio

A função `task::spawn` inicia imediatamente a execução da tarefa assíncrona no pool de threads mantido pelo runtime. O método `await` sobre o valor retornado permite aguardar a conclusão da tarefa, de forma semelhante à chamada `join` usada

com threads do sistema operacional. No entanto, ao contrário dos threads nativos, o uso de `await` não bloqueia a thread atual, mas apenas suspende a tarefa em execução, permitindo o avanço de outras tarefas cooperativas no mesmo runtime.

No código exemplo, também é observado que a função `main` também é assíncrona. Isso é necessário, por, em Tokio, somente uma tarefa assíncrona pode invocar `await` sobre outra tarefa. A macro `#[tokio::main]` providencia a execução assíncrona da função `main`.

3.5.1.3. Canais de Comunicação

Embora Rust ofereça diferentes mecanismos para compartilhamento de dados entre threads e tarefas assíncronas, o principal meio para troca de informações entre essas entidades é por meio de canais. O modelo de canais adotado segue o padrão `mpsc` (multiple producer, single consumer), no qual múltiplos produtores podem enviar dados para um único consumidor. Esse modelo minimiza a necessidade de acesso compartilhado à memória, garantindo que as mensagens sejam entregues de forma segura, sem risco de condições de corrida.

A criação de um canal é realizada por meio da função `mpsc::channel`, que retorna dois objetos: o transmissor (`Sender`) e o receptor (`Receiver`). O transmissor envia as mensagens, enquanto o receptor as recebe. O receptor bloqueia até que uma mensagem esteja disponível, o que torna o modelo simples e eficiente para comunicação síncrona. Um exemplo desse mecanismo é apresentado no Código 6, no qual o thread principal cria um canal e um novo thread. Este último envia uma mensagem para o thread principal, que a recebe usando a função `recv()`, que bloqueia até a chegada da mensagem.

```
use std::sync::mpsc;
use std::thread;
fn main() {
    let (tx, rx) = mpsc::channel();
    let t = thread::spawn(move || {
        let mensagem = String::from("Olá do thread!");
        tx.send(mensagem).unwrap();
    });
    let recebido = rx.recv().unwrap(); // recebe a mensagem
    println!("Mensagem recebida: {}", recebido);
    t.join().unwrap();
}
```

Código 6: Canais de Comunicação

Os canais de comunicação são thread-safe, ou seja, podem ser usados por múlti-

plos threads ou tarefas assíncronas sem causar problemas de concorrência. Além disso, o transmissor (`Sender`) pode ser clonado, permitindo que várias threads ou tarefas assíncronas enviem mensagens para o mesmo receptor.

3.5.2. Compartilhamento Seguro de Dados

Em Rust, o compartilhamento seguro de dados entre threads e tarefas assíncronas é um princípio central garantido pela linguagem por meio de diversas abstrações de sincronização e controle de acesso. Essas ferramentas permitem que dados sejam compartilhados entre threads de forma controlada, evitando condições de corrida e garantindo a integridade dos dados. As principais ferramentas para garantir o compartilhamento seguro de dados incluem `Mutex<T>`, `RwLock<T>` e `Arc<T>`. Os dois primeiros são *wrappers* que envolvem um dado a ser compartilhado. O `Mutex<T>` garante acesso exclusivo, tanto para leitura quanto para escrita, enquanto o `RwLock<T>` permite leituras concorrentes, mas restringe a escrita a uma thread por vez, garantindo exclusividade nesse caso.

O *wrapper* `Arc<T>` (atomic reference counting) é utilizado para permitir o compartilhamento de um `Mutex<T>` ou `RwLock<T>` entre múltiplas threads, gerenciando de forma segura a contagem de referências e evitando problemas como referências inválidas ou uso incorreto de memória.

```
use std::sync::{Arc, Mutex}; use std::thread;
fn main() {
    let counter = Arc::new(Mutex::new(0));
    let handles = vec![];
    for _ in 0..10 {
        let counter = Arc::clone(&counter);
        let handle = thread::spawn(move || {
            let mut num = counter.lock().unwrap();
            *num += 1;
        });
        handles.push(handle);
    }
    for handle in handles {
        handle.join().unwrap();
    }
    println!("Contador: {}", *counter.lock().unwrap());
}
```

Código 7: Exemplo de uso de Mutex em Rust

Além dessas abstrações, Rust também disponibiliza tipos atômicos como `AtomicBool`, `AtomicUsize` e outros tipos numéricos, que permitem a execução de operações seguras de leitura-modificação-gravação em dados compartilhados. Essas operações atômicas são realizadas sem a necessidade de bloqueios explícitos, o que pode

melhorar o desempenho em casos onde a sobrecarga de sincronização seria um problema.

3.5.3. Ferramentas e Padrões Avançados

Para o desenvolvimento de programas multithread em Rust o programador também conta com estruturas de sincronização do tipo barreiras (`Barrier`), semáforos e variáveis de condição (`Condvar`). A exemplo dos recursos de exclusão mútua apresentados anteriormente, também devem ser encapsulados em *wrappers* de proteção ao acesso.

Caso o enfoque a ser dado seja na exploração de paralelismo de dados, a linguagem conta com o suporte de `rayon`. O uso desta *crate* é particularmente interessante por integrar-se de forma transparente com iteradores, oferecendo um modelo funcional de paralelização com mínima complexidade adicional. Ainda se referindo à disponibilidade de modelos de programação de mais alto nível, também é suportada a implementação de sistemas baseados no modelo de atores por *crates* específicas, como a `actix`. Por meio deste recurso, *atores*, as entidades concorrentes e isoladas de execução, comunicam-se exclusivamente por meio de mensagens assíncronas.

3.6. Go

A linguagem de programação Go³ [Cox-Buday 2017] foi criada, no final da década de 2000, com o objetivo de oferecer uma linguagem eficiente e com bom suporte à concorrência. O modelo de concorrência desta linguagem é centrado no uso de goroutines, unidades leves de programação concorrente que são multiplexadas de forma eficiente sobre um conjunto de threads do sistema operacional. A criação e gerenciamento de goroutines não exige o controle direto dos threads por parte do programador. A comunicação entre goroutines é feita por meio de canais, que permitem troca segura de dados, reduzindo a necessidade de mecanismos de sincronização baseados em seções críticas. Esse modelo, fortemente influenciado pela notação de processos de CSP (Communicating Sequential Processes), promove um estilo de programação concorrente mais estruturado e seguro. Go também fornece bibliotecas padronizadas para controle de sincronização, timers, pools de goroutines e estruturas reativas, ampliando o suporte à construção de sistemas concorrentes robustos e escaláveis.

3.6.1. Modelo Básico

O suporte à concorrência em Go foi concebido como parte central da linguagem, oferecendo um modelo estruturado que favorece a decomposição de programas em unidades independentes que se comunicam por meio de canais. Essa organização estimula o pensamento em termos de atividades que cooperam entre si, em vez de manipulação direta de estados compartilhados. A comunicação por canais, portanto, é o mecanismo utilizado para troca de dados entre fluxos concorrentes. A contrapartida é que esse modelo estruturado reduz o número de alternativas para a implementação de algoritmos às suportadas pela própria estrutura oferecida.

A unidade de programação concorrente em Go é a goroutine, que consiste em uma tarefa gerenciada integralmente pelo ambiente de execução da linguagem, o qual é composto por threads do sistema operacional. A comunicação e a coordenação entre essas

³<https://go.dev>

tarefas ocorrem por meio de canais, que funcionam simultaneamente como meio de troca de dados e como mecanismo explícito de sincronização entre goroutines.

3.6.1.1. Goroutines: Criação e Gerenciamento

Goroutines são a principal abstração de concorrência em Go e representam tarefas que podem ser executadas de forma independente. Elas são significativamente mais leves do que os threads do sistema operacional, tanto em termos de custo de criação quanto de consumo de memória.

Para lançar uma tarefa concorrente em Go, basta prefixar a chamada de uma função com a palavra-chave `go`. Isso instrui o ambiente de execução da linguagem a agendar a execução da função como uma nova tarefa concorrente, com retorno imediato após o lançamento da nova tarefa. A função chamada pode ser uma função nomeada, uma lambda ou até um método de uma estrutura. O Código 8 ilustra a criação de duas goroutines a partir da função `main`.

```
package main
import (
    "fmt"
    "sync"
)
func f(nome string, wg *sync.WaitGroup) {
    defer wg.Done()
    fmt.Println("Executando:", nome)
}
func main() {
    var wg sync.WaitGroup

    wg.Add(2) // 2 goroutines
    go f("goroutine 1", &wg)
    go func() {
        defer wg.Done()
        fmt.Println("Executando: goroutine 2")
    }()
    // ...
    wg.Wait()
    fmt.Println("Função main retornou.")
}
```

Código 8: Modelo básico de criação de goroutines em Go

O ambiente de execução de Go gerencia o agendamento e a distribuição das gorou-

tines sobre um conjunto de threads do sistema operacional, por meio de um escalonador cooperativo de múltiplas filas, mantido pelo runtime de execução da linguagem. Esse escalonador permite que as goroutines sejam mapeadas dinamicamente sobre os threads nativos, gerenciados pelo sistema operacional, conforme necessário, promovendo boa utilização dos *cores* disponíveis e evitando sobrecarga com alternância de contexto excessiva. Como o escalonamento é interno à linguagem, o programador não precisa gerenciar diretamente a criação ou finalização de threads.

Por padrão, os programas são encerrados assim que a função `main` retorna, mesmo que existam goroutines em execução. Para evitar essa situação, devem ser utilizados mecanismos de sincronização, como `sync.WaitGroup`, para aguardar a conclusão de goroutines cujo processamento ainda não tenha sido finalizado.

3.6.1.2. Barreiras e Seções Críticas

Embora canais sejam a forma nativa para compartilhamento de informações e sincronização entre goroutines, existem situações em que o controle direto sobre o acesso a dados compartilhados é necessário. Nestes casos, o uso de mecanismos de exclusão mútua, como `sync.Mutex` e `sync.RWMutex`, pode ser mais apropriado, oferecendo uma maneira explícita de coordenar o acesso a seções críticas de código.

O `sync.Mutex` é o tipo mais básico de bloqueio e garante que apenas uma goroutine tenha acesso a uma seção crítica de cada vez. Quando uma goroutine adquire o mutex, outras goroutines que tentarem adquirir o mesmo mutex ficarão bloqueadas até que ele seja liberado. Por outro lado, o `sync.RWMutex` oferece um mecanismo mais sofisticado, permitindo leitura simultânea por várias goroutines, desde que nenhuma goroutine tenha adquirido um bloqueio exclusivo para escrita. Isso é útil quando há um alto volume de leituras e um baixo volume de modificações, proporcionando uma melhor performance em cenários com acesso concorrente a dados.

Além desses mecanismos de exclusão mútua, Go também oferece a estrutura `sync.WaitGroup`, que é um mecanismo de sincronização utilizado para aguardar a conclusão de um conjunto de goroutines. O `sync.WaitGroup` atua como uma barreira, permitindo que uma goroutine principal ou outra goroutine aguarde a conclusão de múltiplas tarefas concorrentes antes de prosseguir com a execução. Essa ferramenta é particularmente útil em cenários onde o sincronismo entre a execução de múltiplas goroutines é necessário, sem depender diretamente da comunicação por canais.

Embora os canais sejam frequentemente utilizados para comunicação, os mecanismos de exclusão mútua fornecem uma solução eficiente para coordenar o acesso a dados compartilhados em memória, especialmente quando a comunicação por canais não é suficiente ou desejada.

3.6.2. Comunicação Entre Goroutines

A comunicação entre goroutines em Go é feita de forma segura e eficiente usando canais. Go oferece um modelo de passagem de mensagens por canais que elimina a necessidade de bloqueios explícitos para compartilhar dados entre goroutines.

3.6.3. Comunicação Concorrente com Canais

Canais são o principal mecanismo de comunicação entre goroutines em Go. Eles permitem a troca de dados de forma segura entre tarefas concorrentes, promovendo tanto a transmissão de valores quanto a sincronização implícita entre os participantes da comunicação. Por meio dos canais, é possível construir programas concorrentes que evitam o uso de variáveis compartilhadas e os problemas típicos associados à sua manipulação.

Um canal é criado com a função `make`, especificando o tipo dos dados que serão transmitidos. As operações de envio e recebimento são realizadas com o operador `←`. Ao enviar um valor para um canal, a goroutine emissora fica bloqueada até que outra goroutine esteja pronta para recebê-lo, a menos que a capacidade de bufferização do canal não esteja esgotada. Esse comportamento sincronizado permite que os canais sirvam como ponto de coordenação entre diferentes tarefas. Em casos em que o remetente não mais enviará valores, a função `close` permite sinalizar o encerramento do canal. Essa prática é especialmente útil quando o receptor está em um laço de leitura, evitando bloqueios ou a necessidade de lógica adicional para encerrar a leitura. Tais laços de leitura são implementados pelo construtor `for-range`, permitindo iterar sobre todas as mensagens recebidas e encerrando ao seu término.

```
for val := range ch {  
    fmt.Println(val)
```

Além dos canais bidirecionais, é possível restringir a direção de uso de um canal, criando canais unidirecionais. Essa especialização pode ser útil para modularizar o código e reforçar garantias de segurança no uso da comunicação, indicando de maneira explícita se determinada função apenas envia ou apenas recebe dados por um canal.

Para lidar com múltiplos canais simultaneamente, Go oferece a construção `select`, que permite aguardar operações de envio ou recebimento em diferentes canais. O `select` avalia de forma não determinística os casos que estiverem prontos para prosseguir, o que permite, por exemplo, implementar mecanismos de multiplexação, onde diferentes fontes de dados podem ser atendidas de maneira reativa. Esse recurso também permite a implementação de timeouts e cancelamento de operações concorrentes, quando combinado com canais auxiliares ou temporizadores, como `time.After`.

Canais podem ser definidos como com ou sem buffer. Um canal sem buffer exige que o remetente e o receptor estejam prontos simultaneamente para que a operação de envio ou recebimento seja completada. Já um canal com buffer permite que um número limitado de valores seja enviado mesmo que nenhuma goroutine esteja aguardando para recebê-los imediatamente. Isso introduz uma forma de desacoplamento entre remetente e receptor, útil em situações onde a produção e o consumo de dados ocorrem em ritmos distintos.

O uso de canais com buffer pode melhorar a performance de sistemas concorrentes, desde que o tamanho do buffer seja escolhido cuidadosamente, de acordo com as características da aplicação. Um buffer muito pequeno pode levar a bloqueios frequentes, enquanto um buffer muito grande pode introduzir latência e dificultar o controle de fluxo.

3.6.4. Sincronização e Exclusão Mútua

Embora Go ofereça mecanismos de comunicação entre goroutines por meio de canais, há situações em que é necessário controlar diretamente o acesso a recursos compartilhados. Para esses casos, a linguagem fornece estruturas de sincronização no pacote `sync`, como `Mutex` e `RWMutex`.

O tipo `sync.Mutex` implementa exclusão mútua. Ele permite que apenas uma goroutine acesse uma seção crítica por vez, impedindo condições de corrida quando múltiplas goroutines tentam modificar o mesmo recurso.

Para usá-lo, declara-se uma variável do tipo `sync.Mutex` e envolve-se o acesso ao dado com chamadas aos métodos `Lock()` e `Unlock()`. Exemplo no Código 9.

```
var m sync.Mutex var contador int
func incrementar() {
    m.Lock()
    contador++
    m.Unlock()
}
```

Código 9: Modelo básico de criação de goroutines em Go

É fundamental garantir que o método `Unlock()` seja sempre chamado, mesmo em casos de erro. Uma alternativa é, imediatamente após a obtenção do mutex, automatizar a liberação do mutex ao final da execução da função na qual o mutex foi adquirido, invocando sobre o objeto mutex a operação `defer`. Em termos práticos, a invocação da operação `defer` instrui o compilador a introduzir um código ao final da execução da função que executa o desbloqueio, independente se este término ocorreu normalmente ou por uma operação de pânico.

O tipo `sync.RWMutex` é uma variação do `Mutex` que permite múltiplas leituras simultâneas, desde que nenhuma escrita esteja em andamento. Ele é útil quando há predominância de operações de leitura sobre um mesmo recurso compartilhado.

O método `RLock()` bloqueia o mutex para leitura, enquanto `Lock()` bloqueia para escrita. A liberação deve ser feita com `RUnlock()` ou `Unlock()`, respectivamente.

3.6.5. Ferramentas e Bibliotecas Avançadas

Além das primitivas básicas de sincronização e comunicação, a linguagem Go oferece ferramentas adicionais que auxiliam na construção de sistemas concorrentes mais robustos e eficientes. Dentre elas, destaca-se o pacote `context`, usado para controlar o ciclo de vida de goroutines. Com ele, é possível propagar sinais de cancelamento e configurar limites de tempo para operações, o que facilita o gerenciamento cooperativo de tarefas, especialmente em aplicações com chamadas encadeadas e alto grau de concorrência. O

cancelamento explícito por meio de `context.WithCancel` ou a imposição de prazos com `context.WithTimeout` permitem evitar vazamentos de goroutines e garantir encerramento limpo de atividades.

Outro recurso útil é a estrutura `sync.Pool`, que implementa um pool de objetos reutilizáveis. Seu uso é recomendado em cenários nos quais há criação intensiva de objetos temporários, como buffers ou estruturas auxiliares. Ao minimizar a pressão sobre o coletor de lixo, essa estrutura pode contribuir significativamente para a redução do custo de alocação dinâmica e melhorar a performance de aplicações paralelas. No entanto, o conteúdo de um `sync.Pool` pode ser descartado a qualquer momento, e seu uso deve ser reservado a situações nas quais a perda de elementos armazenados não compromete a lógica da aplicação.

3.7. Elixir

A linguagem Elixir⁴ [Marx et al. 2018] foi projetada para a construção de sistemas concorrentes, distribuídos e tolerantes a falhas. Baseada na máquina virtual Erlang (BEAM), herda um modelo de concorrência centrado em processos leves, isolamento completo e comunicação por troca de mensagens. Em contraste com abordagens baseadas em threads e memória compartilhada, Elixir adota o modelo de atores, no qual cada processo atua como uma unidade independente, com estado próprio e interação exclusivamente assíncrona. Essa arquitetura favorece a escalabilidade e a robustez, características centrais em aplicações como servidores web, sistemas de telecomunicações e pipelines de dados em tempo real. Nesta seção, analisamos os principais recursos oferecidos por Elixir para programação concorrente, começando pela criação e comunicação entre processos, passando por mecanismos de recepção e tratamento de falhas, até abstrações de mais alto nível disponíveis na linguagem e em seu ecossistema.

3.7.1. Modelo Básico

O modelo de concorrência adotado por Elixir baseia-se na criação massiva de processos leves, também chamados de processos Erlang. Esses processos são completamente isolados entre si, não compartilham memória e se comunicam exclusivamente por meio de troca de mensagens. Essa abordagem deriva diretamente da máquina virtual BEAM, desenvolvida para suportar sistemas distribuídos e tolerantes a falhas.

A criação de processos em Elixir é feita com a função `spawn`, que recebe como argumento uma função a ser executada de forma concorrente. Cada processo criado dessa forma possui seu próprio espaço de execução e pode ser identificado por um identificador do tipo `pid` (process identifier). Como esses processos são extremamente leves, ocupando apenas poucos kilobytes de memória, é comum a criação de milhares ou até milhões de processos em aplicações reais.

A seguir, um exemplo simples de criação de processos:

```
defmodule Demo do
  def saudacao do
```

⁴<https://elixir-lang.org>

```
IO.puts("Olá do processo!")
end
end
pid = spawn(Demo, :saudacao, [])
```

No exemplo acima, o processo principal invoca `spawn/3`, passando o módulo, a função e os argumentos a serem usados. A função `saudacao` será executada concorrentemente, imprimindo uma mensagem. O valor retornado por `spawn` é o `pid` do processo criado, que pode ser usado para interação posterior.

Embora este exemplo não envolva comunicação entre processos, ele já ilustra o princípio fundamental da concorrência em Elixir: a criação de unidades isoladas e independentes de execução.

3.7.2. Envio e Recepção de Mensagens

A comunicação entre processos em Elixir é feita por meio de troca de mensagens, utilizando um modelo assíncrono. Cada processo possui uma caixa de correio própria (mailbox), que armazena as mensagens recebidas até que sejam explicitamente processadas. O envio é realizado com o operador `send`, enquanto a recepção é feita com a construção `receive`.

O envio de mensagens em Elixir pode ser feito utilizando a forma `pid <- mensagem`, uma construção da linguagem que equivale a `send(pid, mensagem)`. Essa operação permite que qualquer processo envie uma mensagem a outro, desde que conheça seu `pid`. As mensagens podem ser de qualquer tipo de dado Elixir, incluindo átomos, tuplas, listas ou estruturas mais complexas.

A estrutura `receive` permite que um processo filtre e trate mensagens conforme padrões definidos, como no exemplo abaixo:

```

defmodule Mensageiro do
  def iniciar do
    receive do
      {:ola, remetente} ->
        IO.puts("Recebido: olá de #{inspect(remetente)}")

      _mensagem ->
        IO.puts("Mensagem desconhecida: #{inspect(_mensagem)}")
    end
  end
end

pid = spawn(Mensageiro, :iniciar, [])
send(pid, {:ola, self()})

```

Código 10: Modelo básico de criação de processos em Elixir

Neste exemplo, o processo criado por `spawn` fica aguardando uma mensagem. Ao receber a tupla `:ola, remetente`, corresponde ao padrão e imprime uma saudação. Caso a mensagem não corresponda ao padrão esperado, uma resposta genérica é impressa. O uso de `self()` retorna o `pid` do processo atual (o remetente).

Por padrão, o `receive` bloqueia a execução até que uma mensagem seja recebida. É possível, no entanto, especificar um tempo limite com a construção `after`:

```

receive do
  :ping ->
    IO.puts("pong")
after
  1000 ->
    IO.puts("Nenhuma mensagem recebida em 1 segundo")
end

```

Esse controle de tempo evita que um processo fique indefinidamente bloqueado esperando por uma mensagem que talvez nunca chegue. Caso o tempo especificado (em milissegundos) expire sem o recebimento de mensagens, a cláusula `after` é executada. No exemplo, 1.000 ms.

A estrutura `receive`, além de permitir a recepção de mensagens, oferece recursos expressivos para tratar diferentes padrões de mensagens. Cada cláusula dentro do `receive` especifica um padrão e as ações a serem tomadas quando esse padrão for correspondente. O mecanismo de correspondência segue as regras do `pattern matching` da linguagem, permitindo o uso de tuplas, listas, átomos e até guardas lógicas com `when` para maior controle.

Além da filtragem, é comum que o código precise lidar com mensagens inesperadas. Uma prática recomendada é incluir uma cláusula coringa, como `_ -> . . .`, para capturar qualquer mensagem não tratada explicitamente, evitando que fiquem acumuladas na mailbox.

3.7.3. Monitoramento e Tolerância a Falhas

A concorrência em Elixir foi concebida com foco na construção de sistemas robustos e resilientes a falhas. Essa resiliência se apoia em dois pilares: a detecção de falhas entre processos e a capacidade de reinício automático, organizados sob a filosofia conhecida como “let it crash”. Essa abordagem parte do princípio de que erros são inevitáveis e que os processos devem falhar rapidamente, delegando a recuperação ao sistema de supervisão.

Elixir fornece primitivas para que processos observem uns aos outros. A função `Process.monitor/1` cria uma relação de monitoramento unidirecional: o processo monitorado não tem conhecimento da existência do monitor. Se o processo monitorado for encerrado, o monitor receberá uma mensagem `:DOWN, ref, :process, pid, razao`, informando o motivo do término. O monitoramento é indicado quando não se deseja que os dois processos compartilhem destino, ou seja, que a falha de um não cause a finalização do outro.

Por outro lado, a função `link/1` estabelece um vínculo bidirecional. Se um dos processos falhar de maneira não tratada, o outro será automaticamente finalizado, a menos que intercepte o sinal de saída com mecanismos como `Process.flag(:trap_exit, true)`. Esse tipo de ligação é útil quando processos dependem mutuamente para garantir a consistência de uma operação.

Com base nesses mecanismos, Elixir organiza processos em estruturas hierárquicas chamadas árvores de supervisão (supervision trees). Um supervisor é um processo especial que monitora outros processos filhos e aplica estratégias definidas de reinício em caso de falha. O módulo `Supervisor` permite especificar como e quando cada processo deve ser reiniciado, por exemplo: reinício individual (`:one_for_one`), reinício em cascata (`:one_for_all`) ou reinício proporcional (`:rest_for_one`).

Essa arquitetura permite que aplicações sejam estruturadas em componentes pequenos, isolados e recuperáveis, nos quais falhas locais não comprometem o funcionamento global. A tolerância a falhas em Elixir, portanto, não é um recurso adicional, mas um princípio de design central no desenvolvimento de sistemas concorrentes confiáveis.

3.7.4. Agentes, Tasks e GenServers

Além do modelo explícito baseado em `spawn`, envio de mensagens e `receive`, Elixir oferece abstrações de mais alto nível para encapsular lógica concorrente de forma estruturada. Entre elas, destacam-se os módulos `Agent`, `Task` e `GenServer`, todos construídos sobre os princípios fundamentais de processos isolados e troca de mensagens.

O módulo `Agent` é usado para encapsular e gerenciar estado de maneira simples. Ele fornece uma interface funcional para atualizar e consultar valores mantidos por um processo. Essa abordagem permite compartilhamento controlado de estado sem recor-

rer a variáveis globais ou memória compartilhada. Um agente é iniciado com a função `Agent.start_link`, que cria um processo que mantém o estado interno:

```
{:ok, pid} = Agent.start_link(fn -> 0 end)
Agent.update(pid, fn state -> state + 1 end)
Agent.get(pid, fn state -> state end)
```

O módulo `Task` fornece uma forma conveniente de executar funções assíncronas e aguardar seu resultado. É útil em situações onde se deseja disparar uma computação paralela de curta duração. A função `Task.async` cria um processo e retorna uma referência, enquanto `Task.await` bloqueia até a resposta estar disponível:

```
task = Task.async(fn -> realizar_calculo() end)
resultado = Task.await(task)
```

Por fim, o módulo `GenServer` (Generic Server) é uma abstração poderosa para construir servidores de processos com ciclo de vida bem definido, estado interno persistente e suporte integrado a chamadas síncronas e assíncronas. Um `GenServer` implementa funções de callback como `handle_call`, `handle_cast` e `handle_info` para lidar com diferentes tipos de mensagens. Ele é amplamente usado em aplicações robustas e é compatível com o sistema de supervisão padrão de Elixir.

3.7.5. Fluxo Paralelo de Dados

O módulo `Flow`, parte do ecossistema Elixir, oferece uma abstração funcional de alto nível para processamento concorrente e paralelo de coleções de dados. Construído sobre a biblioteca `GenStage`, o `Flow` simplifica a criação de pipelines paralelos, delegando à infraestrutura subjacente a coordenação de demanda, o balanceamento de carga e a divisão do trabalho entre múltiplos processos.

A estrutura de um fluxo é composta por três etapas principais: entrada (*source*), transformação (*transform*) e agregação ou coleta (*sink*). A entrada pode ser uma coleção enumerável, como listas ou streams, e o fluxo é iniciado com `Flow.from_enumerable/1`. A seguir, podem ser encadeadas operações como `map/2`, `filter/2`, `flat_map/2` e `reduce/3`, cada uma aplicada em paralelo por diferentes instâncias de processo. A distribuição do trabalho entre essas instâncias é controlada pela função `Flow.partition/2`, que permite, por exemplo, especificar o número de stages (processos consumidores), chaves de particionamento e estratégias de roteamento.

Por exemplo, a função `Flow.partition(stages: n)` cria `n` stages consumidores, entre os quais os dados são distribuídos com base em um particionador hash ou definido pelo usuário. Cada um desses stages executa em seu próprio processo, permitindo o uso eficiente de múltiplos núcleos da CPU.

O processamento é executado de forma preguiçosa, ou seja, apenas quando uma função terminal como `Enum.to_list/1`, `Enum.reduce/3` ou `Enum.each/2` for chamada. Durante a execução, o `Flow` garante consistência entre os estágios, aplica controle de demanda para evitar sobrecarga (backpressure) e permite modelar sistemas complexos, como pipelines de produtores e consumidores, de forma declarativa.

3.8. Estudos de Caso

Esta seção apresenta dois estudos de caso, a implementação do problema do produtor/consumidor e a implementação do cálculo da n -ésima posição da série de Fibonacci. Neste texto, apenas parte do código é apresentada para ilustrar o uso das ferramentas. Para os estudos de caso, são apresentadas diferentes implementações em C++, Rust, Go e Elixir, sem, contudo, esgotar as possibilidades de construção de soluções aos problemas nessas linguagens. Como nota, cabe informar que as implementações foram concebidas com intuito de oferecer um material didático, e não houve preocupações com questões de desempenho.

3.8.1. Reprodutibilidade

Os exemplos apresentados neste texto encontram-se disponíveis, completos, no repositório criado para complementar o material deste texto <https://github.com/GersonCavalheiro/JAI2025>. O repositório conta também com outras implementações, dos mesmos casos de estudo, e de outros. Para facilitar o uso das ferramentas de programação trabalhadas neste texto, o repositório também conta com um `Dockerfile`, para construção de uma imagem contendo os recursos necessários.

3.8.2. Produtor/Consumidor

Neste algoritmo, produtores e consumidores compartilham uma área de dados comuns, uma lista compartilhada, onde itens produzidos pelos produtores são enfileirados e consumidos pelos consumidores. Os parâmetros de lançamento deste programa são o número de produtores e consumidores a serem instanciados e o número de itens a serem produzidos pelos produtores individualmente. Os itens produzidos são os primeiros números primos. Cada produtor ao completar sua cota de produção, finaliza. A função `main` é responsável por sinalizar os consumidores do término do programa introduzindo o valor `-1` na fila compartilhada de itens.

3.8.2.1. Produtor/Consumidor em C++

Em C++ foram desenvolvidas três versões de programas implementando o algoritmo Produtor-Consumidor. Todas as versões, em comum, o acesso à seção crítica é controlado por mutex, tendo como apoio um segundo mecanismo de sincronização: variável de condição, promessas e futuros, e cancelamento cooperativo.

Na versão empregando variável de condição na sua solução, cada produtor, ao finalizar a produção de um número primo, obtém ingresso a uma seção crítica adquirindo um mutex com o recurso `std::unique_lock`, podendo então o inserir na fila compartilhada protegida de itens produzidos. Após a inserção, os consumidores são sinalizados

da presença de um novo item na lista com uma notificação na variável de condição com `cv.notify_one()` que tem sinaliza algum eventual consumidor que esteja suspenso em espera para realizar um consumo. Cada consumidor, por sua vez, aguarda itens que exista pelo menos um item na fila para ser consumido. Após consumir um item, verifica se é o valor sentinela (número negativo) para encerrar sua execução. O valor sentinela é reenqueueado para acordar outros consumidores restantes. Essa implementação se caracteriza por implementar uma estratégia convencional e robusta e utilizando um mecanismo nativo de espera/bloqueio, garantindo eficiência. No entanto, a programação do algoritmo desta forma exigiu que fossem introduzidos mecanismos para coordenar explicitamente a notificação da produção de itens e acesso em regime de exclusão mútua à lista compartilhada e introduzir explicitamente sinalização da condição de parada. O Código 11 apresenta, de forma simplificada, a estrutura da implementação desta versão do problema.

```

std::mutex mtx;
std::condition_variable cv;
std::queue<int> buffer;

void produtor(int id, int total) {
    int count = 0, num = 2;
    while (count < total) {
        if (is_prime(num)) {
            std::unique_lock<std::mutex> lock(mtx);
            buffer.push(num);
            cv.notify_one();
            count++;
        }
        num++;
    }
}

void consumidor(int id) {
    while (true) {
        std::unique_lock<std::mutex> lock(mtx);
        cv.wait(lock, [] { return !buffer.empty(); });
        int val = buffer.front(); buffer.pop();
        if (val < 0) { buffer.push(val); break; }
        ... // processa item
    }
}

```

Código 11: Produtor-Consumidor em C++ utilizando variável de condição.

Na versão baseada em promessas e futuros, Código 12, a coordenação da finalização entre produtores e consumidores é realizada por meio de objetos `std::promise` e `std::future`, enquanto o acesso à fila compartilhada ainda é protegido por `std::mutex`. Cada produtor, ao gerar um número primo, obtém acesso exclusivo à lista de itens produzidos utilizando `std::lock_guard`, inserindo o item na fila de forma segura. Ao concluir sua participação, o produtor sinaliza seu término invocando `set_value()` em seu objeto `promise`, permitindo que o thread principal detecte que aquela produção foi encerrada por meio da `future` correspondente. Já os consumidores executam de forma contínua, consultando a fila em busca de itens. Caso a fila esteja vazia, aguardam brevemente com `sleep_for()`, realizando uma espera ativa leve. Ao consumir um item, verificam se é o valor de término, o que indica que devem encerrar sua execução. O valor de indicação de término é reenqueueado para garantir que os demais consumidores também recebam o sinal de parada. Essa implementação evita o uso de variáveis de condição, adotando uma abordagem mais explícita de sincronização da finalização dos produtores com base em futuros. Por outro lado, impõe o custo de uma espera ativa no lado do consumidor e exige o uso adicional de objetos de sincronização dedicados ao encerramento do fluxo de produção.

Na terceira versão implementada empregando cancelamento cooperativo, Código 13, é adotado o uso da classe `std::jthread`. Cada thread produtor gera números

```

std::mutex mtx;
std::queue<int> buffer;

void produtor(int id, int total, std::promise<void> prom) {
    int count = 0, num = 2;
    while (count < total) {
        if (is_prime(num)) {
            {
                std::lock_guard<std::mutex> lock(mtx);
                buffer.push(num);
            }
            count++;
            num++;
        }
        prom.set_value(); // sinaliza fim da produção
    }
}

void consumidor(int id) {
    while (true) {
        int val = -1;
        {
            std::lock_guard<std::mutex> lock(mtx);
            if (!buffer.empty())
                { val = buffer.front(); buffer.pop(); }
        }
        if (val == -1) break;
        if (val >= 0) ... // processa item
        else std::this_thread::sleep_for(...);
    }
}

```

Código 12: Produtor-Consumidor em C++ utilizando promessas e futuros.

primos enquanto o token associado não sinalizar uma solicitação de parada. A inserção dos itens na fila compartilhada é feita dentro de uma seção crítica protegida por `std::mutex`, utilizando `std::unique_lock`, seguida de uma notificação aos consumidores com `cv.notify_one()`. Cada thread consumidor, por sua vez, aguarda a chegada de novos itens utilizando uma variável de condição combinada com um predicado que leva em consideração tanto a presença de itens na fila quanto o estado do `stop_token`. Quando a fila estiver vazia e o cancelamento tiver sido solicitado, o consumidor encerra sua execução de forma segura. A sinalização de parada é realizada pelo programa principal após um intervalo predeterminado, utilizando o método `request_stop()` oferecido por `std::jthread`. Essa abordagem destaca-se por integrar de forma coesa o controle de cancelamento ao ciclo de vida das threads, promovendo uma finalização ordenada e segura sem a necessidade de valores sentinela ou mecanismos adicionais de sinalização explícita. Em contrapartida, exige que cada thread verifique periodicamente o estado do token de parada, o que introduz um aspecto de programação cooperativa ao controle de execução.

```

std::mutex mtx;
std::condition_variable cv;
std::queue<int> buffer;

void produtor(std::stop_token st, int id, int total) {
    int count = 0, num = 2;
    while (count < total && !st.stop_requested()) {
        if (is_prime(num)) {
            {
                std::unique_lock<std::mutex> lock(mtx);
                buffer.push(num);
            }
            cv.notify_one();
            count++;
            num++;
        }
    }
}

void consumidor(std::stop_token st, int id) {
    while (!st.stop_requested()) {
        int val = -1;
        {
            std::unique_lock<std::mutex> lock(mtx);
            cv.wait(lock, [&] { return !buffer.empty()
                || st.stop_requested(); });
            if (st.stop_requested()) continue;
            val = buffer.front(); buffer.pop();
        }
        ... // processa item
    }
}

```

Código 13: Produtor-Consumidor em C++ utilizando cancelamento cooperativo.

3.8.2.2. Produtor/Consumidor em Rust

Em Rust foram desenvolvida duas versões para o problema. A primeira utiliza uma estrutura de dados (fila) compartilhada entre produtores e consumidores, protegida por mutex, enquanto a segunda utiliza canais assíncronos para comunicação. Desta forma, a primeira versão implementa uma solução baseada em condição associada ao estado da fila compartilhada, enquanto a segunda emprega uma abordagem baseada em comunicação explícita entre threads.

Na versão empregando uma fila compartilhada, Código 14, a seção crítica acessando o dado compartilhado é protegida por `Mutex` e sincronizada com `Condvar`. Os produtores sinalizam disponibilidade de novos elementos por meio de chamadas a `notify_one()` sobre a variável de condição. Já os threads consumidores aguardam até que haja itens disponíveis na fila (`Condvar::wait`) para suspender sua execução de maneira eficiente até serem notificados. Ao detectar um valor especial (`u32::MAX`) inserido após o término da produção, os consumidores encerram sua execução.

```
use std::sync::{Arc, Mutex, Condvar};
use std::thread;

fn main() {
    let buffer = Arc::new((Mutex::new(Vec::new()), Condvar::new()));
    let buffer_p = Arc::clone(&buffer);
    let produtor = thread::spawn(move || {
        for i in 1..=5 {
            let (lock, cvar) = &*buffer_p;
            let mut buf = lock.lock().unwrap();
            buf.push(i);
            println!("Produtor gerou {}", i);
            cvar.notify_one();
        }
        let (lock, cvar) = &*buffer_p;
        let mut buf = lock.lock().unwrap();
        buf.push(u32::MAX);
        cvar.notify_one();
    });

    let buffer_c = Arc::clone(&buffer);
    let consumidor = thread::spawn(move || {
        loop {
            let (lock, cvar) = &*buffer_c;
            let mut buf = lock.lock().unwrap();
            while buf.is_empty() {
                buf = cvar.wait(buf).unwrap();
            }
            let val = buf.remove(0);
            if val == u32::MAX {
                break;
            }
            ... // processa item
        }
    });

    produtor.join().unwrap();
    consumidor.join().unwrap();
}
```

Código 14: Produtor-Consumidor em Rust utilizando variável de condição.

Na segunda versão implementada em Rust, os itens produzidos são enviados aos consumidores via um canal de comunicação (`std::sync::mpsc::channel`), Código 15. Os consumidores compartilham um único receptor protegido com `Arc<Mutex<Receiver<Item>>`, garantindo que apenas um thread por vez tenha acesso à operação de recebimento. Ao tentar consumir, cada thread adquire o mutex, realiza a chamada a `recv` e, se obtiver sucesso, imprime o item recebido. Quando todos os produtores finalizam, o canal é fechado, e os consumidores detectam o encerramento por meio de um erro retornado por `recv`.

3.8.2.3. Produtor/Consumidor em Go

Na linguagem Go foram realizadas duas implementações do problema do produtor-consumidor, a primeira mais próxima a concepção tradicional da solução, utilizando seções críticas, controladas por mutex, para acesso a dados compartilhados. Já a segunda

```

use std::sync::{Arc, Mutex, mpsc};
use std::thread;
struct Item {
    produtor_id: usize,
    valor: usize,
}
fn main() {
    let (tx, rx) = mpsc::channel();
    let rx = Arc::new(Mutex::new(rx));
    let mut handles = vec![];
    for id in 1..=2 {
        let tx = tx.clone();
        let h = thread::spawn(move || {
            for i in 0..3 {
                let valor = 100 * id + i;
                tx.send(Item { produtor_id: id, valor }).unwrap();
            }
        });
        handles.push(h);
    }
    drop(tx);

    for id in 1..=2 {
        let rx = Arc::clone(&rx);
        let h = thread::spawn(move || {
            loop {
                let item = {
                    let rx = rx.lock().unwrap();
                    rx.recv()
                };
                match item {
                    Ok(Item { produtor_id, valor }) => {
                        ... // processa item
                    }
                    Err(_) => {
                        break;
                    }
                }
            }
        });
        handles.push(h);
    }
    for h in handles {
        h.join().unwrap();
    }
}

```

Código 15: Produtor-Consumidor em Rust utilizando canais de comunicação.

implementação apresenta uma abordagem idiomática mais característica à linguagem, utilizando exclusivamente canais de comunicação.

Na primeira implementação em Go, Código 16, a coordenação entre os threads produtores e consumidores combina o uso de canais de comunicação com seções críticas protegidas por `sync.Mutex`. Os threads produtores, após produzirem cada item, acessam uma variável global para registrar o último número testado. Esse acesso é protegido por uma região crítica, garantindo que apenas um thread por vez atualize o contador global e escreva no vetor compartilhado. Após preencher esse vetor com todos os primos esperados, o canal de comunicação é utilizado para que os consumidores solicitem os valores gerados. Os threads consumidores, por sua vez, leem da lista de primos também dentro de seções críticas protegidas por `sync.Mutex`, incrementando um contador global que determina quando o consumo deve ser encerrado. Essa abordagem adota um modelo híbrido: os dados são transmitidos por canal, mas a sincronização de acesso ao estado global (como contadores e o vetor de primos) é realizada por exclusão mútua.

A segunda versão implementada em Go, Código 17, emprega canais para comunicação e encerramento automático da execução. Cada thread produtor gera números primos (função `ehPrimo`), e os envia por meio de um canal do tipo `Item`, que associa o valor gerado à identificação do produtor. Os threads consumidores leem continuamente do canal compartilhado, processando cada item recebido com uma pequena simulação de carga. O encerramento da execução dos consumidores ocorre de forma natural com o fechamento do canal, acionado automaticamente após a finalização de todos os produtores, monitorada por um `sync.WaitGroup`. Essa estratégia elimina a necessidade de valores sentinela ou verificação de condições globais, simplificando o controle do ciclo de vida das goroutines. A sincronização entre threads ocorre implicitamente por meio do canal, que garante consistência e exclusividade no acesso aos dados trafegados, dispensando o uso de seções críticas.

```

func isPrime(n int) bool { ... }
func producer(id, n int, tx chan int,
wg *sync.WaitGroup,
produced *int, mu *sync.Mutex) {
defer wg.Done()
num := 2
for {
mu.Lock()
if *produced >= n {
mu.Unlock()
break
}
mu.Unlock()

if isPrime(num) {
mu.Lock()
if *produced < n {
tx <- num
*produced++
}
mu.Unlock()
}
num++
}
}

func consumer(id, n int, rx chan int,
wg *sync.WaitGroup,
consumed *int, mu *sync.Mutex) {
defer wg.Done()
for {
mu.Lock()
if *consumed >= n { mu.Unlock() break }
mu.Unlock()
item := <-rx
mu.Lock()
*consumed++
mu.Unlock()
... // processa item
}
}

func main() {
tx := make(chan int, BUFFER_SIZE)
var produced, consumed int
var mu sync.Mutex
var wg sync.WaitGroup
for i := 0; i < numProdutores; i++ {
wg.Add(1)
go producer(i, n, tx, &wg, &produced, &mu)
}
for i := 0; i < numConsumidores; i++ {
wg.Add(1)
go consumer(i, n, tx, &wg, &consumed, &mu)
}
wg.Wait()
}

```

Código 16: Produtor-Consumidor em Go empregando seção crítica.

3.8.2.4. Produtor/Consumidor em Elixir

Em Elixir foram realizadas implementações de três programas para o problema produtor-consumidor. Todas versões empregam canais de comunicação (`send` e `receive`), e fazem uso do paralelismo oferecido pela criação de processos leves com `spawn`.

A versão com canal único centraliza a comunicação em um único processo intermediário que gerencia uma fila de mensagens compartilhada, exigindo lógica de repetição e controle de disponibilidade explícitos. A segunda versão utiliza um processo distribuidor dedicado à entrega das mensagens aos consumidores, adotando uma estratégia de balanceamento em *round-robin*. Já a terceira versão aproveita a biblioteca `Flow`, que abstrai a distribuição e o particionamento dos dados de forma automática, integrando o paralelismo com uma semântica funcional e pipelines concorrentes.

A primeira versão implementada, Código 18, utiliza um processo dedicado como canal de comunicação entre produtores e consumidores. Cada thread produtor envia mensagens do tipo `{:put, valor}` sobre esse canal, o qual mantém internamente uma fila de itens produzidos. Os consumidores, solicitam novos itens com mensagens `{:get, pid}`, e recebem a resposta diretamente por meio de uma nova mensagem enviada ao seu próprio identificador de processo. Essa abordagem exige que cada consumidor implemente lógica de nova tentativa em caso de fila vazia, utilizando `receive` com `after` para repetição.

A segunda versão em Go, 19, substitui a fila centralizada empregada na versão anterior por um processo distribuidor, eliminando a necessidade de polling nos consu-

```

func produtor(id int, n int, tx chan int, wg *sync.WaitGroup) {
    defer wg.Done()
    num := 2
    enviados := 0
    for enviados < n {
        if isPrime(num) {
            tx <- num
            enviados++
        }
        num++
    }
}

func consumer(id int, rx chan int, wg *sync.WaitGroup) {
    defer wg.Done()
    for item := range rx {
        fmt.Printf("Consumidor %d consumiu: %d\n", id, item)
    }
}

func main() {
    tx := make(chan int, 10)
    var wg sync.WaitGroup
    wg.Add(2)
    go produtor(1, 3, tx, &wg)
    go produtor(2, 3, tx, &wg)
    go func() {
        wg.Wait()
        close(tx)
    }()
    wg.Add(1)
    go consumer(1, tx, &wg)
    wg.Wait()
}

```

Código 17: Produtor-Consumidor em Go empregando canais de comunicação.

```

defmodule ProdutorConsumidor do
  canal_loop(items) do
    {:put, item}
    {:get, pid}
    - {:item, valor}
    - {:retry}
    {:shutdown}
  end
  canal_pid = spawn(fn -> canal_loop([]) end)
  Process.register(canal_pid, :canal)
  main(args) do
    gerar_primos(n)
    canal_pid = spawn(...)
    Process.register(canal_pid, :canal)
    for i <- 1..nc, do: spawn(...)
    Enum.chunk_every(primos)
    Enum.each(..., fn -> spawn(...) end)
  end

  produtor(id, primos) doA
    Enum.each(primos)
    send(:canal, {:put, p})
    :timer.sleep(:rand.uniform)
  end
  consumidor(id, timeout 1000) doB
    send(:canal, {:get, self()})
    receive do
      {:item, valor} -> ... processa item
      {:retry}
    after
      timeout -> consumidor(...)
    end
  end
end

```

Código 18: Produtor-Consumidor em Elixir canal único comunicação.

midores e reduzindo o acoplamento entre os produtores e consumidores. O processo distribuidor assume a responsabilidade de receber os itens por um canal de comunicação e os distribuir, em ordem cíclica, aos consumidores.

A terceira versão implementada, Código 20, emprega a biblioteca `Flow` para organizar os produtores como fontes de dados e os consumidores como estágios particionados de um fluxo concorrente. O fluxo é automaticamente particionado com `Flow.partition`, e cada etapa aplica transformações sobre os dados recebidos. O modelo favorece concisão, paralelismo implícito e integração com o paradigma funcional próprio à linguagem, dispensando a implementação manual de lógica de troca de mensagens ou de controle de ciclo de vida dos processos.

3.8.3. Cálculo de Fibonacci

O cálculo da n -ésima posição na série de Fibonacci é dado por $Fibonacci(n)$, quando $n > 1$ é dado por $Fibonacci(n - 1) + Fibonacci(n - 2)$ e para $n = 0$ ou $n = 1$, $Fibonacci(n) = n$. Este problema é frequentemente abordado no estudo de ferramentas para programação

```

defmodule ProdutorConsumidor do
  distribuidor(nc) do
    consumidores = for i <- 1..nc, do: spawn(...)
    loop(consumidores)
  end
  loop(consumidores) do
    receive do
      {:item, valor} ->
        send(dest, {:item, valor})
        loop(resto ++ [dest])
    end
  end
  main(args) do
    buffer_pid = spawn(fn -> distribuidor(nc) end)
    produtores = for i <- 1..np, do: spawn(...)
    Enum.each(produtores, &Process.monitor/1)
    wait_for_produtores(np)
    for _ <- 1..nc, do: send(buffer_pid, {:item, -1})
  end

  wait_for_produtores(n) do
    receive do
      {:DOWN, _, :process, _, _} ->
        wait_for_produtores(n - 1)
    end
  end
  produtor(id, primos, buffer_pid) do
    Enum.each(primos)
    send(buffer_pid, {:item, p})
  end
  consumidor(id) do
    receive do
      {:item, valor} -> ... # processa item
    end
  end
end

```

Código 19: Produtor-Consumidor em Elixir com canal distribuidor.

```

defmodule ProdutorConsumidorFlow do
  main(args) do
    fluxo =
      Flow.from_enumerable(1..n_prod)
      |> Flow.flat_map(fn produtor_id ->
        Enum.map(primos, fn p ->
          {produtor_id, p}
        end) ++ [[:fim_produtores, produtor_id]]
      end)
      |> Flow.partition(stages: n_cons)
      |> Flow.map(fn
        {:fim_produtores, _} -> :ignore
        (_, item) -> consumidor = phash(item, n_cons)
        ... # processa item
      end)
  end

  fluxo
  |> Enum.reduce(%{}, fn
    :ignore, acc -> acc
    consumidor, acc ->
      Map.update(acc, consumidor, 1, &(&1 + 1))
  end)
  Enum.each(1..n_cons, fn id ->
    ...
  end)
end

```

Código 20: Produtor-Consumidor em Elixir com fluxo concorrente.

concorrente, paralela ou distribuída devido ao elevado grau de concorrência recursiva que o problema expõe e, também, ao desbalanceamento regular da carga (o custo para calcular $Fibonacci(n - 1)$ é maior que o custo para calcular $Fibonacci(n - 2)$). Nas implementações realizadas, o programa recebe como parâmetro o valor n que corresponde à posição da série a ser calculada. Deve ser observado que, diferente do exemplo Produtor/Consumidor, o algoritmo de Fibonacci, a princípio, não necessita compartilhamento de memória, sendo a comunicação entre as unidades de trabalho naturalmente modeladas em parâmetros de entrada e retorno de e para funções.

3.8.4. Fibonacci em C++

Nesta linguagem são apresentadas duas implementações. A primeira utiliza criação assíncrona de tarefas e a segunda o mecanismo de cancelamento coletivo com apoio de variável de condição.

As duas implementações realizadas adotam estratégias bastante distintas quanto ao controle e à criação de threads. A versão baseada em `std::async` delega à biblioteca padrão a responsabilidade pela criação e agendamento dos threads, utilizando

`std::future` para recuperar os resultados das computações paralelas. Essa abordagem oferece simplicidade de uso e semântica direta, mas limita o controle sobre o número total de threads ativos e pode provocar sobrecarga do sistema para valores maiores de entrada, já que não há reutilização de recursos ou coordenação centralizada. Em contraste, a implementação baseada em `std::jthread` organiza a execução concorrente por meio de uma fila de tarefas compartilhada e um conjunto fixo de threads gerenciados manualmente, evitando a criação explosiva de threads recursivas. Essa solução adota uma arquitetura de *thread pool*, utilizando sincronização explícita com `std::mutex`, `std::condition_variable` e contadores atômicos, o que permite um uso mais eficiente dos recursos e maior previsibilidade no escalonamento das tarefas.

A primeira implementação, apresentada no Código 21, utiliza `std::async` com a política `std::launch::async`, indicando lançamento da execução assíncrona imediata de uma função em um novo thread. Cada chamada recursiva gera duas novas chamadas concorrentes, encapsuladas como *futures*. A sincronização se dá pela invocação de `get()` sobre cada *future*, o que bloqueia a execução até que o valor esteja disponível.

```
unsigned long long fib(int n) {
    if (n <= 1) return n;
    auto f1 = std::async(std::launch::async, fib, n - 1);
    auto f2 = std::async(std::launch::async, fib, n - 2);
    return f1.get() + f2.get();
}

int main() {
    int n;
    fib(n);
}
```

Código 21: Fibonacci em C++ com tarefas assíncronas.

A segunda versão implementada, Código 22, emprega um conjunto fixo de threads com instanciadas com `std::jthread`, cada thread responsável por processar tarefas enfileiradas em uma estrutura compartilhada. O envio de novas tarefas à fila é feito com o método `enqueue()`, e as tarefas internas realizam chamadas recursivas agendadas, que ao completarem sinalizam a continuidade da computação por meio de funções de retorno (*callbacks*) e contadores atômicos.

3.8.4.1. Fibonacci em Rust

Em Rust, o algoritmo de Fibonacci teve duas implementações. A primeira com criação explícita recursiva de threads. A segunda foi implementada utilizando um pool de threads, suportada por uma *crate* (biblioteca) não oficial da linguagem amplamente aceita pela comunidade (*rayon*).

A primeira versão desenvolvida, código simplificado em Código 23, é implementada utilizando os recursos básicos de Rust, criando explicitamente threads com `std::thread::spawn`. A sincronização é feita pela invocação ao `join` sobre o thread criado, obtendo o resultado do cálculo do thread.

A segunda versão implementada, Código 24, adota o modelo de composição de tarefas, responsáveis pelo cálculo, submetidas a um pool de execução, suportado pela *crate rayon*.

```

class ThreadPool {
    std::vector<std::jthread> threads;
    std::queue<std::function<void()>> tasks;
    std::mutex mtx; std::condition_variable cv;
    bool done = false;
    thread_loop(stop_token st) {
        while (!st.stop_requested()) {
            wait_for_task();
            execute_task();
        }
    }
    enqueue(task) {
        lock_guard(mtx); tasks.push(task); cv.notify_one();
    }
    ~ThreadPool() {
        lock_guard(mtx); done = true; cv.notify_all();
    }
}

fib_parallel(pool, n, callback) {
    shared left, right, pending = 2;

    done = [=]() {
        if (--pending == 0) callback(left + right);
    };
    pool.enqueue([=]() {
        fib_parallel(..., [=](res) {
            left = res;
            done();
        });
    });
    pool.enqueue([=]() {
        fib_parallel(..., [=](res) {
            right = res;
            done();
        });
    });
}

main() {
    int n;
    ThreadPool pool(hardware_concurrency());
    resultado = 0; done = false;
    fib_parallel(pool, n, [&](res) {
        resultado = res; done = true; cv.notify_one();
    });
    wait_until(done);
    print resultado;
}

```

Código 22: Fibonacci em C++ com tarefas assíncronas.

```

use std::thread;
fn fib(n: u32) -> u64 {
    if n <= 1 {
        return n as u64;
    }
    let t1 = thread::spawn(move || fib(n - 1));
    let t2 = thread::spawn(move || fib(n - 2));
    t1.join().unwrap() + t2.join().unwrap()
}

fn main() {
    n
    let res = fib(n);
}

```

Código 23: Fibonacci em Rust com criação recursiva de threads.

3.8.4.2. Fibonacci em Go

Em Go foram implementadas duas versões para o cálculo de Fibonacci. A primeira utilizando uma área de memória compartilhada, a segunda exclusivamente canais.

A primeira implementação apresentada, Código 25, adota uma abordagem baseada no uso de `sync.WaitGroup` para coordenar a conclusão das goroutines e acesso atômico no acesso à variável compartilhada que acumula o resultado. Esta variável compartilhada é instanciada na função `main`, e seu endereço é passado recursivamente a todas as goroutines criadas. Cada chamada recursiva, ao concluir seu cálculo, atualiza essa variável de forma segura, usando o `mutex` para evitar condições de corrida. O `WaitGroup` garante que todas as chamadas concorrentes terminem antes que o programa principal continue, assegurando a consistência do valor final.

A segunda implementação, Código 26, é mais idiomática para a linguagem Go. Ela também utiliza uma estratégia recursiva, mas utiliza exclusivamente canais para comunicação entre as goroutines, instanciadas para os cálculos para execução assíncrona,

```

fn fib(n) {
  if n <= 1 { return n; }
  (a, b) = rayon::join(
    || fib(n - 1, threshold),
    || fib(n - 2, threshold)
  )
  return a + b
}

main() {
  n
  resultado = fib(n)
}

```

Código 24: Fibonacci em Rust com pool de threads.

```

var mu sync.Mutex
func fib(n, res *uint64, wg *sync.WaitGroup) {
  defer wg.Done()
  if n <= 1 {
    mu.Lock()
    *res += uint64(n)
    mu.Unlock()
    return
  }
  var r1, r2 uint64
  var wgInner sync.WaitGroup
  wgInner.Add(2)
  go fib(n-1, &r1, &wgInner)
  go fib(n-2, &r2, &wgInner)
  wgInner.Wait()
  mu.Lock()
  *res += r1 + r2
  mu.Unlock()
}

func main() {
  var res uint64
  var wg sync.WaitGroup
  wg.Add(1)
  fib(n, &res, &wg)
  wg.Wait()
}

```

Código 25: Fibonacci em Go com WaitGroup e Mutex.

mais especificamente, do envio do retorno do dado calculado pela goroutine. A função principal inicia a computação em uma goroutine e aguardando o resultado final por meio do canal principal.

```

func fib(n, ch chan <- uint64) {
  if n <= 1 {
    ch <- uint64(n)
    return
  }
  ch1 := make(chan uint64)
  ch2 := make(chan uint64)
  go fib(n-1, ch1)
  go fib(n-2, ch2)
  res1 := <-ch1
  res2 := <-ch2
  ch <- res1 + res2
}

func main() {
  n
  resultCh := make(chan uint64)
  go fib(n, resultCh)
  result := <-resultCh
}

```

Código 26: Fibonacci em Go com sincronização via canais.

3.8.4.3. Fibonacci em Elixir

Em Elixir foram produzidas duas versões para cálculo de Fibonacci. A primeira versão explora os mecanismos básicos do modelo de concorrência de Elixir, processos leves e

troca de mensagens. A segunda abstrai o uso direto de processos e mensagens, utilizando tarefas assíncronas.

A primeira implementação, Código 27, explora recursos oferecidos pelo modelo básico de concorrência da linguagem. São criados dois processos leves com `spawn`, um para cada chamada recursiva, e a comunicação entre esses processos é realizada por meio de troca de mensagens `send/receive`.

```
defmodule Fib do
  def compute(n) when n <= 1, do: n
  def compute(n) do
    parent = self()
    spawn(fn ->
      send(parent, {:fib1, compute(n - 1)})
    end)
    spawn(fn ->
      send(parent, {:fib2, compute(n - 2)})
    end)
    receive do
      {:fib1, res1} ->
        receive do
          {:fib2, res2} ->
            res1 + res2
          end
        end
      end
  end
end

defmodule Main do
  def main(_) do
    n = 10
    result = Fib.compute(n)
    end
  end
  Main.main([])
end
```

Código 27: Fibonacci em Elixir com troca de mensagens explícitas.

A segunda implementação, Código 28, utiliza o módulo `Task`, de forma a abstrair a criação de processos e a troca de mensagens, lançando a execução de tarefas assíncronas responsáveis pelo cálculo. Cada chamada recursiva é efetivada com uma chamada à `Task.async`, sendo os resultados são coletados com `Task.await`.

```
defmodule Fib do
  def compute(n) when n <= 1, do: n
  def compute(n) do
    task1 = Task.async(fn -> compute(n - 1) end)
    task2 = Task.async(fn -> compute(n - 2) end)
    res1 = Task.await(task1)
    res2 = Task.await(task2)
    res1 + res2
  end
end

defmodule Main do
  def main(_) do
    n = 10
    result = Fib.compute(n)
    end
  end
  Main.main([])
end
```

Código 28: Fibonacci em Elixir com tarefas assíncronas.

3.9. Considerações Finais

Este material apresentou uma visão comparativa dos principais recursos de programação concorrente oferecidos pelas linguagens C++, Rust, Go e Elixir. Cada uma dessas linguagens adota modelos distintos para expressar a concorrência. Enquanto C++ privilegia o controle explícito por meio de threads nativos e mecanismos como `std::mutex` e `std::atomic`, Rust promove segurança por construção, com verificações em tempo de compilação que restringem o compartilhamento inseguro de dados. A linguagem Go posiciona-se com um modelo de programação que explora canais de comunicação entre goroutines integradas ao seu *runtime*, enquanto Elixir estrutura a concorrência com

base no modelo de atores, promovendo isolamento entre processos leves e troca assíncrona de mensagens.

As diferenças entre essas abordagens não se limitam à sintaxe ou às bibliotecas disponíveis, mas refletem decisões fundamentais de projeto, moldando a forma como os programas concorrentes são estruturados, sincronizados e executados. Nesse sentido, o objetivo deste texto não foi eleger uma linguagem ou modelo superior, mas oferecer subsídios para que o leitor compreenda essas escolhas e suas implicações práticas e conceituais. A identificação da linguagem mais adequada para um determinado problema será, em grande parte, resultado da experiência prática acumulada pelo programador, combinada com o conhecimento das características de cada modelo de execução.

O conteúdo aqui apresentado deve ser entendido como um ponto de partida. Em nenhum momento buscou-se extrair o máximo potencial de cada linguagem ou esgotar suas possibilidades. Cabe ao leitor, motivado por interesse e curiosidade, aprofundar-se por meio das referências citadas, da documentação oficial e das comunidades ativas que sustentam o desenvolvimento dessas ferramentas, tendo sempre em mente que a construção de competência em programação multithread requer prática constante, análise crítica e familiaridade com os padrões idiomáticos de cada ambiente.

A programação concorrente permanece como uma das áreas mais desafiadoras e dinâmicas da Computação. Com a crescente adoção de arquiteturas paralelas nos mais diversos campos de aplicação, o domínio das abstrações, modelos e ferramentas aqui discutidos torna-se cada vez mais essencial. Recomenda-se que o aprofundamento seja feito em uma ou algumas poucas linguagens, favorecendo o domínio técnico, mas sempre com atenção à diversidade de opções que surgem e evoluem continuamente. Espera-se que este material contribua para esse processo de aprendizado e fomente investigações futuras.

Referências

- [Amdahl 1967] Amdahl, G. M. (1967). Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the April 18-20, 1967, spring joint computer conference*, pages 483–485.
- [Andrews 1999] Andrews, G. R. (1999). *Foundations of Parallel and Distributed Programming*. Addison-Wesley Longman Publishing Co., Inc., USA, 1st edition.
- [Boehm 2005] Boehm, H.-J. (2005). Threads cannot be implemented as a library. *SIG-PLAN Not.*, 40(6).
- [Burns and Wellings 1998] Burns, A. and Wellings, A. (1998). *Concurrency in Ada (2nd ed.)*. Cambridge University Press, USA.
- [Cavaleiro and Du Bois 2014] Cavaleiro, G. G. H. and Du Bois, A. R. (2014). Ferramentas modernas para programação multicore. In Salgado, A. C., Lóscio, B. F., Alchieri, E., and Barreto, P. S., editors, *Atualizações em Informática 2014*, volume 1, pages 41–83. SBC, Porto Alegre.

- [Cavaleiro and Santos 2007] Cavaleiro, G. G. H. and Santos, R. R. (2007). Multiprogramação leve em arquiteturas multi-core. In Kowaltowski, T. and Breitman, K. K., editors, *Atualizações em Informática 2007*, pages 327–379. PUC-Rio, Rio de Janeiro.
- [Cavalheiro 2009] Cavalheiro, G. G. H. (2009). Programação com pthreads. In Mattos, J. C. B., Da Rosa Junior, L. S., and Pilla, M. L., editors, *Desafios e Avanços em Computação: O Estado da Arte*, pages 137–151. Editora e Gráfica Universitária - PREC UFPel, Pelotas.
- [Chandra et al. 2001] Chandra, R., Dagum, L., Kohr, D., Maydan, D., McDonald, J., and Menon, R. (2001). *Parallel programming in OpenMP*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.
- [Cox-Buday 2017] Cox-Buday, K. (2017). *Concurrency in Go: Tools and Techniques for Developers*. O’Reilly Media, Inc., 1st edition.
- [Culler et al. 1999] Culler, D. E., Singh, J., and Gupta, A. (1999). *Parallel Computer Architecture: A Hardware/Software Approach*. Morgan Kaufmann.
- [Gupta et al. 2021] Gupta, M., Bhargava, L., and Indu, S. (2021). Mapping techniques in multicore processors: current and future trends. *The Journal of Supercomputing*, 77(8):9308–9363.
- [Herlihy et al. 2020] Herlihy, M., Shavit, N., Luchangco, V., and Spear, M. (2020). *The Art of Multiprocessor Programming*. Morgan Kaufmann, 2nd edition.
- [Kleiman et al. 1996] Kleiman, S., Shah, D., and Smaalders, B. (1996). *Programming with Threads*. Sun Soft Press ; Prentice Hall, Mountain View, Calif. ; Upper Saddle River, NJ.
- [Marx et al. 2018] Marx, B., Tate, B., and Valim, J. (2018). *Adopting Elixir: From Concept to Production*. Pragmatic Bookshelf, Dallas, TX.
- [McCool et al. 2012] McCool, M., Robison, A. D., and Reinders, J. (2012). *Structured Parallel Programming: Patterns for Efficient Computation*. Morgan Kaufmann Publishers, an imprint of Elsevier.
- [Pacheco 2022] Pacheco, P. (2022). *An Introduction to Parallel Programming*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2nd edition.
- [Pilla et al. 2009] Pilla, M. L., Santos, R. R., and Cavaleiro, G. G. H. (2009). Introdução à programação para arquiteturas multicore. In Dorneles, R. V. and Stein, B., editors, *IX Escola Regional de Processamento de Alto Desempenho*, pages 71–202. SBC, Porto Alegre.
- [Rauber and Rüniger 2010] Rauber, T. and Rüniger, G. (2010). *Parallel Programming for Multicore and Cluster Systems*. Springer.
- [Silberschatz et al. 2018] Silberschatz, A., Galvin, P. B., and Gagne, G. (2018). *Operating System Concepts*. Wiley, 10 edition.

- [Silva et al. 2022] Silva, G., Bianchini, C., and Costa, E. (2022). *Programação Paralela e Distribuída com MPI, OpenMP e OpenACC para computação de alto desempenho*.
- [Skillicorn and Talia 1998] Skillicorn, D. B. and Talia, D. (1998). Models and languages for parallel computation. *ACM Comput. Surv.*, 30(2):123–169.
- [Sutter and Larus 2005] Sutter, H. and Larus, J. (2005). Software and the concurrency revolution: Leveraging the full power of multicore processors demands new tools and new thinking from the software industry. *Queue*, 3(7):54–62.
- [Tanenbaum and Bos 2022] Tanenbaum, A. S. and Bos, H. (2022). *Modern Operating Systems*. Prentice Hall, 5 edition.
- [Team 2021] Team, T. R. (2021). *The Rust Programming Language*. Rust Core Team. Official language guide.
- [Thomas 2022] Thomas, D. (2022). *Programming Elixir 1.6: Functional |> Concurrent |> Pragmatic |> Fun*. Pragmatic Bookshelf, Raleigh, North Carolina, 7 edition.
- [Troutwine 2018] Troutwine, B. L. (2018). *Hands-On Concurrency with Rust: Confidently build memory-safe, parallel, and efficient software in Rust*. Packt, Birmingham.
- [Williams 2019] Williams, A. (2019). *C++ Concurrency in Action, Second Edition*. Manning, 2 edition.

Capítulo

4

Incorporação de Modelos de Linguagem em Larga Escala em Dispositivos Móveis: Otimização, Personalização e Desafios

Nicollas R. de Oliveira (UFF), João Vitor V. Silva (UFF), Guilherme N. N. Barbosa (UFF), Dianne S. V. de Medeiros (UFF), Diogo M. F. Mattos (UFF)

Abstract

The chapter discusses techniques and tools for optimizing and customizing Large Language Models (LLMs) for resource-constrained devices. The high computational cost and power consumption challenges of these models limit their applicability in environments with diverse capabilities. Thus, the chapter introduces models such as GPT, Llama, and Falcon, based on architectures such as AutoEncoders and Transformers, which have transformed natural language processing with impressive text understanding and generation capabilities. Quantization is presented as a crucial technique for improving the usability of models on resource-constrained devices. Finally, the practical applications of adapting and embedding quantized models on resource-constrained devices and the risks associated with the use of LLMs are considered.

Resumo

O capítulo discute as técnicas e ferramentas para otimizar e personalizar modelos de linguagem em larga escala (Large Language Models - LLMs) para dispositivos com recursos limitados. Os desafios impostos pelo alto custo computacional e consumo de energia desses modelos limitam sua aplicabilidade em ambientes com capacidades variadas. Assim, o capítulo apresenta modelos como GPT, Llama e Falcon, baseados em arquiteturas como AutoEncoders e Transformers, que transformaram o processamento de linguagem natural com impressionantes capacidades de compreensão e geração de texto. A quantização é apresentada como uma técnica crucial para aprimorar a usabilidade dos modelos em dispositivos limitados. Por fim, as aplicações práticas da adaptação e incorporação de modelos quantizados em dispositivos com recursos limitados e os riscos associados ao uso de LLMs são elencados.

Este capítulo foi realizado com recursos do CNPq, CAPES, RNP e FAPERJ. Ferramentas de Inteligência Artificial Generativa, incluindo ChatGPT, Grammarly e Llama3.1, foram empregadas na revisão textual deste trabalho.

4.1. Introdução

A popularização de Modelos de Linguagem em Larga Escala (*Large Language Models* - LLMs) implica uma profunda transformação no campo do processamento de linguagem natural. Inspirados pelo GPT-3, diversos modelos, como OPT, PaLM, BLOOM, Chinchilla e LLaMA, demonstraram que o aumento na escala do modelo resulta em capacidades de compreensão e geração aprimoradas. Consequentemente, modelos com dezenas, ou até mesmo centenas, de bilhões de parâmetros se tornaram um padrão no cenário atual da inteligência artificial. Yang *et al.* argumentam que a proficiência linguística de LLMs está correlacionada à presença de quatro características principais [Yang et al., 2024]. A primeira delas é a compreensão contextual profunda, que envolve a capacidade de resolver ambiguidades semânticas bem como identificar relações discursivas. A segunda característica é a geração textual fluente, que implica a habilidade de produzir textos gramaticalmente corretos, com vocabulário diverso e não repetitivo, e adequados ao contexto pragmático exigido, considerando diferentes estilos e tons. A terceira característica é a consciência de domínio do conhecimento e abrange a capacidade de realizar inferências a partir de informações implícitas e adaptar o discurso a diferentes níveis de especialização. Por fim, destaca-se a execução direcionada, característica especialmente útil na resolução de problemas complexos e na tomada de decisões a partir de diretrizes formuladas em linguagem natural pelo usuário. Essas habilidades estão intimamente atrelada à engenharia de *prompt*, que se dedica a formular instruções, em linguagem natural, eficazes para eliciar as respostas desejadas do modelo. Essas competências fazem dos LLMs ferramentas fundamentais em diversos campos, incluindo análise de texto, geração de conteúdo e suporte à decisão.

A implementação dos LLMs, entretanto, enfrenta desafios significativos quando realizada em dispositivos móveis, de computação de borda ou com recursos computacionais limitados. Além de questões tradicionais relacionadas à implantação em servidores na nuvem, como latência, segurança e conectividade contínua, a execução de LLMs em dispositivos restritos em recursos computacionais exige soluções que minimizem os altos custos computacionais e a pegada ambiental associada [Xu et al., 2024]. Para abordar esses desafios, técnicas de otimização de uso de memória e técnicas avançadas de quantização têm sido desenvolvidas. Métodos como a quantização pós-treinamento em 8 bits de pesos e ativações [Yao et al., 2024a] permitem reduzir significativamente o consumo de memória e energia sem comprometer a precisão. No entanto, modelos como o LLaMA, com 65 bilhões de parâmetros, continuam a exigir recursos significativos de memória, consumindo até 65 GB de VRAM apenas para armazenar os pesos. Em aplicações de sequência longa, os caches de chave-valor podem alcançar dezenas de *gigabytes*, excedendo os recursos de memória e processamento disponíveis [Zhao et al., 2024].

Diante desse contexto, este capítulo tem como objetivo aprofundar o conhecimento sobre modelos de linguagem em larga escala e suas aplicações em dispositivos com recursos computacionais limitados, como *smartphones* e dispositivos embarcados. O capítulo busca apresentar os fundamentos dos LLMs, incluindo suas arquiteturas baseadas em *Transformers* e *AutoEncoders*, e também explorar os desafios práticos associados à sua implementação em ambientes restritos. Entre esses desafios, destacam-se o alto consumo de memória e energia, que tornam inviável a execução direta de modelos de grande porte em dispositivos móveis ou embarcados. Dessa forma, o capítulo adota uma abor-

dagem que combina teoria e prática, capacitando os leitores a entenderem os principais aspectos técnicos e metodológicos necessários para a adaptação eficiente desses modelos. Um dos focos centrais do capítulo é a otimização de LLMs para execução eficiente em dispositivos móveis. Para tanto, são exploradas diversas técnicas, como quantização, poda e destilação de conhecimento, com ênfase especial na quantização. O capítulo discute o impacto dessas técnicas na eficiência computacional e na viabilidade do uso de LLMs em cenários em que o processamento local é essencial para reduzir dependências de conectividade com a nuvem e melhorar a privacidade dos dados.

O capítulo também aborda estratégias para personalização e adaptação de modelos de linguagem em cenários específicos. Isso inclui a configuração de fluxos de processamento, o ajuste fino (*fine-tuning*) para domínios específicos e a incorporação de técnicas que permitam a execução desses modelos em plataformas como Open WebUI. A atividade prática descrita proporciona aos leitores a oportunidade de experimentar diretamente a quantização e personalização de LLMs, avaliando seu impacto no consumo de recursos e na qualidade das respostas geradas.

O restante do capítulo está organizado da seguinte forma. A Seção 4.2 apresenta os fundamentos do processamento de linguagem natural. A Seção 4.3 discute a arquitetura de modelos de linguagem em larga escala. O processo de treinamento dos modelos e as técnicas de otimização do uso de recursos computacionais são apresentados na Seção 4.4. Classes e exemplos de modelos são elencados na Seção 4.5. As métricas para avaliação de modelos de linguagem em larga escala são apresentadas na Seção 4.6. A atividade prática é descrita na Seção 4.7 e a Seção 4.8 discute desafios e tendências de pesquisa. Por fim, as considerações finais são expostas na Seção 4.9.

4.2. Fundamentos e Evolução da Linguística Computacional

O Processamento de Linguagem Natural (*Natural Language Processing* - NLP), ou linguística computacional, dedica-se ao desenvolvimento de sistemas computacionais para compreender e manipular a linguagem humana. Esse processo requer o uso de *corpora* robustos, coleções estruturadas de textos que sustentam análises em níveis morfológicos, sintáticos, semânticos e pragmáticos. A adequação dos *corpora* à tarefa e o equilíbrio entre a diversidade e especificidade do seu vocabulário influenciam diretamente o aprendizado do modelo, afetando sua capacidade de interpretar ocorrências inéditas ou combinações de padrões gramaticais previamente conhecidos. Independentemente do objetivo, a condução do processamento textual inicia com o pré-processamento dos dados brutos, que podem ser estruturados ou não.

O *pipeline* genérico de pré-processamento dos dados utilizados no treinamento de modelos de linguagem é ilustrado na Figura 4.1. Esse *pipeline* organiza o fluxo de transformação dos dados brutos até sua conversão em representações tokenizadas, apropriadas para uso em etapas seguintes como vetorização e treinamento do modelo de linguagem. Inicialmente, os dados brutos são coletados de diferentes fontes, podendo ser estruturados, semi-estruturados ou não estruturados. Em seguida, ocorre a etapa de **filtragem e seleção**, na qual são aplicadas abordagens que visam garantir a qualidade, relevância e consistência dos dados utilizados. Essa etapa pode incluir (i) uma *Filtragem por idioma*, por meio de modelos de detecção automática, como LangID ou FastText; (ii) uma *Fil-*

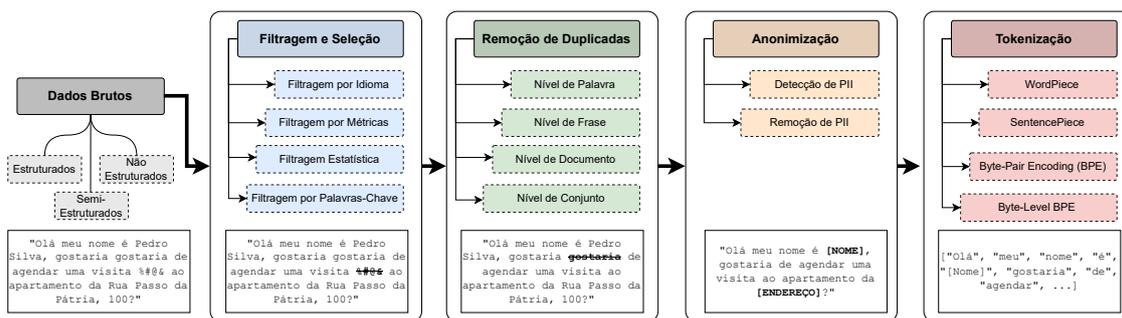


Figura 4.1. Pipeline genérico de pré-processamento de dados usados no treinamento de modelos de linguagem em larga escala. Adaptado de [Zhao et al., 2023].

tragem por métricas, com base em legibilidade, coerência ou comprimentos típicos de sequência; (iii) uma *Filtragem estatística*, para remoção de *outliers*, dados ruidosos ou altamente redundantes; ou até (iv) uma *Filtragem por palavras-chave*, útil para garantir que os dados estejam alinhados a domínios de interesse. Na sequência, realiza-se a **remoção de duplicatas**, etapa essencial para mitigar o risco de sobreajuste durante o treinamento dos modelos. A detecção de duplicatas pode ser conduzida em diferentes níveis de granularidade, começando pelo nível de palavra, em que repetições internas são identificadas por comprometerem a naturalidade e diversidade linguística do *corpus*. Em seguida, no nível de frase, são aplicadas técnicas de comparação exata ou medidas de similaridade semântica para detectar redundâncias parciais. No nível de documento, busca-se identificar cópias completas de textos, garantindo que o modelo não aprenda padrões repetidos de maneira artificial. Por fim, a análise no nível de conjunto permite identificar sobreposição entre diferentes subconjuntos de dados, como entre múltiplas fontes de coleta, prevenindo assim a contaminação cruzada e garantindo a diversidade real do conjunto final de treinamento. A etapa seguinte corresponde à **anonimização**, com foco na identificação e eliminação de informações pessoais identificáveis (*Personally Identifiable Information – PII*). Esta tarefa compreende duas subetapas: (i) *detecção de PII*, que pode ser realizada por meio de expressões regulares, modelos de Reconhecimento de Entidades Nomeadas (NER), ou classificadores supervisionados; e (ii) *remoção ou substituição*, geralmente através de máscaras genéricas, de modo a garantir a privacidade e conformidade com regulamentos como a LGPD ou GDPR. Por fim, o pipeline realiza a **tokenização**, processo de segmentação do texto em unidades menores chamadas *tokens*. Essa segmentação pode ocorrer em diferentes níveis, dependendo do modelo ou da tarefa-alvo. Dentre os métodos clássicos incluem: (i) *WordPiece*, utilizado por modelos como BERT, que combina palavras frequentes com subpalavras raras; (ii) *SentencePiece*, amplamente utilizado em modelos treinados em múltiplos idiomas, por sua independência de pré-tokenização; (iii) *Byte-Pair Encoding (BPE)*, que substitui pares de caracteres ou subpalavras mais frequentes; (iv) *Byte-Level BPE*, uma extensão do BPE que opera diretamente no nível de bytes, mantendo compatibilidade com textos não normalizados. Esses algoritmos permitem representar texto de forma compacta e eficiente, facilitando o aprendizado de representações semânticas. A preservação de contexto e a modelagem de dependências entre os *tokens* são posteriormente tratadas pelos modelos de linguagem, que capturam essas relações em seus mecanismos internos de atenção ou recorrência.

Modelos de linguagem em larga escala podem ser compreendidos como teorias

Tabela 4.1. Comparativo detalhado dos formatos de precisão numérica nas diferentes gerações de modelos de linguagem.

	SLMs	NLMs	PLMs	LLMs
Abordagem Principal	Regras e/ou Contagem Estatística	Redes Neurais	Transferência de Aprendizado	Transformers em Escala Massiva
Dados de Treinamento	Corpora específicos de tamanho limitado	Corpora de tamanho moderado	Grandes corpora diversificados	Corpora massivos
Representação Semântica	N-grams, Markov Chains	Embeddings densos	Embeddings contextuais	Embeddings contextuais profundos e ricos
Transferência de Aprendizado	Limitada/Inexistente	Limitada	Alta	Muito Alta
Capacidade de Generalização	Baixa	Moderada	Boa	Excelente
Complexidade do Modelo	Baixa	Moderada	Alta	Muito Alta
Requisitos de Hardware	Baixos	Moderados	Altos	Muito Altos
Capacidade de Raciocínio	Praticamente Inexistente	Limitada	Moderada	Avançada
Aplicação Típica	Reconhecimento de Fala, N-grams	Tradução, Geração Simples	Resposta a Perguntas, Análise de Sentimento	Geração de Texto, Sumarização, Chatbots, Geração de Código

linguísticas computacionais, capazes de codificar princípios estruturais não triviais que facilitam tanto a aquisição quanto o processamento da linguagem [Baroni, 2021]. Antes de serem treinados com dados específicos de uma língua, as arquiteturas dessas redes profundas atuam como teorias gerais, definindo um espaço de gramáticas possíveis. Após o treinamento com dados de uma língua específica, os LLMs se comportam como uma gramática funcional, isto é, sistemas computacionais capazes de, dado um enunciado em uma língua, prever se a sequência é aceitável para um falante idealizado dessa língua.

A evolução dos modelos de linguagem demonstra uma sucessão de paradigmas, representados por quatro categorias principais, cada qual com abordagens, técnicas e capacidades específicas para o processamento de dados linguísticos. A Tabela 4.1 compara e resume as características das diversas gerações de modelos de linguagem. Sendo a primeira geração de modelos de linguagem, os **Modelos Estatísticos de Linguagem** (*Statistical Language Models* - SLMs) basearam seu desenvolvimento em métodos de aprendizado estatístico. Estes modelos utilizam a suposição de Markov para prever palavras em uma sequência, conhecida como abordagem de modelos *n-gram*. Dessa forma, os SLMs preveem uma palavra com base nas *n* palavras imediatamente anteriores, mas enfrentam desafios significativos devido à escassez de dados, especialmente ao tentar estimar modelos de alta ordem. Para mitigar esse problema, técnicas de suavização, como a suavização aditiva e as estimativas de *Good-Turing* e *backoff*, são largamente empregadas. Tais estratégias focam na redistribuição das probabilidades calculadas diretamente das frequências observadas no *corpus*, garantindo que nenhuma probabilidade seja zero e ajustando as probabilidades baixas para valores mais realistas. Embora eficazes em tarefas básicas, como correção ortográfica, os SLMs enfrentam dificuldades ao lidar com dependências de longo alcance e são limitados pelo crescimento exponencial das probabilidades de transição necessárias para modelos de ordem superior.

Os **Modelos de Linguagem Neural** (*Neural Language Models* - NLMs) representam um avanço significativo em relação à primeira geração de modelos, sobretudo na superação das limitações na captura de relações semânticas e dependências de longo

alcance. Esse progresso é amplamente atribuído ao uso de técnicas de incorporação de palavras (*word embeddings*), que são representações distribuídas em que palavras são mapeadas para vetores densos em um espaço contínuo. Essas representações permitem que os NLMs identifiquem semelhanças entre palavras com base em seu contexto de uso, ampliando sua capacidade de generalização e a compreensão semântica. Arquiteturas de NLMs como Redes Neurais Recorrentes (*Recurrent Neural Networks - RNNs*) e suas variantes mais avançadas, como LSTMs (*Long Short-Term Memory*) e GRUs (*Gated Recurrent Unit*), são projetadas para lidar com dados sequenciais, o que permite que o modelo retenha informações ao longo de uma sequência de palavras. No entanto, os NLMs demandam grande volume de dados para treinamento eficaz e apresentam alto custo computacional, tanto em treinamento quanto em inferência.

Posteriormente, os **Modelos de Linguagem Pré-treinados** (*Pre-trained Language Models - PLMs*) introduziram uma abordagem inovadora baseada no pré-treinamento em *corpora* extensos não rotulados, seguido pelo ajuste fino (*fine-tuning*) para tarefas específicas. Sendo o mais proeminente da geração, o modelo BERT (*Bidirectional Encoder Representations from Transformers*) adaptou a então recente arquitetura *Transformer* e a combinou com um mecanismo de autoatenção para oferecer representações contextuais bidirecionais robustas. Esse modelo elevou significativamente o desempenho em tarefas como geração de diálogo e detecção de emoções em textos. A eficácia do BERT, juntamente com modelos subsequentes, catalisou o desenvolvimento de PLMs com arquiteturas otimizadas e estratégias de pré-treinamento mais sofisticadas, como o GPT e o RoBERTa.

O surgimento dos **Modelos de Linguagem em Larga Escala** (*Large Language Models - LLMs*) resulta da aplicação prática do ganho de escala, aprimorando arquiteturas e métodos de treinamento herdados dos PLMs. Explora-se a correlação entre o aumento do tamanho do modelo, dado pelo número de parâmetros, e o volume de dados de treinamento, refletindo em melhorias abruptas de desempenho em tarefas generativas e compreensivas. Comparativamente, enquanto PLMs detêm um treinamento pautado em *corpora* compostos por centenas de milhões até poucos bilhões de *tokens*, os LLMs como o GPT-3 alcançam centenas de bilhões de *tokens*. Além das aplicações tradicionais de PLN, como análise de sentimento, tradução e sumarização automática multilíngue, os LLMs têm atuado de forma semiautomática em tarefas que antes exigiam alta supervisão humana. Entre essas tarefas, destacam-se: (i) a didática, auxiliando na prática conversacional em ambientes de aprendizado de idiomas; (ii) a extração e correlação de conhecimentos complexos estruturados em textos acadêmicos [Dagdelen et al., 2024]; (iii) a segurança digital, detectando indícios de desinformação em textos ou *links* compartilhados; e (iv) o gerenciamento de redes, atuando como assistentes conversacionais inteligentes na interpretação de intenções do operador [de Oliveira et al., 2025].

4.3. Estrutura e Funcionamento de Modelos de Linguagem em Larga Escala

Os LLMs utilizam uma arquitetura que processa dados textuais provenientes de múltiplas fontes. Após a etapa inicial de pré-processamento, o treinamento envolve estágios como inicialização aleatória de parâmetros, cálculo da função de perda, otimização iterativa e ajustes dos parâmetros. Esses modelos oferecem serviços como tradução de texto, sumarização, análise de sentimentos e aplicações específicas em domínios como

ciências médicas e política. Com o surgimento de modelos avançados como GPT, LLaMa e Bard, além de variantes como Alpaca e GPT-Huggingface, os LLMs consolidaram-se como ferramentas essenciais para tarefas de NLP [Raiaan et al., 2024].

A arquitetura *Transformer* é a base dos LLMs devido à sua eficiência no processamento de sequências textuais. Diferentemente de métodos tradicionais baseados em iteração, o *Transformer* emprega mecanismos de atenção para identificar relações contextuais entre diferentes partes do texto. Essa abordagem permite o processamento de sequências de tamanhos variados e garante maior flexibilidade, tornando-se a arquitetura predominante em tarefas de PLN [Vaswani et al., 2017]. Os componentes da arquitetura *Transformer*, elencados a seguir, desempenham papéis específicos e integrados em seu funcionamento. O texto de entrada é dividido em *tokens*, que podem ser palavras ou subpalavras, e transformado em representações numéricas chamadas incorporações ou *embeddings*, que capturam similaridades semânticas entre palavras. Para garantir que a ordem das palavras seja considerada, informações posicionais são incorporadas a esses *embeddings*. O codificador (*encoder*) processa a entrada e gera representações ocultas que capturam o significado contextual do texto, enquanto o decodificador (*decoder*) utiliza essas informações para prever a próxima palavra em uma sequência, baseando-se no contexto anterior. Durante esse processo, os *embeddings* de saída são convertidos novamente em texto, preservando as informações posicionais. Por fim, a camada linear transforma as saídas do decodificador em um espaço de maior dimensão e a função *softmax* gera distribuições probabilísticas, implicando previsões com maior precisão e robustez.

Os mecanismos de atenção desempenham um papel essencial no desempenho dos LLMs permitindo identificar relações contextuais entre *tokens*. O mecanismo de Autoatenção (*Self-Attention*) estabelece conexões dentro do mesmo bloco codificador-decodificador, analisando como diferentes partes do texto se relacionam. O mecanismo de Atenção Total (*Full Attention*) é uma implementação direta e completa da autoatenção, avaliando todas as possíveis interações entre *tokens*. Já a Atenção Cruzada (*Cross Attention*) utiliza as representações geradas pelo codificador como consultas para o decodificador, permitindo integrar informações contextuais mais amplas ao processo de geração de texto [Zheng et al., 2025, Lin et al., 2022]. As melhorias no mecanismo de atenção são organizadas em várias abordagens que visam aprimorar sua eficiência e flexibilidade. A Atenção Esparsa (*Sparse Attention*) reduz a complexidade computacional ao introduzir esparsidade no cálculo das interações entre *tokens*, enquanto a Atenção Linearizada (*Linearized Attention*) simplifica os cálculos utilizando mapas de características de *kernel*. Métodos de compressão de Protótipo e Memória (*Prototype and Memory Compression*) otimizam o uso de memória ao reduzir o número de pares de consulta chave-valor. A Autoatenção de Baixa-Patente (*Low-rank Self-Attention*) explora propriedades de baixa patente (*rank*) para tornar o processamento mais eficiente. Já a Atenção com Precedência (*Attention with Prior*) complementa ou substitui a atenção padrão ao incorporar distribuições prévias. Por fim, o mecanismo Multi-Cabeça Aprimorado (*Improved Multi-Head Mechanism*) introduz o conceito de multi-cabeça referindo-se a uma projeção de atenção que foca em diferentes partes da entrada. O uso de múltiplas cabeças permite que o modelo capture diversas representações simultaneamente, enriquecendo a capacidade de aprendizado e aumentando a eficiência computacional.

4.3.1. Aprendizado Baseado em *Transformers*

A arquitetura do modelo *Transformer* é composta por vários componentes interligados que permitem o processamento eficiente de sequências de dados, como textos. Sua principal inovação reside no uso do mecanismo de atenção, que substitui métodos tradicionais sequenciais, possibilitando o processamento paralelo e a captura de relacionamentos de longo alcance no texto. A entrada do modelo começa com a tokenização do texto. Como o *Transformer* não processa os *tokens* sequencialmente, é necessário fornecer informações sobre a posição de cada *token* na sequência e isso é feito por meio de codificações posicionais adicionadas ao *embedding*, normalmente utilizando funções trigonométricas que representam a posição de cada *token* no texto.

A arquitetura principal se divide em duas partes: o **codificador** (*encoder*) e o **decodificador** (*decoder*), quando a tarefa exige geração de texto. O **codificador** é composto por várias camadas idênticas, geralmente entre seis e doze, em que cada camada contém uma atenção auto-regressiva (auto-atenção) e uma rede neural *feedforward* totalmente conectada. A atenção auto-regressiva permite que o modelo avalie a importância de cada *token* em relação aos demais, dando foco às partes mais relevantes do texto para formar uma compreensão contextualizada. Essa camada de atenção calcula as inter-relações entre todos os *tokens* simultaneamente, o que permite ao modelo captar relacionamentos de longo alcance de forma eficaz. Em seguida, a saída passa por uma camada de processamento *feedforward*, que ajusta e combina as informações extraídas. Cada camada é seguida por operações de normalização e mecanismos de *dropout*, auxiliando na estabilidade do treinamento.

Quando o *Transformer* inclui um decodificador seu papel é gerar a saída, como uma tradução ou uma continuação de uma frase. O **decodificador** também é formado por múltiplas camadas, contendo uma atenção mascarada que garante que, na geração de cada *token*, o modelo só leve em consideração os *tokens* anteriores, preservando a causalidade. Há também uma atenção que integra as representações extraídas pelo codificador, permitindo que o decodificador alinhe a entrada com a saída de forma eficiente durante a geração do texto. Essa arquitetura possibilita ao modelo entender o contexto de toda a sequência ao mesmo tempo, ao contrário das abordagens sequenciais, como redes recorrentes, aumentando muito a velocidade de treinamento e aprimorando o desempenho.

Por fim, após o processamento nas camadas de atenção e *feedforward*, as saídas passam por uma camada linear que ajusta a dimensionalidade, seguida por uma função *softmax* que fornece as probabilidades para os próximos *tokens* ou para tarefas de classificação. Dessa forma, a arquitetura do *Transformer* consegue representar o conteúdo de textos de maneira profunda e contextualizada, o que revolucionou o campo do processamento de linguagem natural ao possibilitar o treinamento de modelos cada vez maiores e mais precisos.

A Figura 4.2 representa a arquitetura do modelo *Transformer*, mostrando o fluxo de dados desde a entrada textual até a geração da saída. O texto é inicialmente convertido em vetores por meio de incorporações e codificações posicionais. Esses vetores passam por blocos repetidos de codificador, compostos por mecanismos de auto-atenção e redes *feedforward*, que capturam relações contextuais entre as palavras. Em seguida, o decodificador, também com camadas repetidas, utiliza atenção mascarada e cruzada para gerar

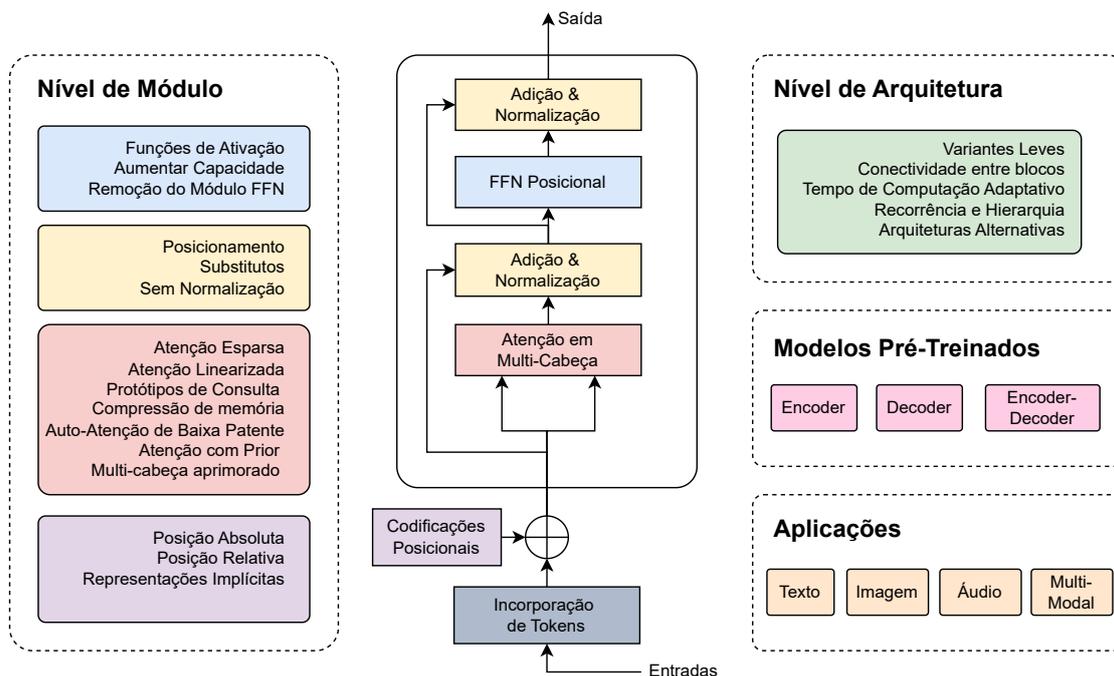


Figura 4.2. Arquitetura básica de um modelo *Transformer* com variações estruturais em nível de módulo e arquitetura, para diferentes aplicações e com tipos de modelos pré-treinados distintos.

a sequência de saída. Por fim, os vetores passam por uma camada linear e uma função *softmax* para prever a próxima palavra com base em probabilidades.

4.3.2. Variações da Arquitetura Principal do Modelo *Transformer*

As variantes principais do modelo *Transformer* são os modelos *Encoder*, modelos *Decoder* e modelos *Encoder-Decoder*, conforme mostra a Figura 4.2 nos modelos pré-treinados. Modelos do tipo **Encoder** utilizam apenas a parte codificadora do *Transformer*, permitindo que as camadas de atenção acessem todas as palavras da frase simultaneamente, o que confere uma atenção bidirecional. Eles são pré-treinados corrompendo sentenças, por exemplo, mascarando palavras aleatórias, e ensinados a reconstruí-las. Esses modelos são ideais para tarefas que exigem compreensão completa da sentença, como classificação de sentenças, reconhecimento de entidades nomeadas e respostas a perguntas de forma extrativa. Representantes dessa família de modelos são o BERT [Aftan e Shah, 2023] e o DistilBERT [Sanh et al., 2020].

Modelos do tipo **Decoder** utilizam apenas a parte decodificadora do *Transformer*, em que a atenção é unidirecional, permitindo que cada palavra acesse apenas as anteriores na sentença. Chamados de modelos auto-regressivos, são pré-treinados para prever a próxima palavra em uma sequência. Eles são mais indicados para tarefas de geração de texto. Representantes dessa classe são os modelos LLaMa [Dubey et al., 2024], Gemma 3 [Jadouli e Amrani, 2025] e DeepSeek V3 [Guo et al., 2025].

Modelos **Encoder-Decoder**, ou de sequência para sequência, utilizam tanto o codificador quanto o decodificador do *Transformer*. O codificador acessa toda a sentença de entrada, enquanto o decodificador só acessa as palavras anteriores a cada etapa. Seu

pré-treinamento varia, mas geralmente envolve a reconstrução de sentenças corrompidas, como no modelo T5 [Raffel et al., 2020], que substitui trechos por um *token* especial e prevê o texto correspondente. Esses modelos são ideais para tarefas que exigem gerar novas sentenças a partir de uma entrada, como sumarização, tradução e respostas generativas a perguntas. Os representantes dessa família são os modelos BART [Lewis et al., 2019] e T5 [Raffel et al., 2020].

4.3.3. Mecanismos de Auto-Atenção

O mecanismo de auto-atenção é fundamental no modelo *Transformer* porque permite que as representações de entrada e saída sejam altamente contextuais e dinâmicas, sem depender de estruturas sequenciais rígidas como redes neurais recorrentes (*Recurrent Neural Network* - RNNs) ou redes neurais convolucionais (*Convolutional Neural Network* - CNNs). Ao usar atenção, o modelo consegue identificar quais partes da sequência são mais relevantes para cada elemento, facilitando a captura de dependências de longo alcance e relações globais entre palavras ou *tokens*. Essa habilidade de focar diferentes partes da entrada simultaneamente torna o processamento mais eficiente e paralelo, além de melhorar a capacidade de modelar relações complexas em tarefas como tradução, resumo e compreensão de texto, alcançando resultados de ponta em desempenho. Ademais, o mecanismo de atenção melhora a escalabilidade do modelo e permite maior paralelização durante o treinamento, o que reduz consideravelmente o tempo necessário para ajustar os parâmetros. Isso, combinado com sua capacidade de modelar dependências de longo alcance de forma mais direta, faz do mecanismo de atenção o componente central que viabiliza a eficácia do *Transformer* e a sua superioridade em muitas tarefas de processamento de linguagem natural. Dentre os mecanismos de atenção utilizados nesse modelo, destacam-se a atenção por produto escalar e a atenção em multi-cabeças.

A **Atenção por Produto Escalar** (*Scaled Dot-Product Attention*) é um mecanismo central no modelo *Transformer* que calcula a atenção com base na similaridade entre vetores de consulta e vetores de chave, associando-os a vetores de valor [Vaswani et al., 2017]. O processo começa com o cálculo do *produto escalar* entre cada consulta e todas as chaves para medir a compatibilidade. Esse valor é então normalizado dividindo por $\sqrt{d_k}$, em que d_k é a dimensão dos vetores de chave. Essa normalização evita que valores muito grandes empurrem a função *softmax* para regiões com gradientes muito pequenos. Após a normalização, aplica-se a função *softmax* para gerar uma distribuição de probabilidade, os pesos, que é usada para calcular uma soma ponderada dos valores. A equação da atenção escalada por produto escalar é dada por

$$\text{Attention}(Q, K, V) = \text{softmax} \left(\frac{QK^T}{\sqrt{d_k}} \right) V,$$

em que Q representa as consultas, K as chaves e V os valores. O termo QK^T mede a compatibilidade entre consultas e chaves, sendo escalado por $\sqrt{d_k}$ para estabilizar os valores. A função *softmax* transforma esses escores em pesos, e a multiplicação final por V gera uma saída ponderada com base nas relações entre os elementos da sequência.

Na prática, as consultas, chaves e valores são organizadas em matrizes $Q \in \mathbb{R}^{n \times d_k}$, $K \in \mathbb{R}^{m \times d_k}$ e $V \in \mathbb{R}^{m \times d_v}$, permitindo computações em paralelo e de forma eficiente. Em

comparação, a *atenção aditiva*, proposta por [Bahdanau et al., 2014], calcula a função de compatibilidade com uma rede neural com uma camada oculta dada por

$$\text{AdditiveAttention}(q, k) = v^\top \tanh(W_1 q + W_2 k).$$

Embora ambas as abordagens tenham complexidade teórica semelhante, a atenção por produto escalar é mais eficiente na prática, pois aproveita multiplicações de matrizes otimizadas. No entanto, sem a escala por $\sqrt{d_k}$, seu desempenho tende a se degradar quando d_k é grande, pois os valores do produto escalar se tornam grandes em magnitude, o que empurra a função *softmax* para regiões com gradientes extremamente pequenos, dificultando o treinamento.

O mecanismo de **Atenção em Multi-Cabeças** (*Multi-Head Attention*) estende a atenção tradicional ao projetar linearmente as consultas, chaves e valores, múltiplas vezes com diferentes matrizes aprendidas [Vaswani et al., 2017]. Em vez de calcular a atenção diretamente com vetores na dimensão total d_{model} , o modelo cria h cabeças de atenção, cada uma operando com vetores de dimensão reduzida d_k e d_v . Para cada cabeça i , aplica-se a atenção escalada por produto escalar da seguinte forma:

$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V) = \text{softmax}\left(\frac{QW_i^Q(KW_i^K)^\top}{\sqrt{d_k}}\right) VW_i^V,$$

em que Q , K e V são projetados por matrizes aprendidas W_i^Q , W_i^K e W_i^V . As pontuações de atenção são calculadas via produto escalar, escaladas por $\sqrt{d_k}$, normalizadas com *softmax* e usadas para ponderar os valores projetados VW_i^V , produzindo a saída da cabeça i . Cada saída $\text{head}_i \in \mathbb{R}^{T \times d_v}$, em que T é o número de posições da sequência, é então concatenada com as demais e projetada de volta para a dimensão original, da seguinte forma:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \dots, \text{head}_h)W^O.$$

Assim, permite-se que o modelo atenda simultaneamente a diferentes subespaços de representação em diferentes posições da entrada, algo que uma única cabeça de atenção não conseguiria capturar de forma eficaz. No *Transformer* original, utilizam-se $h = 8$ cabeças, com $d_k = d_v = \frac{d_{\text{model}}}{h} = 64$, o que mantém o custo computacional semelhante ao da atenção com dimensão total. O uso de múltiplas cabeças melhora significativamente a capacidade do modelo de capturar padrões diversos e complexos, sendo um dos pilares do sucesso dos *Transformers*.

4.3.4. Modelos *Transformers* Eficientes

Os *Transformers* tradicionais apresentam uma complexidade computacional e de memória quadrática ($O(N^2)$) [Tay et al., 2022]. Essa complexidade se origina no mecanismo de autoatenção, que exige que cada *token* calcule sua relação com todos os demais *tokens* em uma sequência. Essa característica torna o uso desses modelos extremamente custosos em cenários que envolvem sequências longas, como grandes documentos, vídeos extensos e imagens de alta resolução.

Para mitigar esses problemas, surgem os chamados *Transformers eficientes*, que utilizam técnicas que reduzem significativamente a complexidade, tornando-os mais escaláveis para diversas aplicações práticas. Exemplos de Transformers Eficientes são citados a seguir [Tay et al., 2022]. O **Linformer** reduz a complexidade quadrática da atenção para uma complexidade linear ($O(N)$) ao utilizar aproximações de baixa patente (*low-rank*). Essa técnica consiste em projetar as matrizes chave e valor em dimensões menores, reduzindo significativamente os custos computacionais, sem comprometer excessivamente a performance. O *Linformer* é especialmente útil em contextos que demandam o processamento de textos extensos. O **Performer** utiliza a técnica de *kernelização* para reformular o mecanismo de autoatenção, eliminando a necessidade explícita de calcular a matriz completa de atenção. Assim, o *Performer* reduz a complexidade para linear ($O(N)$), ideal para aplicações que requerem inferência rápida e uso eficiente de recursos computacionais limitados. O **Longformer** introduz a combinação de atenção local e *tokens* globais específicos, permitindo uma atenção eficiente e quase-linear ($O(N)$). Essa abordagem é particularmente eficaz em aplicações que envolvem processamento e análise de documentos muito longos, como relatórios técnicos, textos jurídicos e artigos científicos. Os **Sparse Transformers** são modelos que utilizam padrões esparsos pré-definidos, como combinações de atenção local e atenção dilatada (*strided attention*). O resultado é uma redução da complexidade para aproximadamente $O(N\sqrt{N})$. Os *Sparse Transformers* são eficazes especialmente na modelagem de imagens e vídeos, capturando simultaneamente detalhes locais e informações contextuais de longo alcance.

Embora muitos mecanismos eficientes de atenção utilizem padrões fixos para reduzir a complexidade computacional, algumas abordagens inovadoras introduzem **padrões aprendidos**. Esses modelos definem, de maneira dinâmica e adaptativa, quais *tokens* irão interagir no cálculo da atenção, oferecendo flexibilidade significativa ao modelo. Dois exemplos notáveis dessa categoria são o *Routing Transformer* e o *Reformer*. O *Routing Transformer* utiliza técnicas de **clusterização dinâmica**, como o algoritmo *k-means*, para organizar *tokens* em grupos similares durante o processamento da sequência. Em cada camada do modelo, os *tokens* são agrupados dinamicamente, permitindo que a atenção seja calculada somente entre *tokens* pertencentes ao mesmo *cluster*. Dessa forma, o *Routing Transformer* consegue reduzir significativamente a complexidade computacional, pois cada *token* não precisa interagir com todos os outros, mas somente com *tokens* semanticamente relacionados. Essa abordagem reduz a complexidade computacional para aproximadamente $O(N\sqrt{N})$, ao mesmo tempo em que oferece capacidade adaptativa de capturar relações contextuais relevantes. Além disso, contribui para a melhoria na eficiência do modelo em sequências muito longas, onde as relações mais importantes são agrupadas automaticamente.

O *Reformer* introduz um mecanismo conhecido como **Locality-Sensitive Hashing (LSH)**, ou *hashing sensível à localidade*. O LSH é uma técnica que permite agrupar *tokens* semelhantes rapidamente em *buckets* (grupos) com alta probabilidade. A atenção é, então, calculada apenas dentro desses grupos, limitando o alcance da atenção e reduzindo drasticamente o custo computacional e o consumo de memória. Especificamente, o mecanismo do *Reformer* consiste em aplicar funções de *hashing* que preservam a proximidade (*localidade*), agrupando automaticamente *tokens* próximos ou similares em representações. Em seguida, a atenção é computada somente dentro desses grupos,

o que reduz a complexidade para $O(N \log N)$. Essa técnica é utilizada de forma eficiente tanto durante o treinamento quanto na inferência, aproveitando a capacidade adaptativa da abordagem. O *Reformer* é particularmente adequado para contextos que possuem alta redundância, como documentos textuais longos ou registros de *logs*.

4.4. Treinamento e Otimização dos Modelos de Linguagem em Larga Escala

O treinamento dos Modelos de Linguagem em Larga Escala pode ser organizado nas etapas de pré-treinamento, ajuste fino e refinamento através do aprendizado por reforço com *feedback* humano. Na etapa de pré-treinamento, o modelo é treinado de forma auto-supervisionada em um grande *corpus* para prever os próximos *tokens* a partir da entrada. As escolhas de arquitetura dos LLMs variam entre modelos *Encoder-Decoder* e modelos somente com *Eecoder*, utilizando diferentes blocos de construção e funções de perda [Naveed et al., 2023]. Na etapa de ajuste fino (*fine-tuning*), existem diferentes métodos para LLMs e cada um visa adaptar o modelo pré-treinado a objetivos específicos. O aprendizado por transferência (*transfer learning*) consiste em refinar o modelo utilizando dados específicos de uma tarefa-alvo. Apesar de LLMs já apresentarem bom desempenho geral, o ajuste com dados de tarefas específicas melhora ainda mais seus resultados em aplicações concretas. O **instruction-tuning** tem como objetivo ensinar o modelo a seguir instruções escritas em linguagem natural. Isso é feito ajustando o modelo com dados compostos por uma instrução e um par entrada-saída correspondente [Chung et al., 2024]. Esse processo é eficaz para melhorar a capacidade de generalização do modelo em tarefas não vistas anteriormente (*zero-shot*), além de torná-lo mais responsivo a *prompts* fornecidos por usuários. Por fim, o ajuste por alinhamento (**alignment-tuning**) trata da segurança e ética no comportamento do modelo. Como LLMs podem gerar respostas incorretas, tendenciosas ou prejudiciais, o ajuste por alinhamento usa *feedback* humano para ensinar o modelo a evitar essas saídas problemáticas [Ouyang et al., 2022]. Isso é feito expondo o modelo a exemplos de respostas inadequadas e ajustando seus parâmetros para torná-lo mais útil, honesto e inofensivo.

O alinhamento de LLMs para o comportamento desejado é frequentemente feito por meio de aprendizado por reforço com *feedback* humano (*Reinforcement Learning with Human Feedback* - RLHF) [Ouyang et al., 2022]. Nesse processo, o modelo é inicialmente ajustado com demonstrações humanas e, em seguida, passa por etapas adicionais de treinamento com modelagem de recompensas e aprendizado por reforço (*Reinforcement Learning* - RL). Na etapa de modelagem de recompensas (*Reward Modeling* - RM), um modelo é treinado para classificar ou ranquear respostas geradas de acordo com preferências humanas. Para isso, pessoas avaliam as respostas com base nos critérios Útil, Honesto e Inofensivo (*Helpful, Honest, Harmless* - HHH), e essas anotações servem de base para ensinar o modelo a reconhecer respostas melhores ou piores. Por fim, na etapa de aprendizado por reforço, o modelo usa as classificações fornecidas pelo modelo de recompensa para ajustar seu comportamento. Utilizando algoritmos como a otimização de política proximal (*Proximal Policy Optimization* - PPO), o modelo é otimizado para gerar respostas preferidas. Esse processo é iterativo e continua até que o modelo atinja desempenho estável e alinhado com os objetivos desejados.

Contudo, a inferência e o treinamento desses modelos exigem alta capacidade de processamento, grande volume de memória e um consumo energético considerável,

Tabela 4.2. Comparativo detalhado dos formatos de precisão numérica em LLMs.

	FP32	FP16	FP8 (E4M3)	FP8 (E5M2)	INT8	INT4
Bits Totais	32	16	8	8	8	4
Bits Sinal	1	1	1	1	N/A	N/A
Bits Expoente	8	5	4	5	N/A	N/A
Bits Mantissa	23	10	3	2	N/A	N/A
Tipo Principal	Ponto Flutuante	Ponto Flutuante	Ponto Flutuante	Ponto Flutuante	Inteiro	Inteiro
Alcance Dinâmico	Muito Amplo	Amplo	Moderado	Amplo	Limitado	Muito Limitado
Precisão	Alta	Moderada	Baixa	Muito Baixa	Baixa	Muito Baixa
Uso de Memória	Muito Alto	Alto	Médio	Médio	Baixo	Muito Baixo

tornando sua implementação direta inviável em ambientes restritos. Para mitigar essas barreiras, estratégias são desenvolvidas em duas frentes [Loukas et al., 2023, Xu et al., 2024]: a primeira visa a otimização da arquitetura e da distribuição da carga, por meio de modelos colaborativos e hierárquicos que fragmentam a demanda computacional entre diferentes componentes do sistema; a segunda foca a minimização do custo operacional intrínseco aos LLMs, através da redução da complexidade de suas matrizes internas, mantendo seu desempenho. Nesse contexto, quantização, poda e destilação de conhecimento são três técnicas principais de compressão de modelos [Ouyang et al., 2022]. Essas técnicas e suas variantes buscam aprimorar a eficiência operacional dos LLMs, garantindo sua viabilidade para uma gama expandida de aplicações ao equilibrar desempenho, consumo de memória e velocidade de inferência.

4.4.1. Quantização

Essencialmente, a quantização é um procedimento de mapeamento que visa reduzir a granularidade da representação numérica dos dados. No contexto de redes neurais, essa técnica se traduz na transformação de pesos e ativações, tipicamente representados em formato de ponto flutuante de alta precisão, para representações com menor largura de *bits*. No sistema binário, a representação de um número de ponto flutuante, fundamentada na base 2, é formalmente expressa como $N = \text{Sinal} \times \text{Mantissa} \times 2^{\text{Expoente}}$. Nesta formalização, o *bit* de **Sinal** é convencionalmente alocado como 1 *bit*, indicando a polaridade do número, sendo 0 para valores positivos e 1 para negativos. O **Expoente** é o componente que estabelece a ordem de magnitude do valor, determinando o deslocamento do ponto binário e, conseqüentemente, ampliando o alcance dinâmico. A alocação de um número maior de *bits* para o expoente resulta diretamente na capacidade de representar um intervalo significativamente mais amplo de valores. Por fim, a **Mantissa** encapsula os dígitos significativos do número, determinando intrinsecamente a precisão ou granularidade da representação numérica. Uma maior quantidade de bits dedicados à mantissa possibilita a representação de números com uma fidelidade superior ao valor real, permitindo a inclusão de um número elevado de casas decimais. A Tabela 4.2 sumariza as características dos principais tipos de formatos de precisão utilizados em LLMs.

Conseqüentemente, a quantização provê uma redução expressiva no tamanho do

modelo e nas demandas computacionais, resultando em uma inferência significativamente mais rápida e menor consumo de memória [Xu et al., 2024]. Três estratégias comumente utilizadas para quantização são mostradas na Figura 4.3: *Round-to-Nearest*, Quantização de Precisão Mista e Quantização de Pesos e Ativações. Na primeira estratégia, os pesos são arredondados uniformemente para INT3, reduzindo a precisão e resultando em alta perplexidade. Na segunda, os pesos mais relevantes são mantidos em FP16, definidos com base nas ativações de entrada, melhorando a perplexidade, mas introduzindo ineficiências de hardware. Na terceira, aplica-se um escalonamento adaptativo antes da quantização, determinado pela magnitude média das ativações por coluna, permitindo preservar a acurácia do modelo sem comprometer a eficiência computacional. Existem duas abordagens principais para implementar a quantização, cada uma com suas características e benefícios específicos.

A primeira abordagem, conhecida como **Treinamento com Reconhecimento de Quantização**² (*Quantization-Aware Training* – QAT) incorpora as operações de quantização diretamente no processo de treinamento, permitindo que o modelo ajuste seus parâmetros considerando restrições de baixa precisão numérica previamente estabelecidas. Essa integração tende a resultar em melhor acurácia após a quantização, uma vez que o modelo aprende a mitigar ativamente os efeitos adversos introduzidos pela representação de menor precisão. Todavia, a aplicação de QAT em modelos de linguagem de grande escala impõe dois desafios significativos, a preservação da generalização e a reprodutibilidade do treinamento original [Liu et al., 2023]. O primeiro diz respeito à manutenção da capacidade *zero-shot*, característica central dos LLMs. Para tanto, é aconselhável que o conjunto de dados empregado no ajuste fino reflita a distribuição observada no pré-treinamento, evitando desvios que prejudiquem o desempenho do modelo quantizado. O segundo desafio está na dificuldade de replicar o processo original de treinamento, devido à escala computacional envolvida e à complexidade dos pipelines utilizados.

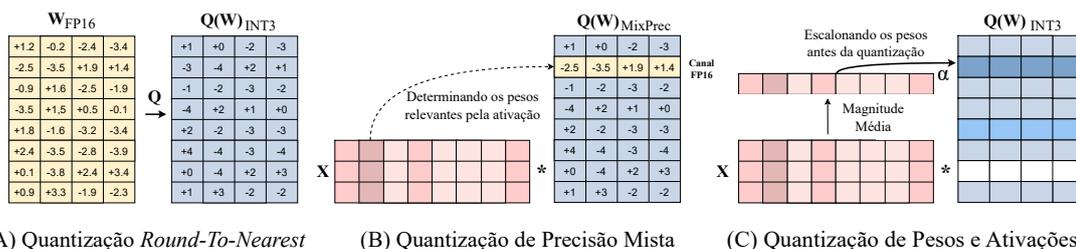


Figura 4.3. Comparação entre três estratégias de quantização para modelos de linguagem. (A) A Quantização Round-To-Nearest reduz a precisão e resulta em alta perplexidade. (B) A Quantização de Precisão Mista melhora significativamente a perplexidade, mas introduz ineficiências de hardware. (C) A Quantização de Pesos e Ativações permite preservar a acurácia do modelo sem comprometer a eficiência computacional, operando exclusivamente com pesos no formato INT3.

A segunda abordagem, denominada **Quantização Pós-Treinamento** (*Post-Training Quantization* - PTQ) consiste na aplicação de técnicas de quantização após o término do treinamento do modelo, sem necessidade de ajuste fino. Por não exigir retropropagação, essa abordagem é computacionalmente mais eficiente que o QAT e apresenta

²Disponível em <https://github.com/facebookresearch/LLM-QAT>.

menor custo de implementação, sendo particularmente atrativa em cenários com recursos limitados. No contexto de PTQ, destacam-se duas variantes principais:

- **Quantização Apenas de Pesos** (*Weight-only Quantization* – WoQ). Nessa configuração, apenas os pesos da rede neural são quantizados, enquanto as ativações permanecem em precisão total, geralmente FP32. Trata-se de uma técnica mais simples e com menor impacto sobre o desempenho do modelo quando as ativações apresentam distribuição estável e baixo dinamismo. Essa abordagem é apropriada para reduzir o uso de memória e simplificar a execução, especialmente em dispositivos com suporte limitado a quantização de ativação.
- **Quantização de Pesos e Ativações** (*Activation-aware Weight Quantization* – AWQ). Introduzida por Lin *et al.*, essa variante otimiza o processo de quantização ao estender sua aplicação tanto a pesos quanto a ativações, elevando a redução na complexidade computacional [Lin *et al.*, 2024]. Como visto na Figura 4.3, a premissa central do AWQ reside na observação empírica de que a importância dos pesos para o desempenho de LLMs é heterogênea. Uma fração mínima, tipicamente entre 0,1% e 1%, dos pesos é considerada relevante. Ao desconsiderar seletivamente a quantização desses pesos críticos, o AWQ mitiga substancialmente a perda de precisão induzida pela quantização. A identificação desses canais de peso salientes é guiada pela distribuição das ativações, em vez da distribuição dos próprios pesos. Canais de peso que correspondem a ativações de maior magnitude são inferidos como mais relevantes, dado seu papel no processamento de características mais representativas. Visando a eficiência em implementações de hardware e para evitar a complexidade da precisão mista, a metodologia AWQ analisa o erro de quantização do peso e propõe que o escalonamento desses canais salientes pode reduzir seu erro de quantização relativo. Esse método é particularmente vantajoso para arquiteturas de hardware, como GPUs e ASICs, devido à otimização da multiplicação de matrizes, operação crucial em computações neurais.

4.4.2. Mecanismos Avançados de Atenção para *Transformers* Eficientes

O mecanismo tradicional de autoatenção nos *Transformers* requer o cálculo das relações entre todos os pares de *tokens* em uma sequência, resultando em complexidade quadrática ($O(N^2)$). Para superar esse desafio, foi proposto o conceito de **atenção esparsa**, que restringe o cálculo da atenção apenas a um subconjunto limitado e predefinido ou aprendido de pares de *tokens*. A atenção esparsa reduz significativamente a complexidade computacional e o consumo de memória. No entanto, envolve relações de compromisso importantes, pois ao limitar a conexão entre *tokens*, pode-se perder parcialmente a capacidade de captar dependências complexas de longo alcance se não implementada cuidadosamente.

Existem diversas variantes de atenção esparsa, sendo as mais conhecidas a Atenção Local, a Atenção Dilatada e a Atenção Global-Local [Tay *et al.*, 2022]. A **Atenção Local** considera apenas um número limitado de *tokens* próximos ao *token* atual, assumindo que informações relevantes estão frequentemente próximas no texto. A **Atenção Dilatada** (*Strided Attention*), em que *tokens* atendem a elementos posicionados em intervalos fixos, permite que o modelo capture informações contextuais mais distantes com

menos cálculos. A **Atenção Global-Local** combina atenção local para contextos próximos e *tokens* específicos que são atendidos globalmente para capturar informações relevantes mais distantes.

Outra classe importante de mecanismos de atenção eficientes é baseada na linearização ou na utilização de *kernels*. Esses métodos evitam o cálculo direto da matriz completa de atenção através de transformações matemáticas inteligentes, tais como o *Linearformer* e o *Performer*, discutidos na Seção 4.3.4. Essas abordagens lineares e baseadas em *kernel* garantem melhor desempenho computacional, embora exijam cuidado especial na definição dos *kernels* e das transformações aplicadas, que podem influenciar diretamente a performance final do modelo.

As técnicas de *downsampling* e *pooling* são também estratégias alternativas para lidar com sequências extensas, reduzindo diretamente a complexidade computacional ao diminuir a resolução ou o número de *tokens* processados. São exemplos dessas técnicas o *Funnel Transformer* e o *Perceiver* [Tay et al., 2022]. A técnica **Funnel Transformer** aplica camadas de *pooling* intercaladas com camadas *Transformers* convencionais para reduzir gradualmente o comprimento da sequência ao longo das camadas intermediárias. À medida que a sequência é comprimida, as camadas subsequentes conseguem operar com custos computacionais menores, ao mesmo tempo em que mantêm as informações essenciais. O *Funnel Transformer* é especialmente útil em tarefas de compreensão textual em que o contexto pode ser condensado sem perda significativa de informação. O **Perceiver** reduz a complexidade processando a sequência original através de um conjunto limitado de *tokens* latentes que atuam como uma representação comprimida da entrada. Esses *tokens* latentes agregam informações globais através de um processo iterativo de atenção. Como resultado, o *Perceiver* pode lidar eficientemente com entradas muito grandes, inclusive em aplicações multimodais que incluem texto, imagens e áudio.

Ambas as abordagens proporcionam uma redução considerável nos custos computacionais e de memória, tornando os *Transformers* viáveis em cenários reais com grandes quantidades de dados ou sequências extensas. Contudo, esses métodos dependem da capacidade dos mecanismos de *pooling* ou *tokens* latentes de reter informações cruciais, o que pode envolver desafios no balanceamento entre compressão e preservação da qualidade da informação.

4.4.3. Poda

A técnica de poda (*pruning*) em redes neurais consiste na remoção seletiva de componentes menos relevantes da rede, como pesos, neurônios ou camadas, com o objetivo de reduzir a complexidade e o tamanho do modelo, mantendo seu desempenho funcional. Essa estratégia é particularmente vantajosa para otimizar grandes modelos, tornando-os mais eficientes em termos computacionais e adequados para execução em dispositivos com recursos limitados. Diferentemente de outras abordagens de otimização para LLMs, a poda atua diretamente na eliminação de parâmetros redundantes ou de baixa relevância [Xu et al., 2024]. Essa técnica pode ser classificada segundo dois critérios principais, o momento da aplicação e o nível de granularidade. Quanto ao momento da aplicação, distinguem-se **poda estática**, realizada após o treinamento do modelo; e a **poda dinâmica**, aplicada durante o processo de treinamento. Com relação ao nível de

granularidade, a literatura identifica três vertentes principais de poda para LLMs [Kaddour et al., 2023], sendo elas:

- **Poda Estruturada** caracterizada pela remoção de blocos inteiros de parâmetros, como camadas, canais ou filtros, favorecendo a otimização de hardware por promover padrões regulares de acesso à memória e computações mais simples. Nesse contexto, Kaddour *et al.* propuseram o LLM-Pruner³, um método de compressão que preserva a capacidade multitarefa dos modelos sem dependência excessiva do conjunto de dados original [Ma et al., 2023]. O método é dividido em três etapas principais: (i) descoberta de estruturas acopladas, na qual são identificados grupos de neurônios interdependentes a serem podados conjuntamente para evitar degradação do desempenho; (ii) estimação de importância, que avalia a relevância desses grupos por meio de informações de gradiente e da Hessiana aproximada, priorizando a remoção dos menos críticos; e (iii) ajuste fino leve, que utiliza uma quantidade reduzida de dados e tempo para refinar o modelo, acelerando substancialmente a etapa de pós-treinamento.
- **Poda Semi-Estruturada** também conhecida como poda contextual, opera em um nível intermediário de granularidade, removendo padrões regulares de pesos, como linhas ou colunas inteiras de matrizes (*block pruning*), ou canais específicos em redes convolucionais. Diferentemente da poda não estruturada, essa técnica produz subarquitecturas consistentes, ou seja, estruturas neurais que mantêm padrões de conectividade regulares e previsíveis. Essa característica facilita a aceleração em hardware convencional (GPUs/TPUs) sem requerer suporte à esparsidade. Seguindo essa premissa, métodos recentes propõem abordagens mais eficazes para explorar a esparsidade N:M em LLMs. A técnica MaskLLM [Fang et al., 2024] introduz uma abordagem de aprendizagem baseada em Gumbel Softmax, capaz de gerar máscaras semi-estruturadas otimizadas via treinamento fim-a-fim, com alto grau de transferibilidade para diferentes domínios e tarefas. Por outro lado, o método PBS2P [Yan et al., 2025] combina poda semi-estruturada com binarização progressiva, propondo uma estratégia passo-a-passo de otimização que reduz o erro total de quantização e poda, inclusive abaixo do erro da binarização isolada.
- **Poda Não Estruturada** prevê a remoção de pesos específicos de uma rede neural com base em critérios como magnitude (Poda por Magnitude) ou impacto estimado na função de perda. Essa técnica resulta em uma matriz de pesos esparsa, com valores zero distribuídos de forma irregular. Essa abordagem é altamente granular, permitindo a preservação de conexões críticas enquanto elimina parâmetros redundantes, o que pode reduzir o tamanho do modelo em até 90% sem perda significativa de desempenho. No entanto, sua eficiência prática depende de *hardware* especializado, já que arquiteturas convencionais como GPUs padrão, não processam zeros de forma otimizada. Nesse contexto, o SparseGPT⁴ destaca-se como um algoritmo eficiente de poda *one-shot* pós-treinamento, voltado especialmente para modelos da família GPT [Frantar e Alistarh, 2023]. Ele alcança entre 50% e 60%

³Disponível em <https://github.com/horseee/LLM-Pruner>.

⁴Disponível em <https://github.com/IST-DASLab/sparsegpt>.

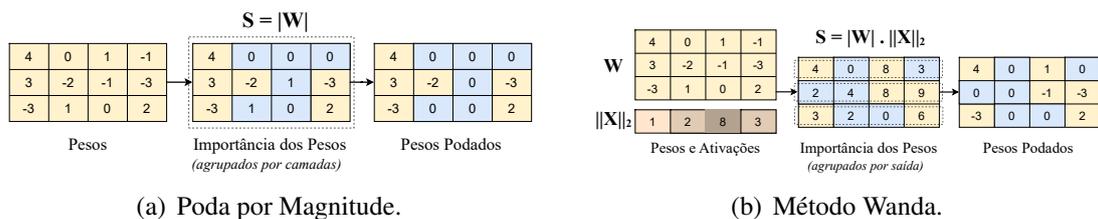


Figura 4.4. Comparação entre duas estratégias de poda não estruturada. Em (a) a Poda por Magnitude remove os pesos com menor valor absoluto, considerando a importância apenas com base na magnitude dos pesos, agrupada por camada. Em (b), o método Wanda considera tanto os pesos quanto a magnitude das ativações de entrada, calculando a importância por saída. Essa abordagem leva em conta a contribuição real dos pesos durante a inferência, resultando em uma poda mais informada e seletiva.

de esparsidade sem necessidade de retreinamento, por meio de uma otimização local baseada em Hessiana aproximada. Na prática, o algoritmo reformula a poda como um problema de regressão esparsa em larga escala, resolvido iterativamente com atualizações parciais nos pesos para compensar os erros introduzidos, permitindo compressão rápida e eficiente. Similarmente, o método Wanda⁵ (*Pruning by Weights and activations*) aplica a poda com base na magnitude dos pesos combinada à norma das ativações de entrada correspondentes [Sun et al., 2023]. Matematicamente esse método é formulado por $S_{ij} = |W_{ij}| \cdot \|X_j\|_2$ onde $(|W_{ij}|)$ representa o valor absoluto do peso na matriz e $(\|X_j\|_2)$ corresponde à norma ℓ_2 (Euclidiana) das ativações associadas, calculada a partir de um pequeno conjunto de dados de calibração. Tal método alcança desempenho semelhante ao do SparseGPT, porém com maior simplicidade uma vez que realiza a poda em uma única passagem direta pelo modelo, sem necessidade de iterações adicionais ou retropropagação. A Figura 4.4 ilustra essas duas estratégias de poda não estruturada.

4.4.4. Destilação de Conhecimento

A destilação de conhecimento (*Knowledge Distillation - KD*) é uma técnica de compressão e transferência de conhecimento em que um modelo de menor porte, denominado aluno, é treinado para reproduzir o comportamento de um modelo maior e mais expressivo, conhecido como professor. Tipicamente, esse processo envolve o uso das distribuições de saída do professor, que representam a probabilidade atribuída a cada classe ou *token*. Essas distribuições são geradas a partir dos *logits*, valores brutos produzidos pelo modelo antes da aplicação da função *softmax*, e refletem o grau de confiança nas previsões. O aluno, então, é otimizado para minimizar a divergência entre suas próprias distribuições e as do professor, aproximando-se do seu comportamento probabilístico [Gou et al., 2021]. Para uma tarefa de classificação multiclasse, em que o modelo estudante geralmente utiliza uma função *softmax* para suas ativações, a função de perda em um processo de KD é composta por duas partes, a perda de destilação e a perda de estudante [Li et al., 2023]. Essa função de perda, representada por \mathcal{L}_{KD} , é definida por

⁵Disponível em <https://github.com/locuslab/wanda>.

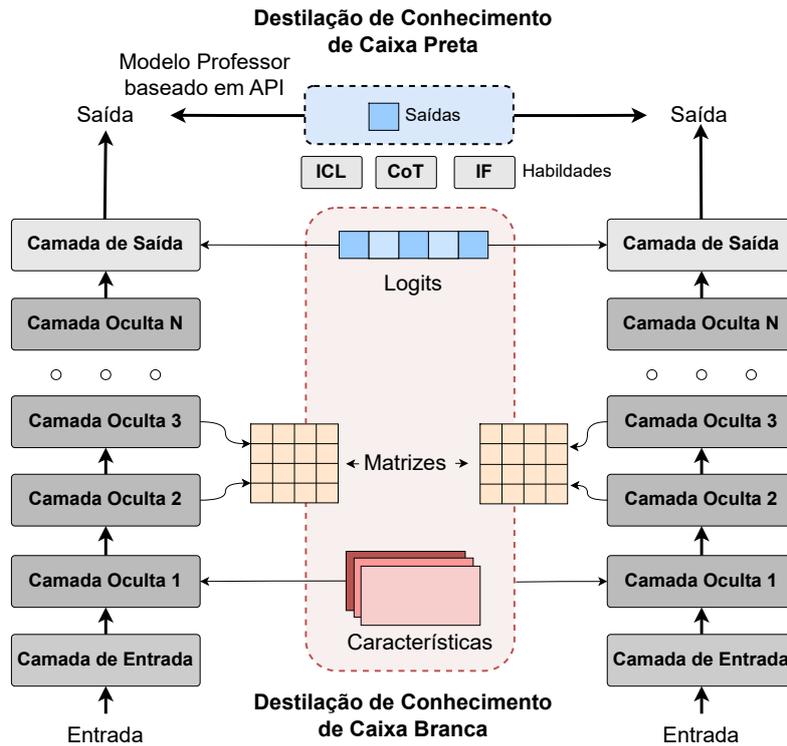


Figura 4.5. Diagrama geral de técnica de destilação de conhecimento, com um modelo professor à esquerda e um modelo estudante à direita. No topo, a destilação de conhecimento de caixa preta utiliza apenas as saídas do professor (logits), obtidas por tarefas como *In-context Learning* (ICL), *Chain-of-Thought* (CoT) e *Instruction Following* (IF). Na base, a destilação de caixa branca envolve o alinhamento de representações internas, como características latentes ou matrizes de atenção, entre os modelos, permitindo uma transferência de conhecimento mais detalhada.

$$\mathcal{L}_{KD} = H(\mathbf{y}, \mathbf{y}_s) + \lambda H(\mathbf{y}_t, \mathbf{y}_s),$$

em que \mathbf{y} representa a verdade fundamental (*ground truth*), os rótulos reais do conjunto de dados de treinamento. As saídas do modelo estudante e do modelo professor são denotadas por \mathbf{y}_s e \mathbf{y}_t respectivamente. H é a função de custo, tipicamente uma entropia cruzada, aplicada tanto à comparação com a verdade fundamental quanto à comparação entre as saídas do professor e do aluno. O parâmetro λ atua como um fator de equilíbrio, ajustando a contribuição de cada uma dessas duas parcelas na perda total. Como ilustrado na Figura 4.5, essa técnica detém duas principais abordagens conforme o nível de acesso ao modelo professor:

- **Destilação de Conhecimento de Caixa Preta** (*Black-box Knowledge Distillation*), abordagem que se fundamenta na ausência de acesso aos parâmetros internos ou representações intermediárias do modelo professor, restringindo o aprendizado do modelo aluno exclusivamente às saídas observáveis, tipicamente distribuições de probabilidade sobre o espaço de saída. Tal configuração é particularmente útil quando o modelo professor é disponibilizado apenas por meio de interfaces restritas, como APIs comerciais, ou quando há incompatibilidade estrutural entre as

arquiteturas de professor e aluno. Nesses casos, o processo de destilação consiste em minimizar a divergência entre as previsões do aluno e as respostas do professor, com base em pares entrada-saída.

- **Destilação de Conhecimento de Caixa Branca** (*White-box Knowledge Distillation*), abordagem que pressupõe total acesso ao funcionamento interno do modelo professor, incluindo seus estados latentes, ativação de camadas intermediárias e pesos. Tal disponibilidade permite que o modelo aluno seja treinado para alinhar não apenas suas previsões finais, mas também suas representações internas às do professor, promovendo uma transferência de conhecimento mais profunda e semântica. Tal alinhamento pode envolver a minimização de distâncias entre *embeddings*, mapas de atenção ou distribuições de ativação, aumentando significativamente a fidelidade do modelo destilado. No entanto, a efetividade dessa abordagem depende de uma compatibilidade arquitetural entre os modelos envolvidos, e sua implementação tende a ser mais complexa e computacionalmente intensiva, exigindo maior controle sobre o *pipeline* de treinamento e acesso integral ao modelo fonte.

Combinando características de ambas as abordagens, o MiniLLM⁶, proposto por Gu *et al.* [Gu *et al.*, 2023], é um método eficaz para destilação de LLMs proprietárias, utilizando a divergência de Kullback-Leibler reversa. Embora formalmente classificado como um método de caixa branca, já que acessa a distribuição de probabilidade do professor, o MiniLLM também incorpora elementos típicos da caixa preta, como a priorização da qualidade das saídas do aluno e estratégias de otimização baseadas em *feedback*. Essa abordagem híbrida permite maior flexibilidade na destilação de modelos restritos, mantendo alto desempenho mesmo com acesso limitado.

4.4.5. Técnicas Avançadas para Otimização de Modelos Embarcados

A otimização eficiente dos Modelos de Linguagem Compactos (*Small Language Models* - SLMs), especialmente aqueles com menos de um bilhão de parâmetros, requer técnicas avançadas para maximizar desempenho sem comprometer a eficiência computacional. Entre as técnicas mais eficazes destacam-se a utilização da atenção multi-consulta (*Multi-Query Attention* - MQA), a atenção por consultas agrupadas (*Grouped-Query Attention* - GQA), compartilhamento de camada e de representações, bem como o uso de arquiteturas profundas e estreitas (*Deep-and-Thin*). Essas abordagens permitem significativa redução no consumo de memória e no tempo de inferência, preservando a precisão e estabilidade dos modelos, fatores críticos especialmente em aplicações com recursos computacionais restritos ou dispositivos móveis [Liu *et al.*, 2024b, Ainslie *et al.*, 2023].

As técnicas de compartilhamento de camada e de representação, **Layer Sharing** e **Embedding Sharing**, são estratégias eficazes para melhorar a eficiência dos modelos de linguagem sem aumentar o custo computacional ou o número total de parâmetros. O **Layer Sharing** consiste em reutilizar os mesmos pesos em múltiplas camadas adjacentes do modelo, permitindo aumentar a profundidade efetiva sem aumentar o tamanho da memória requerida. Isso é especialmente vantajoso em ambientes com restrições severas de memória, pois mantém os pesos na memória cache durante a inferência, minimizando

⁶Disponível em <https://github.com/microsoft/LMOps/tree/main/minillm>.

movimentos custosos de memória. Por outro lado, a técnica de *Embedding Sharing* reutiliza os pesos da camada de representação (*embedding*) de entrada como pesos da camada de projeção de saída, reduzindo consideravelmente o número total de parâmetros dedicados às representações vetoriais das palavras. Essa estratégia maximiza a eficiência no uso dos pesos, sendo particularmente útil em modelos menores, em que as camadas de representação são uma parcela significativa dos parâmetros totais. Ambas as abordagens garantem alto desempenho com menores custos computacionais e de armazenamento [Liu et al., 2024b].

As técnicas de **atenção por múltiplas consultas** (*Multi-Query Attention* - MQA) e **atenção por consultas agrupadas** (*Grouped-Query Attention* - GQA) são estratégias avançadas utilizadas em modelos baseados em *Transformers* para reduzir custos computacionais e o uso de memória durante a inferência, tornando-os ideais para aplicação em dispositivos móveis. A técnica MQA funciona simplificando o mecanismo de atenção tradicional (*Multi-Head Attention* - MHA), que normalmente emprega múltiplas cabeças para consultas, chaves e valores [Shazeer, 2019]. Ao contrário do MHA, o MQA utiliza múltiplas cabeças apenas para as consultas, enquanto as chaves (*keys*) e valores (*values*) são representados por apenas uma única cabeça compartilhada entre todas as consultas. Essa simplificação reduz drasticamente a quantidade de memória necessária para armazenar as representações intermediárias de chaves e valores durante a inferência. Contudo, devido a essa redução significativa, pode ocorrer degradação na qualidade final das respostas, especialmente se não houver adaptação adicional através de um processo conhecido como *uptraining*. O processo de *uptraining* consiste em adaptar um modelo previamente treinado para uma nova configuração arquitetural, realizando um treinamento adicional breve com um subconjunto dos dados originais para que o modelo se ajuste às mudanças introduzidas e recupere ou supere sua capacidade preditiva inicial.

Para contornar a possível perda de qualidade do MQA, surgiu a técnica GQA [Ainslie et al., 2023], que representa um compromisso intermediário entre o MHA e o MQA. No GQA, as cabeças de consulta são divididas em grupos e cada grupo compartilha uma única cabeça de chave e valor. Assim, ao invés de uma única cabeça de chave-valor, como no MQA, ou várias cabeças individuais, como no MHA, o GQA estabelece uma configuração intermediária com mais de uma, mas menos cabeças de chave-valor do que o número total de cabeças de consulta. Essa configuração intermediária mantém boa parte da eficiência computacional e economia de memória do MQA, reduzindo significativamente a necessidade de armazenamento das representações intermediárias, enquanto mantém uma qualidade próxima ao MHA.

As técnicas MQA e GQA se diferenciam essencialmente pelo grau de compartilhamento das representações intermediárias, já que, enquanto o MQA utiliza um compartilhamento completo, sendo uma única cabeça de chave-valor para todas as consultas, o GQA utiliza um compartilhamento parcial em que cabeça de chave-valor é compartilhadas por um grupo de consultas, equilibrando eficiência e qualidade. Para modelos embarcados em dispositivos móveis, tanto o MQA quanto o GQA são altamente vantajosos, pois permitem reduzir a quantidade de dados que precisam ser carregados repetidamente da memória, melhorando o desempenho da inferência e minimizando o consumo energético. O uso dessas técnicas possibilita a implementação prática de modelos de alto desempenho e baixo custo computacional em dispositivos móveis com recursos limitados,

contribuindo para experiências de usuário mais rápidas, eficientes e sustentáveis.

A técnica **Deep-and-Thin** envolve a adoção de modelos com maior profundidade e menor largura (menos parâmetros por camada), contrariando a ideia prevalente de que apenas o número absoluto de parâmetros define o desempenho dos modelos linguísticos. Essa abordagem privilegia um número maior de camadas, porém com menos neurônios em cada camada, aumentando a capacidade do modelo de capturar abstrações complexas e hierárquicas sem aumentar significativamente a quantidade total de parâmetros. Na prática, isso resulta em modelos mais eficientes, com menor custo computacional durante a inferência e maior capacidade de generalização em comparação a modelos tradicionais mais largos e superficiais. Essa estratégia mostrou ganhos significativos em tarefas de raciocínio e compreensão semântico-textual, especialmente em cenários de uso limitado de recursos [Liu et al., 2024b].

4.5. Classes e Características de Modelos de Linguagem em Larga Escala

Um LLM eficaz deve ter quatro características principais: (i) capacidade de captar e modelar o contexto da linguagem natural; (ii) habilidade de gerar texto que semelhante a linguagem humana; (iii) manutenção de contexto e aplicação de conhecimento em domínios específicos; (iv) forte capacidade de seguir instruções, o que é útil para a resolução de problemas e apoio na tomada de decisões [Yao et al., 2024b]. Outro aspecto importante é o gerenciamento de viés presente nos dados. Como os LLMs são treinados em grandes quantidades de texto retiradas da Internet e de outras fontes amplamente disponíveis, há um risco de que esses modelos apresentem vieses presentes nesses dados, o que pode levar a resultados indesejados e até mesmo discriminatórias [Bai et al., 2024].

O **GPT** (*Generative Pre-trained Transformer*) é um modelo de aprendizado profundo baseado na arquitetura *Transformer*, pré-treinado em extensos corpora de texto. Ele pode ser ajustado para uma ampla variedade de tarefas de Processamento de Linguagem Natural, como geração de texto, análise de sentimentos, modelagem de linguagem, tradução automática e classificação. Sua arquitetura representa um avanço significativo em relação a abordagens anteriores baseadas em RNNs e CNNs, ao permitir o processamento eficiente de dependências de longo alcance no texto [Yenduri et al., 2024]. Esse modelo ficou amplamente conhecido em função do ChatGPT desenvolvido pela empresa OpenAI⁷. Esse modelo é capaz de executar diversas tarefas, entre elas *Natural Language Understanding* (NLU), permitindo analisar e interpretar textos e reconhecer entidades e relacionamentos dentro de frases [Mahmud et al., 2025]. A versão inicial, GPT-1, lançada em 2018, empregou uma arquitetura de decodificador *Transformer* com 12 camadas e aproximadamente 110 milhões de parâmetros [Zheng et al., 2021]. Seu treinamento não supervisionado em vastos volumes de texto possibilitou a geração de respostas contextualmente relevantes. Subsequentemente, o GPT-2, introduzido em 2019, expandiu significativamente essa capacidade, com variantes que atingiram até 48 blocos e 1,5 bilhões de parâmetros [Jawahar et al., 2019], resultando em uma notável melhoria na coerência e diversidade da geração textual. O marco seguinte, GPT-3, lançado em 2020, elevou o padrão com 175 bilhões de parâmetros. Essa escala permitiu avanços substanciais em cenários de *few-shot* e *zero-shot learning*, capacitando o modelo a executar tarefas complexas

⁷Disponível em <https://openai.com/>.

com poucos ou nenhum exemplo de treinamento. Em 2023, a OpenAI lançou o GPT-4, um modelo ainda mais avançado e eficiente, com a capacidade de analisar não apenas texto, mas também imagens, caracterizando-o como um modelo multimodal. O GPT-4 tem demonstrado desempenho em níveis humanos em diversas avaliações e exames padronizados, além de gerar textos com maior naturalidade e completude, sublinhando sua escalabilidade e versatilidade multimodal.

O modelo **T5** (*Text-to-Text Transfer Transformer*) é baseado na arquitetura *Transformer encoder-decoder* que reformula todas as tarefas de linguagem natural como problemas de entrada e saída de texto, permitindo uma abordagem unificada para tradução, resumo, classificação, resposta a perguntas e outras tarefas de PLN [Raffel et al., 2020]. A principal inovação desse modelo, consiste em uma abordagem unificada, na qual toda tarefa de linguagem é reformulada como um problema de “texto para texto”, permitindo que funções diversas como responder a perguntas, resumir, classificar e traduzir sejam processadas de forma consistente [Mahmud et al., 2025]. Sua versatilidade o torna uma ferramenta poderosa para uma ampla variedade de aplicações em PLN, consolidando-se como uma das arquiteturas mais abrangentes.

O **BERT** (*Bidirectional Encoder Representations from Transformers*) é um modelo de *Deep Learning* bidirecional baseado na arquitetura *Transformer* que analisa todo o contexto de uma sentença, ou seja, na hora de prever um *token*, o BERT avalia tanto palavras anteriores quanto palavras posteriores, ao contrário de modelos unidirecionais como GPT, que utilizam apenas o histórico [Zheng et al., 2021]. O modelo opera em duas fases sequenciais, pré-treinamento e ajuste fino (*fine-tuning*) [Aftan e Shah, 2023]. A fase de pré-treinamento visa a compreensão da linguagem, enquanto o ajuste fino adapta o modelo para tarefas específicas, tais como análise de sentimentos, resposta a perguntas e sumarização de textos. No pré-treinamento, o BERT emprega duas estratégias, Modelagem de Linguagem Mascarada (*Masked Language Modeling - MLM*) e Previsão da Próxima Sentença (*Next Sentence Prediction - NSP*). A MLM consiste em prever palavras mascaradas em uma sentença com base em seu contexto bidirecional. Por sua vez, a NSP treina o modelo para discernir a relação de coerência entre pares de sentenças, prevendo se uma sentença sucede logicamente a outra [Djeffal et al., 2023]. Essa combinação o torna uma representação extremamente poderosa, capaz de compreender nuances e contextos complexos na linguagem natural.

Um dos principais modelos de código aberto, é o **LLaMA** (*Large Language Model of Meta AI*)⁸ desenvolvido pela empresa Meta. LLaMa é uma família de modelos baseados na arquitetura *Transformer*, projetados para serem uma alternativa capaz de competir com outros modelos comerciais, como o ChatGPT. O LLaMA se destaca por sua arquitetura otimizada, permitindo sua execução em dispositivos com menor capacidade de recursos disponíveis, sem comprometer o desempenho [Touvron et al., 2023]. Essa eficiência faz com que o LLaMA seja uma opção viável para pesquisas que buscam soluções de processamento de linguagem natural, mas que não têm acesso a grandes infraestruturas de computação. Uma das técnicas utilizadas pelo LLaMA, é o uso *embeddings positional* que codificam a posição relativa ou absoluta dos *tokens* na sequência, garantindo que a ordem seja considerada durante o processamento. Já as camadas *feed-forward* subsequentes

⁸Disponível em <https://ai.meta.com/blog/llama-4-multimodal-intelligence/>.

Tabela 4.3. Comparação de diferentes modelos de LLM de código aberto, em termos de provedores e número de parâmetros disponíveis.

Modelo	Versão	Parâmetros	Provedor	Data
GPT [OpenAI et al., 2024]	4	Não divulgado	OpenAI	Março 2024
T5 [Raffel et al., 2020]	1.1	11 B	Google	Janeiro 2023
Gemma [Gemma Team et al., 2025]	3	27 B	Google	Março 2025
LLaMA	4	402 B	Meta AI	Abril 2025
Falcon [Falcon-LLM Team, 2024]	3	10 B	TII	Dezembro 2024
PaLM [Anil et al., 2023]	2	540 B	Google	Maio 2023

utilizam funções de ativação não lineares, como GELU, seguidas por projeções lineares, formando uma estrutura altamente modular e eficiente na captura de relações de longo alcance [Dubey et al., 2024]. Além disso, o modelo incorpora otimizações, como técnicas de *checkpointing* de ativação e atenção eficiente, que reduzem o consumo de memória durante o treinamento, permitindo que modelos de até 400 bilhões de parâmetros sejam treinados de forma mais viável. Essa combinação de melhorias arquiteturais e técnicas de treinamento garante que o LLaMA seja um modelo altamente escalável e versátil.

O modelo **PaLM** (*Pathways Language Model*), desenvolvido pela empresa Google, também utiliza a arquitetura *Transformer*. Esse modelo utiliza a arquitetura **Pathways**, que permite treinar um único modelo em várias tarefas ao mesmo tempo, ao invés de treinar modelos separados para cada tarefa. Esse paralelismo oferece uma grande vantagem em termos de flexibilidade e economia de recursos computacionais, uma vez que reduz o tempo de treinamento. O PaLM foi treinado com um corpus de 780 bilhões de *tokens*, cobrindo uma vasta gama de tarefas linguísticas e de raciocínio, o que contribui para seu desempenho de ponta em tarefas de linguagem natural, geração de código e raciocínio matemático [Chowdhery et al., 2023].

O modelo **Falcon** é uma família de modelos desenvolvida pelo Technology Innovation Institute (TII)⁹, com destaque para o uso de dados exclusivamente extraídos da *web*. Diferente de abordagens anteriores que combinavam *corpora* selecionados manualmente e dados da *web*, o Falcon foi treinado com o conjunto RefinedWeb [Penedo et al., 2023]. Sua arquitetura é baseada no PaLM [Chowdhery et al., 2023], sendo um *transformer* autoregressivo com atenção causal, utilizando ALiBi [Press et al., 2022] para codificação posicional e FlashAttention [Dao et al., 2022] para otimização da atenção.

A Tabela 4.3 apresenta um comparativo entre os diversos modelos LLM disponíveis. Já a Tabela 4.4 mostra a comparação entre os tipos de arquitetura, treinamento e capacidades dos modelos LLM de código aberto.

Os modelos compactos de linguagem são uma classe de modelos projetados especificamente para oferecer desempenho eficiente no processamento local de linguagem natural em ambientes com recursos computacionais limitados, como dispositivos móveis, sistemas embarcados ou aplicações em tempo real [V. et al., 2025]. Em comparação aos LLMs, o número reduzido de parâmetros característico dos SLMs resulta em maior velocidade tanto para a análise de consultas quanto para a geração de respostas [Zhang et al., 2025], o que reduz significativamente seu consumo de memória, processamento e ener-

⁹Disponível em <https://tii.ae/>.

Tabela 4.4. Comparativo entre os tipos de arquitetura, treinamento e capacidades dos modelos LLM.

Modelo	Arquitetura	Treinamento	Capacidades
GPT	<i>Transformer Decoder-only</i> com atenção causal e camadas de <i>self-attention</i>	Pré-treinado via <i>Causal Language Modeling</i> em grandes corpora textuais. Ajuste fino opcional	Geração de texto, sumarização, tradução, QA, diálogo contextual
T5	Arquitetura <i>Encoder-Decoder</i> com abordagem texto-para-texto.	Pré-treinado com <i>span corruption</i> e ajustado supervisionadamente para tarefas específicas.	Tradução, resumo, classificação, QA, geração condicional de texto.
BERT	<i>Transformer Encoder-only</i> com atenção bidirecional.	Pré-treinado com <i>Masked Language Modeling</i> e <i>Next Sentence Prediction</i> .	Classificação, reconhecimento de entidades, QA tipo <i>extractive</i> , análise de sentimento.
LLaMA	<i>Transformer Decoder-only</i> com atenção causal, RoPE, SwiGLU.	Pré-treinado com <i>Causal Language Modeling</i> sobre grande volume de dados públicos.	Geração de texto, resposta a perguntas, diálogo, base para LLMs especializados.
Falcon	<i>Transformer Decoder-only</i> com atenção causal, RoPE e <i>Multiquery Attention</i> .	Pré-treinado com <i>Causal Language Modeling</i> em grandes conjuntos de dados filtrados da <i>web</i> .	Geração de texto eficiente, diálogo, resumo e tarefas gerais de NLP.
PaLM	<i>Transformer Decoder-only</i> com SwiGLU, paralelismo de caminho (Pathways).	Pré-treinado com <i>Causal Language Modeling</i> usando grandes datasets multilíngues e multitarefa.	Raciocínio, QA complexo, geração de código, tradução, generalização multitarefa.

gia, mantendo ainda assim um desempenho satisfatório para tarefas específicas. Modelos compactos são frequentemente otimizados por técnicas como quantização, poda de parâmetros e destilação de conhecimento, permitindo aplicações práticas em situações em que o uso de grandes modelos é inviável devido a restrições técnicas ou econômicas.

O modelo **Phi-3** é uma família de modelos desenvolvida pela Microsoft, cuja versão principal, o phi-3-mini, possui 3,8 bilhões de parâmetros e é baseada na arquitetura de *Transformer* do tipo *decoder*, similar à do LLaMA-2. O treinamento prioriza a qualidade dos dados em vez do aumento de escala. O phi-3 é treinado com 3,3 trilhões de *tokens* compostos por dados da Internet e dados sintéticos gerados por outros LLMs. Adicionalmente, variantes com arquiteturas maiores e capacidades multimodais e multilíngüísticas foram desenvolvidas, como o phi-3.5-MoE, que adota uma estrutura de *Mixture-of-Experts* com 6,6 bilhões de parâmetros ativos, promovendo ganhos significativos em tarefas de raciocínio, matemática e programação [Abdin et al., 2024].

O modelo **DistilBERT** representa uma versão compactada do modelo BERT, desenvolvido com o objetivo de otimizar a eficiência computacional e a velocidade de inferência, sem comprometer a capacidade de compreensão de linguagem natural. Sua arquitetura baseia-se na técnica de destilação de conhecimento, discutida na Seção 4.4.4. DistilBERT preserva aproximadamente 97% do desempenho do BERT original, mas com uma redução de 40% no número de parâmetros e um aumento de 60% na velocidade de inferência [Sanh et al., 2020]. O processo de treinamento integra três funções de perda,

Tabela 4.5. Comparativo entre SLMs em termos de provedores e número de parâmetros.

Modelo	Versão	Parâmetros	Provedor	Data
Phi-3 [Abdin et al., 2024]	mini	3.8 B	Microsoft	Junho 2024
TinyLlama [Zhang et al., 2024]	1.1	1.1 B	StatNLP	Janeiro 2023
DistilBERT [Sanh et al., 2020]	-	66 M	Google	2020
ALBERT [Lan et al., 2020]	large	18 M	Google	2019
Qwen [Yang et al., 2025]	2.5	0.5 B	Alibaba	2025

MLM, destilação e distância do cosseno. Esse modelo é particularmente adequado para uso em dispositivos móveis e ambientes com recursos limitados [Sanh et al., 2020].

O **TinyLlama** é um modelo de linguagem compacto de código aberto, com aproximadamente 1,1 bilhão de parâmetros, desenvolvido a partir da arquitetura do Llama 2. Seu objetivo é ser uma alternativa eficiente e leve a modelos maiores, mantendo o desempenho em tarefas de compreensão, raciocínio e geração de texto. Seu treinamento, que utilizou até 3 trilhões de *tokens* de conjuntos de dados limpos como SlimPajama e StarCoder, incorpora técnicas avançadas como FlashAttention [Zhang et al., 2024]. Além da versão padrão, existem outras especializadas para matemática e programação. Essa combinação de tamanho reduzido e treinamento otimizado torna o TinyLlama ideal para aplicações com recursos computacionais limitados, como em dispositivos móveis e sistemas embarcados [Zhang et al., 2024].

O **ALBERT** é um modelo de linguagem baseado no BERT, desenvolvido para ser mais eficiente em termos de memória e tempo de treinamento, mantendo ou melhorando o desempenho em tarefas de PLN. Esse modelo introduz duas técnicas de redução de parâmetros: (i) Parametrização de incorporação fatorada (*Factorized embedding parameterization*), ao dividir a matriz de vocabulário, pode-se aumentar o tamanho das camadas ocultas sem expandir o número de parâmetros dos *embeddings* de vocabulário; e (ii) o compartilhamento de parâmetros entre camadas (*cross-layer parameter sharing*), que reduz significativamente o número total de parâmetros. O ALBERT substitui a tarefa de predição da próxima sentença (NSP) pela tarefa de predição da ordem das sentenças (SOP), que melhora a modelagem da coerência entre sentenças [Lan et al., 2020].

O **Qwen** é um modelo baseado na arquitetura *Transformer*, desenvolvido para processar sequências de texto com até 1 milhão de *tokens*. Ele foi treinado com dados naturais e sintéticos, utilizando estratégias de extensão progressiva de contexto e ajustes supervisionados em múltiplas etapas, incluindo aprendizado por reforço. Seu funcionamento é otimizado para tarefas de longo contexto, como recuperação de informações, leitura extensiva de documentos e geração de código, por meio de técnicas como atenção esparsa, extrapolação de comprimento e paralelismo de execução [Yang et al., 2025]. As Tabelas 4.5 e 4.6 apresentam um comparativo entre SLMs.

4.6. Avaliação de Modelos de Linguagem em Larga Escala

A avaliação eficaz de modelos de linguagem em larga escala, particularmente no que tange ao impacto de sua compressão, é tratada na literatura por meio de duas abordagens metodológicas principais. A Figura 4.6 ilustra uma taxonomia de métricas e critérios, enquadrados nas duas abordagens distintas: (i) qualidade cognitiva, que reflete a

Tabela 4.6. Comparação entre os tipos de arquitetura, treinamento e capacidades de modelos SLM.

Modelo	Arquitetura	Treinamento	Capacidades
Phi-3	<i>Transformer Decoder</i>	Treinamento supervisionado com dados sintéticos cuidadosamente filtrados (<i>curriculum learning</i>)	Geração de texto, raciocínio lógico, tarefas <i>few-shot</i> com alta performance
TinyLlama	<i>Transformer Decoder</i> (baseado no LLaMA)	Pré-treinamento autoregressivo com <i>corpus</i> diverso e otimizações em memória e velocidade	Geração de texto, compreensão de linguagem, tarefas de NLP em dispositivos de borda
DistilBERT	<i>Transformer Encoder</i> (baseado no BERT)	Distilação do BERT com redução de camadas, mantendo 97% da performance original	Classificação de texto, QA, análise de sentimentos, <i>embeddings</i>
ALBERT	<i>Transformer Encoder</i> com compartilhamento de pesos e <i>embeddings</i> fatorados	Treinamento com <i>autoencoder</i> com predição de frases	Tarefas de NLP como QA, NLI, classificação com alta eficiência em memória
Qwen	<i>Transformer Decoder</i> com <i>Rotary Positional Embeddings</i> e Tokenização baseada em <i>WordPiece</i>	Pré-treinamento autoregressivo em múltiplos domínios, com foco em multilinguismo e código	Geração de texto, respostas em linguagem natural, geração de código, multilinguismo

precisão e a capacidade de generalização do modelo em tarefas não vistas, abrangendo tanto saídas textuais generativas quanto extrativas; e (ii) desempenho operacional, que engloba aspectos fundamentais do custo e da eficiência computacional.

Os paradigmas de *zero-shot*, *few-shot* e suas variantes descrevem diferentes formas de aprendizado em contexto (*in-context learning*) em LLMs, caracterizando como esses modelos podem resolver tarefas sem ajustes nos seus pesos (isto é, sem *fine-tuning*), apenas manipulando a estrutura e o conteúdo do *prompt* de entrada. No aprendizado *zero-shot*, o modelo é condicionado a executar uma tarefa sem receber qualquer exemplo explícito de entrada-saída relacionado à tarefa no *prompt*. A inferência é baseada inteiramente no conhecimento adquirido durante o pré-treinamento em larga escala, explorando a capacidade de generalização do LLM. Esse tipo de raciocínio demonstra como o modelo pode mapear instruções linguísticas abstratas para comportamentos coerentes, mesmo em tarefas inéditas. Por sua vez, o aprendizado *few-shot* envolve a apresentação de um número limitado de exemplos dentro do próprio *prompt*, seguido por uma nova instância de entrada para a qual se espera uma saída correspondente. Essa abordagem permite que o modelo induza padrões latentes a partir dos exemplos e adapte temporariamente seu comportamento, dentro da janela de contexto da inferência. O *one-shot learning* é um caso particular, com um único exemplo disponível. Variantes como o *multi-shot learning* ou *many-shot prompting* fornecem dezenas ou centenas de exemplos no *prompt*, limitados apenas pela capacidade da janela de contexto do modelo. Essas variantes são eficazes em tarefas mais complexas ou sensíveis ao estilo e formato da resposta, beneficiando-se de uma amostragem mais representativa da tarefa-alvo.

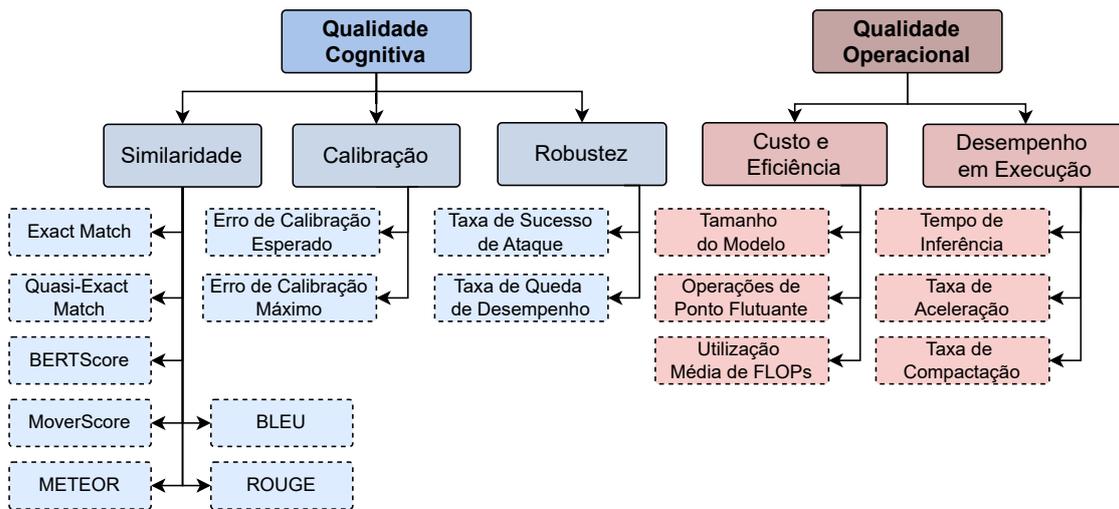


Figura 4.6. Taxonomia de avaliação computacional de modelos de linguagem. Os critérios e métricas de avaliação focam tanto no viés cognitivo, considerando a qualidade da exatidão e capacidade linguística, quanto no viés operacional, considerando a eficiência e desempenho durante a execução.

4.6.1. Critérios e Métricas de Avaliação da Qualidade Cognitiva

A avaliação da qualidade cognitiva de modelos de linguagem representa um desafio multifacetado, impulsionando o desenvolvimento de métricas automatizadas que transcendem a mera correspondência textual. Tradicionalmente, a aferição da qualidade de respostas em PLN tem se baseado extensivamente na análise humana com anotadores, que, embora forneça a verdade fundamental (*ground truth*) de alta fidelidade, é inerentemente subjetiva, demorada e escalável de forma limitada [Chang et al., 2024]. Ademais, a complexidade crescente das saídas de LLMs e a vasta gama de cenários de aplicação tornam a avaliação manual impraticável e suscetível a vieses humanos inconsistentes. Para abordar essa complexidade e mitigar as limitações das abordagens tradicionais, há a proposta de uma estrutura avaliativa automatizada baseada em três critérios chave: a similaridade léxica, a calibração e a robustez. A combinação desses critérios permite uma análise mais abrangente, eficiente e objetiva da capacidade de raciocínio, coerência e adaptabilidade dos LLMs.

O critério de **similaridade léxica** quantifica o grau de sobreposição de termos e sequências de palavras entre o texto gerado por um modelo e um ou mais textos de referência. Essa métrica avalia a correspondência direta de vocabulário e a estrutura frasal, servindo como um indicador da fidelidade ou da relevância do conteúdo gerado em relação a uma base esperada. Sua avaliação pode ser realizada através de diversas métricas, categorizadas tanto por saídas binárias, como *Exact Match* e *Quasi-Exact Match*, quanto por saídas reais, como o *F1-score* e *ROUGE score* [Chang et al., 2024, Yang et al., 2024].

Exact Match (EM) é uma métrica binária que verifica a correspondência literal entre a saída de um modelo e a resposta de referência em tarefas de geração de texto. Em cenários de resposta a perguntas, o EM assume valor 1 se o texto gerado for idêntico à resposta esperada, e 0 caso contrário. **Quasi-Exact Match** (QEM) é uma métrica que estende o conceito de correspondência exata, flexibilizando a exigência de uma equiva-

lência perfeita entre os textos comparados. A QEM considera uma *string* correta mesmo diante de pequenas variações, mediante a aplicação de um pós-processamento leve à saída do modelo. Este processo inclui a normalização para minúsculas, a remoção de espaços em branco e pontuações excessivas, além da exclusão de artigos definidos ou indefinidos. Essa abordagem reconhece que variações triviais em formatação ou na escolha de *stop words* frequentemente não alteram o significado semântico essencial da resposta, conferindo maior robustez à métrica e reduzindo a sensibilidade a detalhes estilísticos irrelevantes para a correção semântica.

F1-score é uma das métricas para avaliação de modelos de classificação, particularmente em problemas binários e cenários desbalanceados em que a classe positiva é rara. Expressa por

$$F_1 = \frac{2 \times \text{Precisão} \times \text{Recall}}{\text{Precisão} + \text{Recall}},$$

a F1-score computa a média harmônica entre outras duas métricas de recuperação de informação, a precisão, dada pela acurácia nas predições positivas, e *recall*, que expressa a capacidade de identificar todos os casos positivos. Matematicamente, a métrica é definida entre $[0, 1]$, em que valores próximos ao limite superior (1) indicam um modelo com alto desempenho, pois precisão e *recall* estão bem equilibrados e maximizados, enquanto valores próximos ao limite inferior (0) representam um modelo ineficaz, com falhas graves na identificação de casos positivos ou excesso de falsos positivos. Em cenários com mais de duas classes, o F1-score pode ser calculado de três formas: (i) *micro-F1* que agrega contribuições de todas as classes, e.g. somatório de verdadeiros positivos, falsos positivos e falsos negativos, sendo robusto a desbalanceamentos e adequado para análise global; (ii) *macro-F1* que calcula a média aritmética dos F1-scores individuais de cada classe, tratando todas igualmente; e (iii) *weighted-F1* que representa a média ponderada pelo número de instâncias de cada classe, combinando sensibilidade a desbalanceamentos com a equidade entre classes.

BLEU (*Bilingual Evaluation Understudy*) é uma métrica amplamente utilizada para avaliar a qualidade de textos gerados por máquinas, especialmente em tarefas como tradução automática e sumarização de texto. Sua metodologia consiste em comparar o texto gerado com um ou mais textos de referência, quantificando a similaridade com base na sobreposição de n -grams, *i.e.* seqüências de palavras. A métrica é calculada por

$$BLEU_{\text{weight}} = BP \cdot \exp \left(\sum_{n=1}^N w_n \log p_n \right),$$

que combina a precisão de n -grams, representada por p_n e ponderada por pesos w_n , com uma penalidade por brevidade (BP) aplicada para evitar traduções excessivamente curtas. O termo exponencial $\exp \left(\sum_{n=1}^N w_n \log p_n \right)$ calcula a média geométrica das precisões para diferentes tamanhos de n -grams, enquanto o BP ajusta o *score* quando a tradução é mais curta que a referência. Quanto mais próximo de 1, mais similar é o texto gerado comparado à tradução humana de referência.

ROUGE (*Recall-Oriented Understudy for Gisting Evaluation*) é um conjunto de

métricas utilizado para quantificar a qualidade da sumarização e tradução automática [Lin, 2004]. Ao contrário do BLEU, que se concentra na precisão, o ROUGE foca principalmente o *recall*, tornando-o particularmente útil para medir quanto do conteúdo do texto de referência é capturado pelo texto gerado. Em sua versão genérica, o ROUGE- N mede a sobreposição de n -grams entre o texto de referência e o texto gerado por um LLM, expresso por

$$ROUGE-N = \frac{\text{Número de } n\text{-gramas sobrepostos}}{\text{Total de } n\text{-gramas no texto de referência}}$$

A variação de N define diferentes granularidades, como a **ROUGE-1**, que avalia a sobreposição de palavras únicas (*unigrams*) e a **ROUGE-2**, que mede a sobreposição de duas palavras consecutivas (*bigrams*). Já o **ROUGE-L** foca na maior subsequência comum, avaliando o quão bem o texto gerado captura a sequência mais longa de palavras compartilhadas, preservando a estrutura do texto de referência. Outras variantes da métrica, como a **ROUGE-W**, atribui maior peso a correspondências contínuas mais longas, ou seja, sequências de palavras mais extensas e correspondentes entre o texto gerado e a referência recebem pontuações mais altas. Paralelamente, a **ROUGE-S** mede a sobreposição de *bigrams* que não são necessariamente consecutivos, capturando correspondências mais flexíveis entre os dois textos.

METEOR (*Metric for Evaluation of Translation with Explicit ORdering*) é uma métrica desenvolvida para avaliar a qualidade de tradução automática e geração de linguagem natural. A métrica melhora o BLEU ao considerar tanto a precisão quanto o *recall* [Banerjee e Lavie, 2005], buscando uma correlação mais fiel ao julgamento humano. Para isso, o METEOR incorpora fatores como sinonímia, *stemming* e a ordem das palavras, aplicando uma penalidade quando esta está incorreta. Essencialmente, é uma métrica baseada em alinhamento de *unigrams* que recompensa traduções que preservam o significado e a fluidez, reconhecendo correspondências de sinônimos e radicais. A métrica é dada por

$$METEOR = (1 - P) \cdot F_{\text{média}},$$

em que $F_{\text{média}}$ é a média harmônica ponderada da precisão e do *recall*, calculada como $F_{\text{média}} = \frac{10 \cdot \text{Precisão} \cdot \text{Recall}}{9 \cdot \text{Precisão} + \text{Recall}}$. O termo P é uma penalidade que varia entre $[0, 1]$, projetada para ajustar o *score* de acordo com a desordem na sequência das palavras. Essa penalidade é calculada com base no número de correspondências e de blocos de palavras não consecutivas entre a tradução gerada e a referência, penalizando traduções com ordem de palavras significativamente diferente da esperada.

Além de métricas baseadas em sobreposição léxica, como BLEU e ROUGE, o **MoverScore**¹⁰ [Zhao et al., 2019] e o **BERTScore**¹¹ [Zhang et al., 2020] representam métricas avançadas que utilizam *embeddings* contextuais para avaliar a similaridade semântica entre textos gerados por LLMs e suas referências. Ambas iniciam com a geração

¹⁰Disponível em <https://github.com/AIPHES/emnlp19-moverscore>.

¹¹Disponível em https://github.com/Tiiiger/bert_score.

de *embeddings* para cada palavra dos textos candidato e de referência, geralmente empregando modelos pré-treinados como o BERT. O MoverScore então se diferencia ao calcular distâncias ponderadas entre pares de *embeddings* e resolver um problema de transporte ótimo (*Earth Mover's Distance* - EMD). Esse problema busca o custo mínimo para mover a massa semântica das palavras do texto candidato para a distribuição de palavras do texto de referência, quantificando o esforço necessário para que os significados do texto gerado se alinhem com os da referência. Em contraste, o BERTScore calcula a similaridade de cosseno entre cada *token* do candidato e seu *token* mais similar na referência, agregando esses *scores* em métricas de precisão, *recall* e *F1-score*. Enquanto o MoverScore se concentra no custo de transporte semântico global, o BERTScore avalia a similaridade contextual de *tokens* individuais e sua agregação.

O critério de **calibração** busca medir a correspondência entre a confiança preditiva do modelo e a acurácia observada de suas saídas. Um modelo é bem calibrado quando a confiança atribuída reflete com precisão a proporção de acertos. Assim, ao atribuir 80% de confiança a um conjunto de previsões, espera-se que 80% estejam corretas. A ausência de calibração, manifestada por superconfiança ou subconfiança, compromete a confiabilidade do modelo, especialmente em aplicações críticas.

Há ainda métricas de erro que podem ser utilizadas na avaliação de LLMs. O **Erro de Calibração Esperado** (*Expected Calibration Error* – ECE) é uma das métricas comumente empregadas para mensurar o desempenho de calibração de modelos preditivos. Seu cálculo envolve a discretização das previsões do modelo em M *bins* de confiança (B_m), em que cada *bin* agrupa previsões com níveis de confiança semelhantes. Para cada *bin*, computa-se a precisão média (*acc*) e a confiança média (*conf*). A métrica é definida como a média ponderada das diferenças absolutas entre a precisão e a confiança em cada *bin*, dada por

$$\text{ECE} = \sum_{m=1}^M \frac{|B_m|}{N} |\text{acc}(B_m) - \text{conf}(B_m)|,$$

em que N é o número total de previsões e $|B_m|$ é o número de previsões no *bin* B_m . Tian *et al.* aplicaram o ECE para analisar a calibração de LLMs otimizados via *Reinforcement Learning from Human Feedback* (RLHF), incluindo modelos como ChatGPT, GPT-4, Claude 1, Claude 2 e Llama2 [Tian et al., 2023].

Complementando a ECE, o **Erro de Calibração Máximo** (*Maximum Calibration Error* – MCE) foca o pior erro de calibração observado. O MCE identifica o maior desvio absoluto entre a precisão e a confiança em qualquer um dos *bins* de confiança. O MCE é particularmente indicado para identificar cenários em que o modelo exibe a maior descalibração, seja por excesso ou falta de confiança, fornecendo *insights* sobre os pontos mais críticos da robustez do modelo. A métrica é dada por

$$\text{MCE} = \max_{m \in \{1, \dots, M\}} |\text{acc}(B_m) - \text{conf}(B_m)|.$$

O critério de **robustez** avalia a capacidade de um modelo manter seu desempenho e funcionalidade sob condições de entrada desafiadoras ou perturbadas. Tal avaliação en-

global a resiliência do modelo a ataques adversariais, eventuais mudanças na distribuição dos dados e a presença de ruído ou variações na entrada. Sendo assim, a robustez é um critério fundamental para garantir a confiabilidade e a segurança dos modelos em ambientes do mundo real, nos quais os dados de entrada podem não ser idênticos aos dados de treinamento ou podem ser manipulados intencionalmente. Dentre as métricas fundamentais baseadas nesse critério, destacam-se a taxa de sucesso de ataque e a taxa de queda de desempenho.

A **Taxa de Sucesso de Ataque** (*Attack Success Rate* – ASR) serve como uma métrica primordial para quantificar a robustez adversarial em LLMs [Wang et al., 2023]. Especificamente, para um conjunto de dados de validação $D = \{(x_i, y_i)\}_{i=1}^N$, onde x_i é a amostra de entrada e y_i é sua verdade fundamental (*ground truth*) correspondente, e dado um método de ataque adversarial A , o ASR mede a proporção de ataques bem-sucedidos. Um ataque é considerado bem-sucedido quando, para uma entrada original x que o modelo substituto f classifica corretamente ($f(x) = y$), o exemplo adversarial gerado $A(x)$ induz o modelo a produzir uma predição incorreta ($f(A(x)) \neq y$). A taxa de sucesso é calculada como

$$ASR = \frac{\sum_{(x,y) \in D} I[f(A(x)) \neq y \text{ e } f(x) = y]}{\sum_{(x,y) \in D} I[f(x) = y]},$$

em que $I[\cdot]$ é a função indicadora, que retorna 1 se a condição dentro dos colchetes for verdadeira e 0 caso contrário. Vale ressaltar que o numerador deve considerar apenas os casos onde o modelo f inicialmente acerta a previsão para a entrada original x , e o ataque é bem-sucedido ao induzir um erro.

A **Taxa de Queda de Desempenho** (*Performance Drop Rate* – PDR) é uma métrica unificada e eficaz para avaliar a robustez do *prompt* em LLMs [Zhu et al., 2024a]. Essa métrica quantifica a degradação relativa do desempenho do modelo após a aplicação de um ataque adversarial ao *prompt*. A degradação é medida comparando o desempenho do modelo com o *prompt* original e com o *prompt* atacado. A PDR é calculada por

$$PDR = 1 - \frac{\sum_{(x,y) \in D} M[f([A(P), x]), y]}{\sum_{(x,y) \in D} M[f([P, x]), y]},$$

em que A representa a função de ataque adversarial aplicada ao *prompt*, P denota a concatenação do *prompt* com a entrada x e M é a função de avaliação de desempenho que varia de acordo com a tarefa específica. Uma PDR elevada indica uma baixa robustez do modelo a alterações no *prompt*.

4.6.2. Critérios e Métricas de Avaliação da Qualidade Operacional

Paralelamente à avaliação da qualidade cognitiva, o desempenho operacional de modelos de linguagem é avaliado com base em métricas que refletem custo computacional, eficiência de execução e viabilidade de implantação. Tais métricas podem ser agrupadas em dois conjuntos principais, (i) aquelas dedicadas a quantificar o custo e eficiência computacional e (ii) aquelas voltadas à medição do desempenho em execução [Chang et al., 2024, Zhu et al., 2024b]. Dentre as métricas focadas em custo e eficiência, a mais

elementar é o **Tamanho do Modelo**, que é expresso pelo número total de parâmetros treináveis e influencia diretamente os requisitos de memória e o custo de armazenamento. Logo, modelos maiores não apenas demandam maior capacidade de armazenamento, mas também exigem mais recursos computacionais tanto para treinamento quanto para inferência. Essa relação entre tamanho e custo computacional pode ser quantificada por meio das **Operações de Ponto Flutuante** (*Floating Point Operations* – FLOPs), que medem a complexidade computacional do modelo, representando o número estimado de operações aritméticas necessárias para processar uma entrada. A redução dos FLOPs é um indicador de maior eficiência computacional, especialmente em modelos comprimidos, pois implica menor carga de processamento. No entanto, a mera contagem de FLOPs não captura totalmente a eficiência prática do modelo. Para avaliar quão bem o hardware disponível é aproveitado, calcula-se a **Utilização Média de FLOPs** (*Mean FLOPs Utilization* - MFU). Essa métrica compara os FLOPs efetivamente utilizados com a capacidade máxima teórica do hardware por ciclo de *clock*, oferecendo uma medida direta da eficiência computacional. Valores elevados de MFU indicam que o modelo está fazendo uso otimizado dos recursos disponíveis, enquanto valores baixos sugerem gargalos ou subutilização.

No segundo grupo de métricas, o **Tempo de Inferência** destaca-se por medir a latência entre a entrada do dado e a geração da resposta pelo modelo. Essa métrica é essencial em aplicações em tempo real e é influenciada pela arquitetura do modelo, paralelismo e otimizações no pipeline de execução. A **Taxa de Aceleração** (T_A) quantifica o ganho de desempenho entre dois modelos distintos, normalmente entre um modelo original e sua versão comprimida. É definida como $T_A = \frac{T_{\text{original}}}{T_{\text{compactado}}}$ em que T_{original} e $T_{\text{compactado}}$ são os tempos médios de inferência de cada modelo. Em contrapartida, a **Taxa de Compactação** (T_C) expressa a economia de espaço após compressão do modelo. Tal métrica é matematicamente definida como $T_C = \frac{S_{\text{original}}}{S_{\text{compactado}}}$, em que S representa o tamanho do modelo, em número de parâmetros ou bytes. Altas taxas de compactação são desejáveis para viabilizar o uso de modelos em dispositivos com recursos limitados.

4.6.3. Arcabouços Públicos de Avaliação

Ao adotar a abordagem baseada no desempenho operacional, é comum a aplicação de *benchmarks* e protocolos de avaliação padronizados. Ferramentas como MMLU, BIG-bench e HELM estabelecem metodologias rigorosas para mensurar o desempenho desses modelos em diferentes cenários, analisando sua capacidade de gerar respostas coerentes, seguras e alinhadas a diretrizes éticas. Esses *benchmarks* estruturam conjuntos de testes abrangentes, permitindo comparações objetivas entre diferentes abordagens e refinamentos sucessivos nos modelos.

O MMLU¹² (*Massive Multitask Language Understanding*), proposto por Hendrycks *et. al.*, compreende um *benchmark* robusto para a avaliação da capacidade de modelos de linguagem na assimilação e aplicação de conhecimento em múltiplos domínios durante o pré-treinamento dos modelos [Hendrycks et al., 2020]. A abordagem central do MMLU reside na avaliação exclusiva em cenários *zero-shot* e *few-shot*, simulando de forma mais fidedigna a avaliação cognitiva humana, que dispensa treinamento

¹²Disponível em <https://github.com/hendrycks/test>.

explícito para cada tarefa específica. Diferentemente dos demais *benchmarks*, o MMLU distingue-se por dois fatores: (i) abrangência multidisciplinar, englobando 57 áreas de conhecimento e (ii) variedade de dificuldade das questões, transitando do nível elementar ao profissional avançado e aferindo tanto o conhecimento geral quanto a habilidade de resolução de problemas. Metodologicamente, o MMLU é construído como um teste massivo e multitarefa, compreendendo questões de múltipla escolha curadas manualmente a partir de fontes abertas, como exames de pós-graduação, licenciamento profissional e materiais de cursos universitários. O *dataset* totaliza 15.908 questões, distribuídas entre um conjunto de desenvolvimento para validação *few-shot* e um conjunto de teste, assegurando um mínimo de 100 exemplos por assunto.

O **BIG-bench**¹³ constitui um *benchmark* colaborativo concebido para uma investigação multifacetada de LLMs existentes [Suzgun et al., 2023]. Composto por 204 tarefas variadas, cada qual contendo múltiplas questões, o BIG-bench permite avaliar aspectos diversos das capacidades dos modelos. Devido ao elevado custo computacional associado à avaliação do *benchmark* completo, uma versão mais leve foi desenvolvida, o **BIG-bench-Lite**, reduzindo o escopo a apenas 24 tarefas. Adicionalmente, o **BIG-bench Hard**¹⁴ (BBH) foi proposto com o objetivo de investigar as tarefas que permanecem insolúveis para os LLMs, selecionando especificamente aquelas nas quais os modelos exibem desempenho inferior ao humano. Dada a elevada dificuldade do BBH, modelos de menor escala frequentemente apresentam desempenho próximo ao aleatório.

O **HELM**¹⁵ (Holistic Evaluation of Language Models) destaca-se como um *benchmark* abrangente por adotar uma abordagem holística e padronizada na avaliação de modelos de linguagem [Liang et al., 2023]. Diferentemente de *benchmarks* tradicionais, que frequentemente se restringem a métricas isoladas como acurácia, o HELM propõe uma estrutura taxonômica que abrange 16 cenários centrais, organizados por tarefa, domínio e idioma, e 7 categorias de métricas, incluindo robustez, justiça, viés e estereótipos, toxicidade, calibração e eficiência. Seu principal diferencial está na capacidade de identificar limitações críticas dos modelos, como quedas acentuadas de desempenho em testes de robustez ou a sub-representação de dialetos não padrão, viabilizando pela primeira vez comparações diretas entre modelos que anteriormente eram avaliados de forma fragmentada.

4.7. Atividade Prática

A atividade prática proposta tem como objetivo principal explorar o processo de quantização de modelos de linguagem e realizar avaliações comparativas entre modelos quantizados e modelos compactos previamente gerados. A atividade é conduzida ao longo de três etapas complementares, descritas e acessíveis em um repositório público no GitHub, disponível em <https://gitlab.com/LabGen/minicurso>. No repositório estão os tutoriais e os códigos para a execução da atividade prática. A **primeira etapa** consiste em aplicar a técnica de Quantização Pós-Treinamento (PTQ) sobre modelos de linguagem já treinados e realizar avaliações comparativas com modelos compactos

¹³Disponível em <https://github.com/google/BIG-bench>.

¹⁴Disponível em <https://github.com/suzgunmirac/BIG-Bench-Hard>.

¹⁵Disponível em <https://crfm.stanford.edu/helm/v0.1.0>.

previamente gerados. Para isso, utiliza-se a biblioteca `PyTorch`¹⁶, que oferece suporte robusto para a quantização dinâmica de modelos. Em particular, o modelo adotado nesta etapa é o DistilBERT, uma versão mais leve do BERT, já ajustada para tarefas de análise de sentimento, disponível via `Hugging Face Transformers`¹⁷. Esse modelo foi treinado sobre o conjunto de dados SST-2 (*Stanford Sentiment Treebank*), amplamente utilizado em *benchmarks* de classificação para análise de sentimentos. A técnica de quantização consiste na conversão dos pesos do modelo de precisão FP32 para o formato INT8, utilizando o `Torch Quantization Toolkit`¹⁸, que permite aplicar quantização dinâmica sobre as camadas lineares do modelo. Durante essa etapa, explora-se a redução de tamanho do modelo, o impacto sobre o tempo de inferência e os efeitos na precisão dos resultados. Embora outras abordagens como o Treinamento com Reconhecimento de Quantização (QAT) possam ser aplicadas, nesta primeira fase o foco está na praticidade e acessibilidade da PTQ. A avaliação é então conduzida com base em tarefas pré-definidas de classificação de texto, utilizando como métrica principal a acurácia, além de indicadores adicionais como o tempo médio de inferência, tamanho final do modelo em disco e a diferença absoluta nas probabilidades preditas entre os modelos original e quantizado. Para o cálculo dessas métricas, utiliza-se a biblioteca `Scikit-learn`¹⁹, bem como ferramentas auxiliares do `PyTorch`.

A **segunda etapa** foca a incorporação dos modelos de linguagem personalizados em dispositivos com restrições de recursos. Tal procedimento é demonstrado utilizando um Raspberry Pi 4²⁰, com ênfase na execução local e eficiente dos modelos utilizando apenas a CPU. Para isso, adota-se a combinação do ambiente Ollama com a interface interativa Open WebUI, ambos executados de forma integrada em um único contêiner Docker. A abordagem adotada consiste na utilização da imagem integrada disponibilizada pelo projeto Open WebUI²¹, que encapsula tanto o servidor Ollama quanto a interface Open WebUI em um único contêiner. Isso permite uma instalação simplificada e automatizada, sem necessidade de configurar múltiplos serviços ou dependências. Como o Raspberry Pi 4 não possui unidade de processamento gráfico dedicada compatível com os requisitos do Ollama, é utilizada a opção de instalação voltada exclusivamente para execução em CPU. Após a instalação, é possível carregar modelos compactos, como o TinyOllama²², disponibilizados pelo próprio repositório do Ollama ou gerados na etapa anterior, e utilizá-los de forma responsiva e acessível mesmo em um ambiente de hardware limitado.

A **terceira etapa** prevê a personalização e utilização dos modelos otimizados nas etapas anteriores, via OpenWebUI no Raspberry Pi. Essa plataforma oferece uma interface amigável para editar e testar fluxo de processamento dos modelos quantizados e não quantizados inseridos na plataforma. Internamente, o Open WebUI oferece recursos para o desenvolvimento de elementos de processamento, o que permite a personalização

¹⁶Disponível em <https://pytorch.org/>.

¹⁷Disponível em <https://huggingface.co/docs/transformers/index>.

¹⁸Disponível em <https://docs.nvidia.com/deeplearning/tensorrt/archives/tensorrt-861/pytorch-quantization-toolkit/docs/index.html>.

¹⁹Disponível em <https://scikit-learn.org/stable/>.

²⁰Disponível em <https://www.raspberrypi.com/>.

²¹Disponível em <https://openwebui.com/>.

²²Disponível em <https://github.com/jzhang38/TinyLlama>.

e a extensão das funcionalidades da plataforma. Esses elementos incluem: (i) *filter*, elemento que permite filtrar e pré-processar dados de entrada para os modelos; (ii) *pipeline*, elemento que facilita a criação de fluxos de trabalho complexos, combinando diferentes elementos de processamento como *filters*. Quando o *pipeline* de filtro é ativado, a mensagem de entrada (*inlet*) é processada antes do LLM e a saída (*outlet*) após, personalizando a interação; (iii) *tool*, elemento que possibilita a integração de ferramentas externas para enriquecer as capacidades dos modelos; (iv) *function*, elemento que permite a definição de funções personalizadas para manipulação de dados e execução de tarefas específicas.

Com essa sequência de etapas, a atividade prática permite compreender os fundamentos da quantização e sua aplicação em modelos de linguagem e também demonstra, de forma concreta, a viabilidade da personalização e execução eficiente desses modelos em plataformas embarcadas com recursos limitados.

4.8. Desafios e Tendências de Pesquisa

Os LLMs transformaram significativamente o processamento de linguagem natural e a inteligência artificial generativa. No entanto, essa transformação trouxe uma série de desafios técnicos e éticos. Um dos principais desafios técnicos está relacionado ao elevado custo computacional e energético envolvido tanto no treinamento quanto na inferência dos modelos. LLMs exigem grande quantidade de memória e capacidade de processamento, o que restringe sua adoção ampla. A escalabilidade de LLMs tem sido acompanhada de custos ambientais e operacionais crescentes, com grandes modelos consumindo megawatts-hora [Argerich e Patiño-Martínez, 2024, de Vries, 2023]. A otimização desses modelos é essencial para reduzir o custo operacional e permitir sua escalabilidade. Nesse contexto, técnicas de quantização, modelos esparsos, mistura de especialistas (*Mixture of Experts - MoE*), poda, destilação, e uso de hardware especializado (e.g., FPGAs, TPUs, ACAPs), vêm ganhando destaque como abordagens promissoras para melhorar a eficiência computacional de LLMs e consequente escalabilidade.

A inferência de LLMs é limitada por sua natureza autoregressiva e pelo elevado consumo computacional. Reduzir a latência e aumentar a eficiência é essencial para aplicações em tempo real [Huang et al., 2024]. Estratégias como a utilização eficiente de parâmetros, por meio da ativação seletiva de subconjuntos do modelo, permitem acelerar a geração de respostas com menor custo computacional. Para tal, é necessário estabelecer critérios bem definidos que permitam encerrar antecipadamente o processo de decodificação, mantendo a qualidade da saída ao mesmo tempo em que economiza recursos computacionais [Huang et al., 2024]. A esparsidade contextual também possibilita ignorar pesos irrelevantes dinamicamente durante a inferência, sem comprometer a acurácia. A Mistura de Especialistas (*Mixture of Experts - MoE*) expande esse princípio, ativando apenas os submodelos especializados mais relevantes para cada entrada, o que permite escalar a capacidade do modelo com eficiência. A otimização da memória cache de chaves e valores em LLMs busca reduzir os custos associados ao armazenamento e gerenciamento de chaves e valores previamente calculados, especialmente em tarefas complexas como cadeia de pensamento e recuperação de informações [Huang et al., 2024].

A quantização de LLMs lida com o desafio adicional de manter a confiabilidade estatística dos resultados, mesmo em ambientes computacionais restritos. Um obstáculo

nesse sentido é o desenvolvimento de algoritmos de inferência escaláveis e confiáveis, superando limitações de métodos como Monte Carlo via Cadeias de Markov (MCMC), que oferecem boa precisão, mas sofrem em escalabilidade. Técnicas recentes como subamostragem de dados e paralelismo assíncrono buscam mitigar esses problemas, garantindo eficiência mesmo em cenários complexos²³.

Outro desafio é integrar computação aproximada para melhorar a eficiência energética, equilibrando precisão e economia de energia, com avanços em modelos de programação, ferramentas de teste, suporte sistêmico e arquiteturas de hardware. Baseado na ideia de que nem todas as tarefas computacionais requerem precisão absoluta, há projetos que exploram o paradigma de computação aproximada para reduzir o consumo de energia e recursos computacionais²⁴. O projeto SHF, desenvolvido pela Washington University e apoiado pela NSF/EUA, é inspirado em um modelo cognitivo que afirma que o cérebro humano realiza raciocínio rápido e simples para a maioria das tarefas e executa um processamento detalhado apenas quando necessário. O projeto desenvolveu técnicas avançadas de compilação automática para modelos quantizados, com suporte a tensores esparsos e seleção adaptativa de bibliotecas otimizadas, que reduzem o consumo energético de LLMs em execução em múltiplas plataformas. Essas inovações foram integradas ao Apache TVM, um dos compiladores mais amplamente utilizados atualmente para aprendizado de máquina, o qual originou a *startup* OctoML. A otimização permitiu que modelos de linguagem sejam compilados e otimizados para execução eficiente em dispositivos com restrições energéticas, como FPGAs e NPUs, bem como em nuvens escaláveis e energeticamente conscientes.

No contexto científico, o projeto A3D3 desenvolve algoritmos personalizados de IA integrados a plataformas heterogêneas, como GPUs e FPGAs, para acelerar o processamento em tempo real em áreas como física de altas energias, astrofísica e neurociência²⁵. Em física de partículas, técnicas como Redes Neurais de Grafos (*Graph Neural Networks* - GNNs) de baixa latência em FPGAs foram exploradas para rastreamento de trajetórias em experimentos como o ProtoDUNE [Cai et al., 2023]. A implementação de inferência semântica em tempo real com `hls4ml` possibilitou aplicações autônomas de IA embarcada em física e veículos autônomos [Ghielmetti et al., 2022, Khoda et al., 2023]. Na área de ondas gravitacionais, ecossistemas de software e inferência acelerada por hardware permitiram implantar aprendizado profundo com alta eficiência para detectar sinais astrofísicos em tempo real [Gunny et al., 2022]. Em colaboração com o experimento ATLAS, foi demonstrada a viabilidade de avaliação de inferência de aprendizado de máquina em FPGAs no contexto do sistema de gatilho de eventos [Jiang et al., 2024]. O projeto contribuiu também para o avanço de algoritmos interpretáveis e escaláveis de aprendizado em grafos, como os baseados em atenção estocástica e representação temporal de redes, com aplicações em múltiplos domínios científicos [Miao et al., 2022].

Além das técnicas de otimização, surgem desafios na gestão eficiente de cargas de trabalho complexas de aprendizado profundo em ambientes de centros de dados, com foco no agendamento de tarefas dinâmicas e heterogêneas. Nesse cenário, há projetos cujo ob-

²³Disponível em https://www.nsf.gov/awardsearch/showAward?AWD_ID=2046760.

²⁴Disponível em https://www.nsf.gov/awardsearch/showAward?AWD_ID=1518703.

²⁵Disponível em https://www.nsf.gov/awardsearch/showAward?AWD_ID=2117997.

jetivo central é o desenvolvimento de um sistema de software intuitivo e eficiente, capaz de realizar, com apenas um clique, a configuração e execução de treinamentos distribuídos de modelos de aprendizado de máquina e aprendizado profundo, inclusive modelos fundacionais de larga escala²⁶. A proposta foca no desafio de gerenciar automaticamente recursos computacionais em ambientes multiusuário e heterogêneos, otimizando desempenho, eficiência estatística e uso de infraestrutura. O projeto desenvolveu dois sistemas: Alpa, para gerenciar o treinamento e a inferência de LLMs, e Pollux, para otimizar o agendamento de tarefas em *clusters* de GPUs. O sistema Alpa constrói planos hierárquicos de execução paralela e aplica passes de compilação para gerar estratégias eficientes de execução, tanto em termos de computação quanto de memória. Já o sistema Pollux utiliza o conceito de *goodput* para redistribuir recursos dinamicamente, promovendo uma utilização justa e otimizada do *cluster*. O projeto também desenvolveu o AMP, um arcabouço que automatiza a identificação e a aplicação de estratégias eficazes de paralelismo de modelo, selecionando a estratégia mais adequada utilizando um modelo de custo sensível à diversidade das estruturas dos modelos e às especificações do *cluster* de computação. O arcabouço se destaca na gestão de modelos complexos com camadas estruturalmente irregulares e em *clusters* que combinam diferentes gerações de GPUs. Outra ferramenta desenvolvida no projeto é a Redco, que simplifica a configuração de treinamento distribuído para LLMs, reduzindo a necessidade de conhecimento técnico aprofundado. Entre as principais funcionalidades do Redco estão regras pré-definidas para a distribuição dos modelos e funções personalizáveis que otimizam os fluxos de trabalho em aprendizado de máquina. Também foram desenvolvidos algoritmos que aumentam a velocidade e a eficiência tanto no pré-treinamento quanto no ajuste fino de modelos fundacionais, bem como softwares voltados para desafios específicos, como inferência de modelo privado e problemas complexos de meta-aprendizado.

Questões de privacidade e uso ético de dados emergem como barreiras críticas [Jiao et al., 2025]. Os LLMs frequentemente são treinados com informações sensíveis, muitas vezes sem consentimento explícito. Durante a inferência, o modelo também pode ter acesso a dados sensíveis fornecidos pelo usuário. Contudo, o acesso a esses dados, seja durante o treinamento ou durante a inferência, fere a privacidade do proprietário dos dados. Existe risco de vazamento de dados, uma vez que os modelos podem aprender e repetir informações sensíveis presentes nos dados de treinamento, como nomes, e-mails e números de documentos. Os dados fornecidos durante o uso do modelo para inferência podem ser armazenados pelo proprietário do modelo para usos diversos, sendo possível haver vazamento do histórico de interações com o modelo. Casos como o da plataforma DeepSeek ilustram esses riscos²⁷. Em janeiro de 2025, um banco de dados ligado à plataforma DeepSeek estava publicamente acessível, não sendo necessária autenticação para acessar as informações. O banco de dados continha histórico de bate-papo dos usuários, dados de *back-end* e informações confidenciais, incluindo fluxos de registros, segredos de API e detalhes operacionais da plataforma. O uso de conteúdos protegidos por direitos autorais no treinamento dos modelos pode gerar processos judiciais. No início de 2025, uma tendência utilizando o gerador de imagens da OpenAI surgiu nas redes sociais

²⁶Disponível em https://www.nsf.gov/awardsearch/showAward?AWD_ID=2008248.

²⁷Disponível em <https://www.wiz.io/blog/wiz-research-uncovers-exposed-deepseek-database-leak>.

para criar imagens com traços característicos do Studio Ghibli²⁸, levando a uma discussão intensa sobre questões éticas do uso de propriedade intelectual para treinamento de LLMs. Apesar de não se ter conhecimento sobre a abertura de processo judicial do Studio Ghibli contra a OpenAI, outros casos que infringem direitos de uso já resultarem em processos judiciais^{29,30}. Soluções emergentes para proteção de dados sensíveis incluem o treinamento com privacidade diferencial e o uso de técnicas de privacidade computacional como aprendizado federado. A implementação de filtros e guarda-corpos (*guardrail*) durante o refinamento e a geração das respostas são medidas que podem ser tomadas para mitigar o problema da privacidade.

Para além de preocupações com a privacidade e direitos autorais, a implementação maciça dos LLMs pode exacerbar desigualdades preexistentes e gerar impactos sistêmicos indesejados [Weidinger et al., 2022]. A concentração hegemônica do desenvolvimento de LLMs em um número restrito de corporações e nações tende a acentuar a assimetria tecnológica global, resultando em uma dependência crítica de infraestruturas computacionais e cognitivas centralizadas. A adoção acelerada de LLMs por entidades públicas e privadas, frequentemente desprovida de avaliações de impacto abrangentes ou de mecanismos robustos de responsabilização, arrisca comprometer direitos individuais e coletivos, particularmente em contextos caracterizados por arcabouços regulatórios incipientes. Adicionalmente, identificam-se riscos intrínsecos relacionados à obsolescência programada de competências humanas, com potenciais repercussões adversas sobre os mercados de trabalho, processos educacionais e a autonomia dos indivíduos diante de sistemas automatizados opacos [Deranty e Corbin, 2024]. A aplicação indiscriminada de LLMs como substituto do julgamento humano, mormente em domínios sensíveis, pode culminar na desumanização de processos decisórios e na erosão de mecanismos tradicionais de validação, revisão e contestação. Na ausência de estruturas de governança transparentes e explicitamente orientadas pelo interesse público, a incorporação ubíqua desses modelos corre o risco de intensificar dinâmicas de exclusão, vigilância e controle algorítmico, transmutando desafios técnicos em questões críticas de justiça social."

Outro risco à utilização de modelos de linguagem de grande escala e genéricos é injeção de *prompts* (*prompt injection*). A injeção de *prompt* é considerada uma das principais vulnerabilidades de segurança relacionadas a modelos de linguagem de grande escala, sendo classificada pela OWASP como a ameaça mais crítica nesse contexto³¹. Esse tipo de ataque ocorre quando entradas enviadas ao modelo são manipuladas com instruções maliciosas capazes de burlar a intenção original da aplicação. O problema se agrava em sistemas que utilizam técnicas de *retrieval-augmented generation* (RAG), nos quais a separação entre dados e instruções pode se tornar nebulosa. Para reduzir os riscos associados ao uso de LLMs, sobretudo no que se refere à segurança e ao uso indevido, filtros e guarda-corpos (*guardrail*) são essenciais. Entre elas estão os filtros

²⁸Disponível em <https://www.forbes.com/sites/danidiplacido/2025/03/27/the-ai-generated-studio-ghibli-trend-explained/>

²⁹Disponível em <https://www.theguardian.com/books/2025/apr/04/us-authors-copyright-lawsuits-against-openai-and-microsoft-combined-in-new-york-with-newspaper-actions>

³⁰Disponível em <https://originality.ai/blog/openai-chatgpt-lawsuit-list>.

³¹Disponível em <https://owasp.org/www-project-top-10-for-large-language-model-applications/assets/PDF/OWASP-Top-10-for-LLMs-v2025.pdf>.

de segurança implementados em modelos como DALL-E 2 e Midjourney, que impedem a geração de conteúdos impróprios (*Not Safe for Work* - NSFW); modelos especializados em segurança, como o LlamaGuard da Meta, capazes de classificar entradas e saídas como seguras ou não; detectores específicos para conteúdos visuais, como os *CLIP-based NSFW detectors*; e técnicas de *watermarking*, voltadas à rastreabilidade e autenticação de mídias geradas por IA. Apesar desses esforços, os mecanismos de proteção ainda apresentam fragilidades, já que muitos ainda são susceptíveis a ataques de *prompt injection* mais elaborados, o que torna a evolução contínua dessas defesas um tema central nas pesquisas atuais sobre segurança em LLMs [Divakaran e Peddinti, 2024].

Paralelamente, na esfera social, os LLMs podem amplificar estereótipos e refletir desigualdades sociais encontradas nos dados utilizados para treinamento. A amplificação de vieses sociais presentes nos dados resulta, por exemplo, na geração de conteúdo discriminatório ou ofensivo, e no reforço de assimetrias de gênero, raça e classe [Gross, 2023, Kotek et al., 2023]. É preocupante a possibilidade de disseminação de desinformação e informações falsas, dada a capacidade dos LLMs de gerar conteúdos convincentes, porém potencialmente falsos ou enganosos. A disseminação desse tipo de informação pode ser feita para manipular a opinião pública ou criar materiais fraudulentos, o que pode ter consequências graves para a saúde pública, a democracia e a harmonia social [Henderson et al., 2018, Cortiz e Zubiaga, 2021]. A mitigação dos efeitos do viés social inserido nos LLMs requer a utilização de técnicas de desenviesamento no treinamento e inferência, avaliações contínuas com métricas de justiça e explicabilidade e uma curadoria ativa dos conjuntos de dados. Nesse sentido, Xue *et al.* criaram um conjunto de dados, denominado OccuQuest [Xue et al., 2023] a fim de refinar LLMs para remover o enviesamento do modelo. Os autores focam no viés social relacionado a profissões, que limita a capacidade dos LLMs ajustados por instruções de gerar respostas úteis para perguntas profissionais específicas. Aplicando esse conjunto de dados ao LLaMa, o modelo obtido superou significativamente variantes do LLaMA de última geração (como Vicuna, Tulu e WizardLM) em perguntas profissionais, segundo avaliações do GPT-4 e avaliações humanas [Xue et al., 2023].

A incorporação de LLMs em sistemas embarcados e infraestruturas críticas traz preocupações adicionais [Jiao et al., 2025]. Diagnósticos médicos, sentenças judiciais, controle e gerenciamento de sistemas em áreas essenciais, como energia, água, gás e telecomunicações, realizados de forma autônoma por LLMs são tarefas potencialmente arriscadas, pois envolvem decisões de alto impacto que exigem precisão, responsabilidade legal e consideração ética, características que os LLMs, por sua natureza estatística e opaca, não são capazes de garantir plenamente. Há ainda o risco de que os usuários desenvolvam uma dependência excessiva ou confiança acrítica nas recomendações geradas automaticamente, o que pode comprometer sua autonomia decisória e contribuir para sobrecarga cognitiva. Modelos interpretáveis e explicáveis, além de interfaces humano-circuito (*human-in-the-loop*), são fundamentais para mitigar esses riscos.

A fim de mitigar o uso indiscriminado dos LLMs, a regulação de LLMs está em rápida evolução globalmente, com legislações como: AI Act, na União Europeia, Executive Order 14110, nos Estados Unidos, e o Marco Legal da IA, no Brasil. AI Act³²

³²Disponível em <https://artificialintelligenceact.eu/>.

classifica sistemas de IA por risco e impõe obrigações específicas para modelos fundacionais. Executive Order 14110³³ foca segurança, privacidade e inovação responsável. O Marco Legal da IA³⁴ está em discussão no Brasil e busca equilibrar inovação com direitos fundamentais. Essas regulações pressionam por transparência, rastreabilidade e controle de uso, exigindo que desenvolvedores e operadores se adaptem a requisitos de conformidade, governança e auditoria algorítmica.

Há também iniciativas de auto-regulação desenvolvidas por empresas do setor. O *Frontier Model Forum*³⁵ e a *Coalition for Content Provenance and Authenticity (C2PA)*³⁶ são duas iniciativas interempresariais que buscam promover a segurança, a transparência e a confiança no uso de tecnologias baseadas em inteligência artificial e na produção de conteúdos digitais [Divakaran e Peddinti, 2024]. O *Frontier Model Forum* foi criado em 2023 pelas empresas Microsoft, Google, OpenAI e Anthropic com o objetivo de fomentar o desenvolvimento seguro e responsável de modelos de linguagem de fronteira, modelos de IA altamente avançados e com amplo poder de generalização. Entre suas metas estão o avanço da pesquisa em segurança para IA, o estabelecimento de melhores práticas para desenvolvimento e implantação desses modelos e o incentivo à colaboração entre setor privado, governos, academia e sociedade civil. A iniciativa também visa apoiar aplicações de IA voltadas à resolução de grandes desafios sociais, como mudanças climáticas, saúde e segurança digital. Já a C2PA reúne empresas como Adobe, Microsoft, BBC, Intel e Tru-eptic, com foco na criação de padrões técnicos para garantir a rastreabilidade e autenticidade de conteúdos digitais. Essa coalizão atua no desenvolvimento de uma infraestrutura padronizada que incorpora metadados criptograficamente protegidos às imagens, vídeos e documentos digitais, permitindo identificar quem criou o conteúdo, quando, onde e com quais ferramentas. A rastreabilidade é essencial no combate à desinformação e ao uso indevido de *deepfakes*, fornecendo mecanismos confiáveis de verificação de autenticidade para o público, plataformas e autoridades.

O desenvolvimento de LLMs exige recursos computacionais e conhecimentos multidisciplinares que extrapolam as capacidades isoladas de empresas ou instituições acadêmicas. Nesse cenário, parcerias estratégicas têm se mostrado cruciais. Iniciativas como o IBM-Illinois Discovery Accelerator³⁷ foi criada para ampliar o acesso à educação em tecnologia e ao desenvolvimento de habilidades, a fim de impulsionar avanços em áreas emergentes da tecnologia, incluindo computação em nuvem híbrida e inteligência artificial, computação quântica, descoberta acelerada de novos materiais e sustentabilidade, com o objetivo de acelerar a descoberta de soluções para desafios globais complexos. Iniciativas de *benchmarking* aberto e conjuntos de dados colaborativos promovem transparência e reprodutibilidade [Liu et al., 2024a].

As tendências de pesquisa em LLMs refletem esforços para superar desafios técni-

³³Disponível em <https://www.govinfo.gov/app/details/CFR-2024-title3-vol1/CFR-2024-title3-vol1-eo14110/summary>.

³⁴Disponível em <https://www25.senado.leg.br/web/atividade/materias/-/materia/157233>.

³⁵Disponível em <https://www.frontiermodelforum.org/>.

³⁶Disponível em <https://c2pa.org/>.

³⁷Disponível em <https://discoveryacceleratorinstitute.grainger.illinois.edu/>

cos, energéticos e éticos da sua adoção em larga escala. Destacam-se avanços em eficiência computacional e energética com técnicas como quantização, poda, modelos esparsos, MoE e hardware especializado. Métodos de inferência seletiva e paralelismo buscam reduzir a latência e manter a confiabilidade estatística. A integração de computação aproximada e compiladores otimizados também tem possibilitado o uso eficiente de LLMs em ambientes restritos. Paralelamente, crescem as preocupações com privacidade, viés e uso indevido de dados, incentivando técnicas como privacidade diferencial e aprendizado federado. No campo regulatório, iniciativas internacionais visam equilibrar inovação com direitos fundamentais. Parcerias institucionais e práticas de ciência aberta têm se mostrado essenciais para o avanço responsável, ético e sustentável desses modelos. A expansão do uso de LLMs deve ser acompanhada por uma abordagem crítica e proativa quanto aos seus impactos éticos, sociais e ambientais. A combinação de regulação inteligente, arquitetura sustentável e colaboração interdisciplinar é essencial para garantir que esses modelos avancem com responsabilidade e tragam benefícios reais à sociedade.

4.9. Considerações Finais

O capítulo apresentou uma análise detalhada sobre os desafios e estratégias para viabilizar a execução de Modelos de Linguagem em Larga Escala (LLMs) em dispositivos com restrições computacionais, como smartphones, sistemas embarcados e equipamentos de borda. Destacou-se o impacto transformador dos LLMs no processamento de linguagem natural, com ênfase em suas capacidades de compreensão contextual, geração fluente e adaptação a diferentes domínios. Foram discutidos os fundamentos teóricos, as arquiteturas baseadas em *Transformers* e autoatenção, bem como técnicas de otimização voltadas à redução de consumo de memória, energia e tempo de inferência. A atividade prática, centrada em quantização e personalização, evidenciou os ganhos concretos dessas abordagens na implantação local de LLMs. Entre as técnicas abordadas, destacaram-se a quantização, a poda e a destilação de conhecimento, que compõem o núcleo das estratégias de compressão de modelos. A quantização, por exemplo, demonstrou-se eficaz ao reduzir a precisão numérica para otimizar memória e acelerar a inferência, com ênfase em métodos como AWQ e PTQ. A poda permitiu eliminar parâmetros redundantes, promovendo esparsidade e eficiência em hardware restrito. Já a destilação mostrou-se útil para transferir o desempenho de modelos maiores para arquiteturas menores, mantendo a qualidade das respostas. Também foram discutidas variantes arquiteturais, como os *Transformers* eficientes (*Linformer* e *Performer*), e mecanismos de atenção adaptativa, voltados à escalabilidade em tarefas extensas.

Como trabalhos futuros, três direções promissoras de pesquisa são vislumbradas. Primeiramente, há o desenvolvimento de técnicas de quantização e poda que incorporem o contexto semântico das tarefas de aplicação, promovendo uma compactação mais informada. Outra direção de pesquisas futuras é o desenvolvimento de LLMs integrados a sensores embarcados e entradas multimodais, que pode ampliar significativamente sua utilidade em sistemas ciberfísicos. Por fim, destaca-se a necessidade urgente de pesquisas voltadas à segurança e confiabilidade dos LLMs, sobretudo no que se refere ao alinhamento ético, mitigação de vieses, proteção contra usos maliciosos e o desenvolvimento de guarda-corpos (*guardrails*) autônomos e adaptáveis, essenciais para garantir o uso responsável de modelos em larga escala em dispositivos acessíveis.

Por fim, espera-se que as técnicas discutidas nesse capítulo sirvam de base para avanços futuros na área, permitindo que dispositivos móveis se beneficiem cada vez mais de modelos de linguagem robustos. Pesquisas contínuas em modelos mais eficientes e estratégias de alinhamento ético serão cruciais para consolidar a aplicação de Modelos de Linguagem de Larga Escala em dispositivos com hardware restrito.

Agradecimentos

Os autores agradecem aos alunos Gabriel de Almeida Campos, Ricardo Araujo de Souza Filho, Lucca Vieira Chatack e Igor Rodrigues Alves pelo apoio no desenvolvimento dos códigos para a atividade prática.

Referências

- [Abdin et al., 2024] Abdin M, Aneja J, Awadalla H, Awadallah A, Awan A A, Bach N, et al. (2024). Phi-3 technical report: A highly capable language model locally on your phone. arXiv:2404.14219 v4 [cs.CL].
- [Aftan e Shah, 2023] Aftan S, Shah H (2023). A survey on BERT and its applications. Em *2023 20th Learning and Technology Conference (LT)*, p. 161–166.
- [Ainslie et al., 2023] Ainslie J, Lee Thorp J, de Jong M, Zemlyanskiy Y, Lebrón F, Sanghai S (2023). GQA: Training generalized multi-query transformer models from multi-head checkpoints. arXiv:2305.13245 [cs.CL].
- [Anil et al., 2023] Anil R, Dai A M, Firat O, Johnson M, Lepikhin D, Passos A, et al. (2023). PaLM 2 technical report. arXiv:2305.10403 [cs.CL].
- [Argerich e Patiño-Martínez, 2024] Argerich M F, Patiño Martínez M (2024). Measuring and improving the energy efficiency of large language models inference. *IEEE Access*, 12:80194–80207.
- [Bahdanau et al., 2014] Bahdanau D, Cho K, Bengio Y (2014). Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.
- [Bai et al., 2024] Bai X, Wang A, Sucholutsky I, Griffiths T L (2024). Measuring implicit bias in explicitly unbiased large language models. *arXiv preprint arXiv:2402.04105*.
- [Banerjee e Lavie, 2005] Banerjee S, Lavie A (2005). METEOR: An automatic metric for MT evaluation with improved correlation with human judgments. Em *Proceedings of the acl workshop on intrinsic and extrinsic evaluation measures for machine translation and/or summarization*, p. 65–72.
- [Baroni, 2021] Baroni M (2021). On the proper role of linguistically-oriented deep net analysis in linguistic theorizing. *CoRR*, abs/2106.08694.
- [Cai et al., 2023] Cai T, Herner K, Yang T, Wang M, Acosta Flechas M, Harris P, et al. (2023). Accelerating machine learning inference with GPUs in ProtoDUNE data processing. *Computing and Software for Big Science*, 7(1).
- [Chang et al., 2024] Chang Y, Wang X, Wang J, Wu Y, Yang L, Zhu K, et al. (2024). A survey on evaluation of large language models. *ACM Trans. Intell. Syst. Technol.*, 15(3).
- [Chowdhery et al., 2023] Chowdhery A, Narang S, Devlin J, Bosma M, Mishra G, Roberts A, et al. (2023). PaLM: Scaling language modeling with pathways. *Journal of Machine Learning Research*, 24(240):1–113.

- [Chung et al., 2024] Chung H W, Hou L, Longpre S, Zoph B, Tay Y, Fedus W, et al. (2024). Scaling instruction-finetuned language models. *Journal of Machine Learning Research*, 25(70):1–53.
- [Cortiz e Zubiaga, 2021] Cortiz D, Zubiaga A (2021). Ethical and technical challenges of AI in tackling hate speech. *The International Review of Information Ethics*, 29.
- [Dagdelen et al., 2024] Dagdelen J, Dunn A, Lee S, Walker N, Rosen A S, Ceder G, Persson K A, Jain A (2024). Structured information extraction from scientific text with large language models. *Nature Communications*, 15(1):1418.
- [Dao et al., 2022] Dao T, Fu D, Ermon S, Rudra A, Ré C (2022). FlashAttention: Fast and memory-efficient exact attention with IO-awareness. Em Koyejo S, Mohamed S, Agarwal A, Belgrave D, Cho K, Oh A, editors, *Advances in Neural Information Processing Systems*, volume 35, p. 16344–16359. Curran Associates, Inc.
- [de Oliveira et al., 2025] de Oliveira N R, Campos G, Rodrigues I, Watanabe J A C, Couto R S, Moraes I M, Medeiros D, Mattos D M F (2025). Quantization effects on large language models for intent-based network management. Em *Proceedings of the 17th International Conference on Network and Telecommunications Management Systems (NTMS)*, Paris, France. IEEE. A ser apresentado.
- [de Vries, 2023] de Vries A (2023). The growing energy footprint of artificial intelligence. *Joule*, 7(10):2191–2194.
- [Deranty e Corbin, 2024] Deranty J P, Corbin T (2024). Artificial intelligence and work: a critical review of recent research from the social sciences. *Ai & Society*, 39(2):675–691.
- [Divakaran e Peddinti, 2024] Divakaran D M, Peddinti S T (2024). Large language models for cybersecurity: New opportunities. *IEEE Security & Privacy*, p. 2–9.
- [Djeffal et al., 2023] Djeffal N, Kheddar H, Addou D, Mazari A C, Himeur Y (2023). Automatic speech recognition with BERT and CTC transformers: A review. Em *2023 2nd International Conference on Electronics, Energy and Measurement (IC2EM)*, volume 1, p. 1–8.
- [Dubey et al., 2024] Dubey A, Jauhri A, Pandey A, Kadian A, Al Dahle A, Letman A, et al. (2024). The Llama 3 herd of models.
- [Falcon-LLM Team, 2024] Falcon-LLM Team (2024). The Falcon 3 family of open models. Relatório técnico, Technology Innovation Institute (TII).
- [Fang et al., 2024] Fang G, Yin H, Muralidharan S, Heinrich G, Pool J, Kautz J, Molchanov P, Wang X (2024). MaskLLM: Learnable semi-structured sparsity for large language models. *arXiv preprint arXiv:2409.17481*.
- [Frantar e Alistarh, 2023] Frantar E, Alistarh D (2023). SparseGPT: Massive language models can be accurately pruned in one-shot. Em *International Conference on Machine Learning*, p. 10323–10337. PMLR.
- [Gemma Team et al., 2025] Gemma Team, Kamath A, Ferret J, Pathak S, Vieillard N, Merhej R, et al. (2025). Gemma 3 technical report. arXiv:2503.19786 [cs.CL].
- [Ghielmetti et al., 2022] Ghielmetti N, Loncar V, Pierini M, Roed M, Summers S, Arrestad T, et al. (2022). Real-time semantic segmentation on FPGAs for autonomous vehicles with hls4ml. *Machine Learning: Science and Technology*, 3(4):045011.

- [Gou et al., 2021] Gou J, Yu B, Maybank S J, Tao D (2021). Knowledge distillation: A survey. *International Journal of Computer Vision*, 129(6):1789–1819.
- [Gross, 2023] Gross N (2023). What ChatGPT tells us about gender: A cautionary tale about performativity and gender biases in AI. *Social Sciences*, 12(8):435.
- [Gu et al., 2023] Gu Y, Dong L, Wei F, Huang M (2023). Minillm: Knowledge distillation of large language models. Em *The Twelfth International Conference on Learning Representations*.
- [Gunny et al., 2022] Gunny A, Rankin D, Harris P, Katsavounidis E, Marx E, Saleem M, Coughlin M, Benoit W (2022). A software ecosystem for deploying deep learning in gravitational wave physics. Em *Proceedings of the 12th Workshop on AI and Scientific Computing at Scale using Flexible Computing Infrastructures*, HPDC '22, p. 9–17. ACM.
- [Guo et al., 2025] Guo D, Yang D, Zhang H, Song J, Zhang R, Xu R, et al. (2025). DeepSeek-R1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.
- [Henderson et al., 2018] Henderson P, Sinha K, Angelard Gontier N, Ke N R, Fried G, Lowe R, Pineau J (2018). Ethical challenges in data-driven dialogue systems. Em *Proceedings of the 2018 AAAI/ACM Conference on AI, Ethics, and Society*, AIES '18, p. 123–129, New York, NY, USA. Association for Computing Machinery.
- [Hendrycks et al., 2020] Hendrycks D, Burns C, Basart S, Zou A, Mazeika M, Song D, Steinhardt J (2020). Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*.
- [Huang et al., 2024] Huang Y, Wan L J, Ye H, Jha M, Wang J, Li Y, Zhang X, Chen D (2024). Invited: New solutions on LLM acceleration, optimization, and application. Em *Proceedings of the 61st ACM/IEEE Design Automation Conference*, DAC '24, New York, NY, USA. Association for Computing Machinery.
- [Jadouli e Amrani, 2025] Jadouli A, Amrani C E (2025). Deep learning with pretrained'internal world'layers: A Gemma 3-based modular architecture for wildfire prediction. *arXiv preprint arXiv:2504.18562*.
- [Jawahar et al., 2019] Jawahar G, Sagot B, Seddah D (2019). What does BERT learn about the structure of language? Em *ACL 2019-57th Annual Meeting of the Association for Computational Linguistics*.
- [Jiang et al., 2024] Jiang Z, Carlson B, Deiana A, Eastlack J, Hauck S, Hsu S C, et al. (2024). Machine learning evaluation in the global event processor FPGA for the ATLAS trigger upgrade. *Journal of Instrumentation*, 19(05):P05031.
- [Jiao et al., 2025] Jiao J, Afroogh S, Xu Y, Phillips C (2025). Navigating LLM ethics: Advancements, challenges, and future directions. *arXiv:2406.18841v4 [cs.CY]*.
- [Kaddour et al., 2023] Kaddour J, Harris J, Mozes M, Bradley H, Raileanu R, McHardy R (2023). Challenges and applications of large language models. *arXiv preprint arXiv:2307.10169*.
- [Khoda et al., 2023] Khoda E E, Rankin D, Teixeira de Lima R, Harris P, Hauck S, Hsu S C, et al. (2023). Ultra-low latency recurrent neural network inference on FPGAs for physics applications with hls4ml. *Machine Learning: Science and Technology*, 4(2):025004.

- [Kotek et al., 2023] Kotek H, Dockum R, Sun D (2023). Gender bias and stereotypes in large language models. Em *Proceedings of The ACM Collective Intelligence Conference, CI '23*, p. 12–24. ACM.
- [Lan et al., 2020] Lan Z, Chen M, Goodman S, Gimpel K, Sharma P, Soricut R (2020). ALBERT: A Lite BERT for self-supervised learning of language representations. *arXiv:1909.11942v6* [cs.CL].
- [Lewis et al., 2019] Lewis M, Liu Y, Goyal N, Ghazvininejad M, Mohamed A, Levy O, Stoyanov V, Zettlemoyer L (2019). BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *arXiv preprint arXiv:1910.13461*.
- [Li et al., 2023] Li L J, Zhou S L, Chao F, Chang X, Yang L, Yu X, Shang C, Shen Q (2023). Model compression optimized neural network controller for nonlinear systems. *Knowledge-Based Systems*, 265:110311.
- [Liang et al., 2023] Liang P, Bommasani R, Lee T, Tsipras D, Soylu D, Yasunaga M, et al. (2023). Holistic evaluation of language models. *Transactions on Machine Learning Research*. Featured Certification, Expert Certification.
- [Lin, 2004] Lin C Y (2004). ROUGE: a package for automatic evaluation of summaries. Em *Workshop on Text Summarization Branches Out, Post-Conference Workshop of ACL 2004, Barcelona, Spain*, p. 74–81.
- [Lin et al., 2024] Lin J, Tang J, Tang H, Yang S, Chen W M, Wang W C, et al. (2024). AWQ: Activation-aware weight quantization for on-device LLM compression and acceleration. *Proceedings of Machine Learning and Systems*, 6:87–100.
- [Lin et al., 2022] Lin T, Wang Y, Liu X, Qiu X (2022). A survey of transformers. *AI open*, 3:111–132.
- [Liu et al., 2024a] Liu S, Lu Y, Fang W, Li M, Xie Z (2024a). OpenLLM-RTL: Open dataset and benchmark for LLM-aided design RTL generation. Em *Proceedings of the 43rd IEEE/ACM International Conference on Computer-Aided Design, ICCAD '24*, p. 1–9. ACM.
- [Liu et al., 2023] Liu Z, Oguz B, Zhao C, Chang E, Stock P, Mehdad Y, et al. (2023). Llm-qat: Data-free quantization aware training for large language models. *arXiv preprint arXiv:2305.17888*.
- [Liu et al., 2024b] Liu Z, Zhao C, Iandola F, Lai C, Tian Y, Fedorov I, et al. (2024b). MobileLLM: optimizing sub-billion parameter language models for on-device use cases. ICML'24. JMLR.org.
- [Loukas et al., 2023] Loukas L, Stogiannidis I, Diamantopoulos O, Malakasiotis P, Vasos S (2023). Making LLMs worth every penny: Resource-limited text classification in banking. Em *Proceedings of the Fourth ACM International Conference on AI in Finance*, p. 392–400.
- [Ma et al., 2023] Ma X, Fang G, Wang X (2023). LLM-Pruner: On the structural pruning of large language models. *Advances in neural information processing systems*, 36:21702–21720.
- [Mahmud et al., 2025] Mahmud D, Hajmohamed H, Almentheri S, Alqaydi S, Aldhaheer L, Khalil R A, Saeed N (2025). Integrating LLMs with ITS: Recent advances, potentials, challenges, and future directions. *IEEE Transactions on Intelligent Transportation Systems*, 26(5):5674–5709.

- [Miao et al., 2022] Miao S, Liu M, Li P (2022). Interpretable and generalizable graph learning via stochastic attention mechanism. Em Chaudhuri K, Jegelka S, Song L, Szepesvari C, Niu G, Sabato S, editors, *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, p. 15524–15543. PMLR.
- [Naveed et al., 2023] Naveed H, Khan A U, Qiu S, Saqib M, Anwar S, Usman M, et al. (2023). A comprehensive overview of large language models. *arXiv preprint arXiv:2307.06435*.
- [OpenAI et al., 2024] OpenAI, Achiam J, Adler S, Agarwal S, Ahmad L, Akkaya I, et al. (2024). GPT-4 technical report. *arXiv:2303.08774 [cs.CL]*.
- [Ouyang et al., 2022] Ouyang L, Wu J, Jiang X, Almeida D, Wainwright C, Mishkin P, et al. (2022). Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744.
- [Penedo et al., 2023] Penedo G, Malartic Q, Hesslow D, Cojocaru R, Cappelli A, Alobeidli H, et al. (2023). The RefinedWeb dataset for Falcon LLM: Outperforming curated corpora with web data, and web data only. *arXiv:2306.01116 [cs.CL]*.
- [Press et al., 2022] Press O, Smith N A, Lewis M (2022). Train short, test long: Attention with linear biases enables input length extrapolation. *arXiv:2108.12409 [cs.CL]*.
- [Raffel et al., 2020] Raffel C, Shazeer N, Roberts A, Lee K, Narang S, Matena M, et al. (2020). Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, 21(140):1–67.
- [Raiaan et al., 2024] Raiaan M A K, Mukta M S H, Fatema K, Fahad N M, Sakib S, Mim M M J, et al. (2024). A review on large language models: Architectures, applications, taxonomies, open issues and challenges. *IEEE Access*, 12:26839–26874.
- [Sanh et al., 2020] Sanh V, Debut L, Chaumond J, Wolf T (2020). DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. *arXiv:1910.01108 [cs.CL]*.
- [Shazeer, 2019] Shazeer N (2019). Fast transformer decoding: One write-head is all you need. *arXiv:1911.02150 [cs.NE]*.
- [Sun et al., 2023] Sun M, Liu Z, Bair A, Kolter J Z (2023). A simple and effective pruning approach for large language models. *arXiv preprint arXiv:2306.11695*.
- [Suzgun et al., 2023] Suzgun M, Scales N, Schärli N, Gehrmann S, Tay Y, Chung H W, et al. (2023). Beyond the imitation game: Quantifying and extrapolating the capabilities of language models. *Transactions on Machine Learning Research*.
- [Tay et al., 2022] Tay Y, Dehghani M, Bahri D, Metzler D (2022). Efficient transformers: A survey. *ACM Comput. Surv.*, 55(6).
- [Tian et al., 2023] Tian K, Mitchell E, Zhou A, Sharma A, Rafailov R, Yao H, Finn C, Manning C D (2023). Just ask for calibration: Strategies for eliciting calibrated confidence scores from language models fine-tuned with human feedback. *arXiv preprint arXiv:2305.14975*.
- [Touvron et al., 2023] Touvron H, Lavril T, Izacard G, Martinet X, Lachaux M A, Lacroix T, et al. (2023). Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

- [V. et al., 2025] V. J M N, Giraldo D M, Segura S G, Corchado J M, la Prieta F D (2025). Bioinspired small language models in edge systems for bee colony monitoring and control. *Internet of Things*, 32:101633.
- [Vaswani et al., 2017] Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez A N, Kaiser L, Polosukhin I (2017). Attention is all you need. Em *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS' 17, p. 6000–6010, Red Hook, NY, USA. Curran Associates Inc.
- [Wang et al., 2023] Wang J, Hu X, Hou W, Chen H, Zheng R, Wang Y, et al. (2023). On the robustness of chatgpt: An adversarial and out-of-distribution perspective. *arXiv preprint arXiv:2302.12095*.
- [Weidinger et al., 2022] Weidinger L, Uesato J, Rauh M, Griffin C, Huang P S, Mellor J, et al. (2022). Taxonomy of risks posed by language models. Em *Proceedings of the 2022 ACM conference on fairness, accountability, and transparency*, p. 214–229.
- [Xu et al., 2024] Xu J, Li Z, Chen W, Wang Q, Gao X, Cai Q, Ling Z (2024). On-device language models: A comprehensive review. *arXiv preprint arXiv:2409.00088*.
- [Xue et al., 2023] Xue M, Liu D, Yang K, Dong G, Lei W, Yuan Z, Zhou C, Zhou J (2023). OccuQuest: Mitigating occupational bias for inclusive large language models. *arXiv:2310.16517v1 [cs.CL]*.
- [Yan et al., 2025] Yan X, Zhang T, Li Z, Zhang Y (2025). Progressive binarization with semi-structured pruning for LLMs. *arXiv preprint arXiv:2502.01705*.
- [Yang et al., 2025] Yang A, Yu B, Li C, Liu D, Huang F, Huang H, et al. (2025). Qwen2.5-1m technical report. *arXiv:2501.15383 [cs.CL]*.
- [Yang et al., 2024] Yang J, Jin H, Tang R, Han X, Feng Q, Jiang H, et al. (2024). Harnessing the power of LLMs in practice: A survey on chatgpt and beyond. *ACM Trans. Knowl. Discov. Data*, 18(6).
- [Yao et al., 2024a] Yao Y, Duan J, Xu K, Cai Y, Sun Z, Zhang Y (2024a). A survey on large language model (LLM) security and privacy: The good, the bad, and the ugly. *High-Confidence Computing*, p. 100211.
- [Yao et al., 2024b] Yao Y, Duan J, Xu K, Cai Y, Sun Z, Zhang Y (2024b). A survey on large language model (LLM) security and privacy: The good, the bad, and the ugly. *High-Confidence Computing*, 4(2):100211.
- [Yenduri et al., 2024] Yenduri G, Ramalingam M, Selvi G C, Supriya Y, Srivastava G, Maddikunta P K R, et al. (2024). GPT (generative pre-trained transformer)— a comprehensive review on enabling technologies, potential applications, emerging challenges, and future directions. *IEEE Access*, 12:54608–54649.
- [Zhang et al., 2024] Zhang P, Zeng G, Wang T, Lu W (2024). TinyLlama: An open-source small language model. *arXiv:2401.02385v2 [cs.CL]*.
- [Zhang et al., 2025] Zhang Q, Liu Z, Pan S (2025). The rise of small language models. *IEEE Intelligent Systems*, 40(1):30–37.
- [Zhang et al., 2020] Zhang T, Kishore V, Wu F, Weinberger K Q, Artzi Y (2020). BERTScore: Evaluating text generation with BERT. Em *International Conference on Learning Representations*.

- [Zhao et al., 2019] Zhao W, Peyrard M, Liu F, Gao Y, Meyer C M, Eger S (2019). MoverScore: Text generation evaluating with contextualized embeddings and earth mover distance. Em *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*, Hong Kong, China. Association for Computational Linguistics.
- [Zhao et al., 2023] Zhao W X, Zhou K, Li J, Tang T, Wang X, Hou Y, et al. (2023). A survey of large language models. *arXiv preprint arXiv:2303.18223*, 1(2).
- [Zhao et al., 2024] Zhao Y, Lin C Y, Zhu K, Ye Z, Chen L, Zheng S, et al. (2024). Atom: Low-bit quantization for efficient and accurate llm serving. *Proceedings of Machine Learning and Systems*, 6:196–209.
- [Zheng et al., 2021] Zheng X, Zhang C, Woodland P C (2021). Adapting GPT, GPT-2 and BERT language models for speech recognition. Em *2021 IEEE Automatic speech recognition and understanding workshop (ASRU)*, p. 162–168. IEEE.
- [Zheng et al., 2025] Zheng Z, Ning K, Zhong Q, Chen J, Chen W, Guo L, Wang W, Wang Y (2025). Towards an understanding of large language models in software engineering tasks. *Empirical Software Engineering*, 30(2):50.
- [Zhu et al., 2024a] Zhu K, Wang J, Zhou J, Wang Z, Chen H, Wang Y, et al. (2024a). PromptRobust: Towards evaluating the robustness of large language models on adversarial prompts. Em *Proceedings of the 1st ACM Workshop on Large AI Systems and Models with Privacy and Safety Analysis, LAMPS '24*, p. 57–68, New York, NY, USA. Association for Computing Machinery.
- [Zhu et al., 2024b] Zhu X, Li J, Liu Y, Ma C, Wang W (2024b). A survey on model compression for large language models. *Transactions of the Association for Computational Linguistics*, 12:1556–1577.