ENUCOMPI

ENCONTRO UNIFICADO DE COMPUTAÇÃO DO PIAUÍ

LIVRO DE MINICURSOS

EDIÇÃO - 2025

ORGANIZADORES:

Dimmy Karson Soares Magalhães
Eduilson Lívio Neves da Costa Carneiro
Francisco Luciani de Miranda Vieira
Ricardo Moura Sekeff Budaruiche
Rodrigo Augusto Rocha Souza Baluz

REALIZAÇÃO: icev Instituto de Ensino Superior

APOIO: SBC Sociedade Brasileira de Computação

G Bank Feito para o hoje

XVII Encontro Unificado de Computação do Piauí
28 a 30 de maio de 2025
Teresina - Piauí

LIVRO DE MINICURSOS
ENUCOMPI 2025

Organização do Livro
DIMMY KARSON SOARES MAGALHÃES
EDUILSON LÍVIO NEVES DA COSTA CARNEIRO
FRANCISCO LUCIANI DE MIRANDA VIEIRA
RICARDO MOURA SEKEFF BUDARUICHE
RODRIGO AUGUSTO ROCHA SOUZA BALUZ

Sociedade Brasileira de Computação
Porto Alegre
2025

Dados Internacionais de Catalogação na Publicação (CIP)

E56 Encontro Unificado de Computação do Piauí e Semana Nacional de Ciência e Tecnologia (25. : 28 – 30 maio 2025 : Teresina)
Minicursos do ENUCOMPI [recurso eletrônico] / organização: Dimmy Karson Soares Magalhães. Dados eletrônicos. – Porto Alegre: Sociedade Brasileira de Computação, 2025.
PDF ; 5 MB

Modo de acesso: World Wide Web.
Inclui bibliografia
ISBN 978-85-7669-635-3 (e-book)

1. Computação – Brasil – Evento. 2. Ciência e Tecnologia. I. Magalhães, Dimmy Karson Soares. II. Carneiro, Eduilson Lívio Neves da Costa. III. Vieira, Francisco Luciani de Miranda. IV. Budaruiche, Ricardo Moura Sekeff. V. Baluz, Rodrigo Augusto Rocha Souza. VI. Sociedade Brasileira de Computação. VII. Título.

CDU 004(063)

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

Biblioteca Digital da SBC – SBC OpenLib

Índices para catálogo sistemático:

1. Ciência e tecnologia dos computadores : Informática – Publicação de conferências, congressos e simpósios etc. ... 004(063)

Mensagem da Coordenação Geral

É com grande satisfação que apresentamos aos leitores o **Livro de Minicursos** do XVII Encontro Unificado de Computação do Piauí (ENUCOMPI 2025), realizado entre os dias 28 e 30 de maio, como parte da programação do IV Congresso de Direito, Negócios e Tecnologia (DNT), promovido pelo Instituto de Ensino Superior (iCEV), em Teresina, Piauí. Este evento consolida-se, a cada edição, como um espaço fundamental para a troca de experiências, a disseminação do conhecimento e a promoção da inovação no cenário da Computação no nosso estado.

Os minicursos aqui reunidos refletem a diversidade de temas e a profundidade dos debates que marcaram o ENUCOMPI 2025, reunindo estudantes, professores, pesquisadores e profissionais de diversas instituições. Com enfoques práticos e atualizados, as atividades ofereceram oportunidades únicas de aprendizado e desenvolvimento de habilidades em áreas como Inteligência Artificial, Desenvolvimento de Software, Segurança da Informação, Computação em Nuvem, entre outras.

A realização deste livro é resultado do esforço coletivo de coordenadores, ministrantes, avaliadores e da equipe organizadora do evento, que não mediram esforços para garantir a qualidade acadêmica e a relevância dos conteúdos apresentados. Agradecemos a todos que contribuíram com dedicação e entusiasmo para o sucesso deste encontro.

Esperamos que este material inspire novos projetos, fortaleça redes de colaboração e continue alimentando o interesse pela Computação e suas múltiplas possibilidades. Desejamos a todos uma excelente leitura e que este livro seja não apenas um registro, mas também um ponto de partida para novas jornadas de conhecimento e inovação.

Sinopse

O **Livro de Minicursos do XVII Encontro Unificado de Computação do Piauí (ENUCOMPI 2025)** reúne uma seleção de conteúdos práticos e inovadores que refletem o dinamismo e a diversidade da área de Computação na atualidade. Organizado durante o IV Congresso de Direito, Negócios e Tecnologia (DNT), promovido pelo iCEV, em Teresina (PI), esta obra é um registro técnico e didático das oficinas realizadas, voltadas tanto para iniciantes quanto para profissionais e pesquisadores que buscam atualização e aprofundamento.

Os capítulos exploram tecnologias emergentes e aplicações práticas com foco na resolução de problemas reais e no estímulo à criatividade. Os temas incluem desde a criação de experiências digitais imersivas por meio do mapeamento de vídeo em tempo real com redes neurais, até o uso de LangGraph para desenvolver assistentes de pesquisa científica personalizados. A obra também apresenta fundamentos e aplicações de Smart Contracts, técnicas de classificação de imagens faciais com Deep Learning, automação de tarefas via integração de APIs como Google Calendar e WhatsApp, e ainda a utilização de rigging e normal map na Unity para criar animações 2D com efeitos tridimensionais.

Com uma linguagem acessível e orientada à prática, este livro oferece uma visão atualizada sobre o estado da arte em Computação, contribuindo para a formação técnica e intelectual de seus leitores. É uma leitura essencial para quem deseja acompanhar a transformação digital de forma ativa, criativa e crítica.

Synopsis

The **XVII ENUCOMPI (Unificado Computing Meeting of Piauí) 2025 Minicourse Book** brings together a selection of practical and innovative content that reflects the dynamism and diversity of today's computing landscape. Organized as part of the IV Congress on Law, Business, and Technology (DNT), hosted by iCEV in Teresina, Brazil, this volume serves as a technical and educational record of the hands-on workshops held during the event, aimed at both beginners and professionals seeking updated knowledge and deeper insights.

The chapters explore emerging technologies and real-world applications, encouraging problem-solving and creativity. Topics include the development of immersive digital experiences through real-time video mapping with neural networks, creating personalized scientific research assistants using LangGraph, an introduction to Smart Contracts, facial image classification using Convolutional Neural Networks, task automation via integration between APIs like Google Calendar and WhatsApp, and the use of rigging and normal maps in Unity to build 2D animations with depth and dynamic lighting from a single image.

With accessible and practice-oriented language, this book offers an up-to-date view of the state of the art in Computing, contributing to the technical and intellectual growth of its readers. It is essential reading for anyone looking to actively, creatively, and critically engage with the digital transformation.

Sumário

Chapter 1

Introdução aos Smart Contracts

Frank Cesár L. Verás e James D. Sousa

Abstract

Smart contracts are self-executing programs stored on blockchain, ensuring secure and transparent execution of digital agreements. This minicourse aims to present the fundamental concepts, advantages, applications, and challenges of smart contracts.

Resumo

Os smart contracts (contratos inteligentes) são programas autoexecutáveis armazenados em blockchain, garantindo execução segura e transparente de acordos digitais. Este minicurso tem como objetivo apresentar os conceitos fundamentais, vantagens, aplicações e desafios dos smart contracts.

1.1. Introdução

A transformação digital tem promovido profundas mudanças na forma como a sociedade interage, realiza negócios e gerencia informações. Nesse contexto, os contratos inteligentes, ou *smart contracts*, surgem como uma evolução natural das tecnologias baseadas em blockchain, proporcionando uma maneira automatizada, segura e transparente de executar acordos digitais.

A ideia dos contratos inteligentes foi introduzida pela primeira vez na década de 1990 por Nick Szabo, um criptógrafo e cientista da computação com interesse profundo em direito contratual. Em seu artigo seminal de 1994, Szabo definiu os *smart contracts* como protocolos computacionais capazes de formalizar, verificar e executar automaticamente os termos de um contrato, sem a necessidade de intermediários ou autoridades centrais.

Para ilustrar o conceito, Szabo utilizou o exemplo de uma máquina de venda automática. Ao inserir uma quantia específica de dinheiro e selecionar um item, a máquina realiza automaticamente a transação, entregando o produto sem intervenção humana. Essa

analogia descreve o funcionamento básico dos contratos inteligentes: uma condição pré-programada que, quando satisfeita, aciona automaticamente uma ação.

Apesar da inovação teórica, a proposta de Szabo permaneceu por anos sem aplicação prática, devido à inexistência de uma infraestrutura tecnológica descentralizada que garantisse integridade, imutabilidade e confiabilidade. Essa limitação foi superada com o surgimento do blockchain — especialmente com o lançamento da plataforma Ethereum, em 2015.

Idealizado por Vitalik Buterin, o Ethereum introduziu uma plataforma descentralizada programável, com suporte à linguagem Solidity, que permitiu o desenvolvimento de contratos inteligentes personalizados. Por meio do uso do blockchain como base de dados imutável e validada por consenso distribuído, os contratos inteligentes do Ethereum permitiram a criação de aplicações descentralizadas (*dApps*) com segurança e transparência inigualáveis até então.

Desde então, outras plataformas blockchain como Binance Smart Chain, Solana, Cardano e Avalanche também passaram a oferecer suporte a contratos inteligentes, cada uma com suas próprias linguagens de programação e mecanismos de consenso. Essa diversidade tecnológica ampliou as possibilidades de uso e impulsionou inovações em setores como finanças descentralizadas (DeFi), gestão de cadeias de suprimento, propriedade intelectual, sistemas eleitorais e governança digital.

O estudo dos contratos inteligentes torna-se, portanto, essencial para profissionais e pesquisadores que atuam nas áreas de tecnologia, direito, economia e segurança da informação. Este minicurso tem como objetivo apresentar os fundamentos teóricos, a evolução histórica, as plataformas atuais e as aplicações mais relevantes dos contratos inteligentes no cenário contemporâneo.

1.2. O que são Contratos Inteligentes

Contratos inteligentes, ou *smart contracts*, são programas de computador armazenados em uma rede blockchain que executam automaticamente termos de um acordo quando determinadas condições pré-definidas são satisfeitas. Diferentemente dos contratos tradicionais, que dependem da confiança entre as partes e da mediação de terceiros (como advogados, cartórios ou bancos), os contratos inteligentes utilizam lógica computacional para garantir a execução imutável e autônoma das cláusulas estabelecidas.

A essência de um contrato inteligente está na ideia de que "o código é a lei" (*code is law*). Ou seja, uma vez implementado e implantado na blockchain, o contrato age conforme o código que o define, sem a possibilidade de intervenção ou modificação externa. Isso garante transparência, previsibilidade e confiança na execução do acordo, já que todos os participantes podem verificar publicamente o seu funcionamento.

A estrutura básica de um contrato inteligente inclui:

- **Condições**: instruções que determinam quando e como uma ação será executada;
- **Ações**: operações automáticas que ocorrem quando as condições são satisfeitas (como transferir um valor, conceder acesso, registrar um dado, etc.);

- **Estado**: dados armazenados na blockchain que representam o contexto ou progresso do contrato;
- **Eventos**: notificações que informam à rede que uma determinada operação foi realizada.

Por estarem implantados em blockchains, os contratos inteligentes herdam suas características fundamentais: descentralização, imutabilidade, auditabilidade e resistência à censura. Isso significa que qualquer tentativa de adulteração, interrupção ou manipulação do contrato é tecnicamente inviável, a menos que haja consenso da rede — o que garante um alto nível de segurança.

Além disso, contratos inteligentes podem interagir com outros contratos e aplicações descentralizadas, formando ecossistemas complexos e autossustentáveis. Essa capacidade de composição é um dos fatores que torna os *smart contracts* fundamentais para o funcionamento da Web3 e das chamadas Finanças Descentralizadas (DeFi).

Em resumo, contratos inteligentes representam uma mudança de paradigma na forma como sistemas digitais podem ser programados para estabelecer confiança, reduzir custos e eliminar intermediários em processos automatizados e transparentes.

1.2.1. Benefícios dos Smart Contracts

Os smart contracts oferecem uma série de benefícios que justificam seu crescente uso em diversas aplicações, especialmente em ambientes que exigem confiança e automação. Esses contratos programáveis tornam possível a execução automática de acordos sem a necessidade de intermediários, o que contribui para maior eficiência e segurança. A seguir, destacam-se os principais benefícios proporcionados por essa tecnologia:

- **Autonomia**: Um dos principais atrativos dos smart contracts é a eliminação de intermediários. Como as cláusulas contratuais são autoexecutáveis e registradas diretamente no código, as partes envolvidas não precisam depender de terceiros, como advogados, cartórios ou instituições financeiras, para validar ou supervisionar o cumprimento do contrato. Isso proporciona maior independência e controle para os usuários, além de reduzir possíveis conflitos de interesse.
- **Segurança**: Os smart contracts são implantados em redes blockchain, que oferecem elevados níveis de segurança criptográfica. Uma vez inserido na blockchain, o código do contrato é praticamente imutável, tornando extremamente difícil qualquer tentativa de manipulação ou fraude. Além disso, a natureza descentralizada da blockchain reforça a resiliência do sistema, garantindo que as transações ocorram mesmo em situações de ataque ou falhas de infraestrutura.
- **Transparência**: Toda a lógica e as condições de um smart contract são públicas e auditáveis por qualquer usuário da rede. Isso significa que todos os envolvidos podem verificar o código e assegurar que ele funciona exatamente como prometido. Essa transparência reduz drasticamente o risco de mal-entendidos ou interpretações ambíguas, fortalecendo a confiança entre as partes e proporcionando maior previsibilidade na execução dos acordos.

- **Eficiência**: A automação proporcionada pelos smart contracts elimina etapas burocráticas e manuais, acelerando a execução de transações e processos. Além disso, como não há necessidade de intermediários, os custos operacionais são significativamente reduzidos. A execução instantânea e a redução de falhas humanas tornam os contratos inteligentes uma ferramenta poderosa para otimizar recursos e aumentar a produtividade em diversas aplicações, desde pagamentos automatizados até cadeias logísticas complexas.

1.3. Funcionamento Geral e Arquitetura dos Contratos Inteligentes

Os contratos inteligentes funcionam com base na tecnologia blockchain, que fornece a infraestrutura necessária para garantir a execução descentralizada, segura e imutável dos programas. Para compreender como esses contratos operam, é essencial analisar sua arquitetura básica, os componentes envolvidos e o fluxo de execução típico.

1.3.1. Estrutura Técnica

Um contrato inteligente é essencialmente um código de programação que define regras e instruções armazenadas em um bloco da blockchain. Após ser implantado (deploy) na rede, ele passa a existir de forma permanente e pode ser acessado e executado por qualquer usuário ou outro contrato.

A estrutura interna de um contrato inteligente geralmente é composta por:

- **Variáveis de estado**: armazenam dados persistentes, como saldos, registros ou status.
- **Funções**: definem as ações que podem ser executadas, como transferências, atualizações ou verificações.
- **Eventos**: notificações que informam sobre mudanças ou execuções ocorridas.
- **Modificadores e restrições**: limitam o acesso a determinadas funções ou condições de execução.

A linguagem mais utilizada para escrever contratos inteligentes é a **Solidity**, que foi desenvolvida especificamente para a blockchain Ethereum. Outras linguagens populares incluem *Rust* (utilizada em Solana), *Michelson* (Tezos) e *Move* (Aptos e Sui).

1.3.2. Fluxo de Execução

O funcionamento de um contrato inteligente pode ser resumido nas seguintes etapas:

1. **Implantação (deploy)**: o contrato é compilado e enviado para a blockchain, gerando um endereço único.
2. **Interação**: usuários ou outros contratos interagem com ele por meio de transações.
3. **Execução**: o código do contrato é executado por todos os nós da rede, validado através do algoritmo de consenso da blockchain.

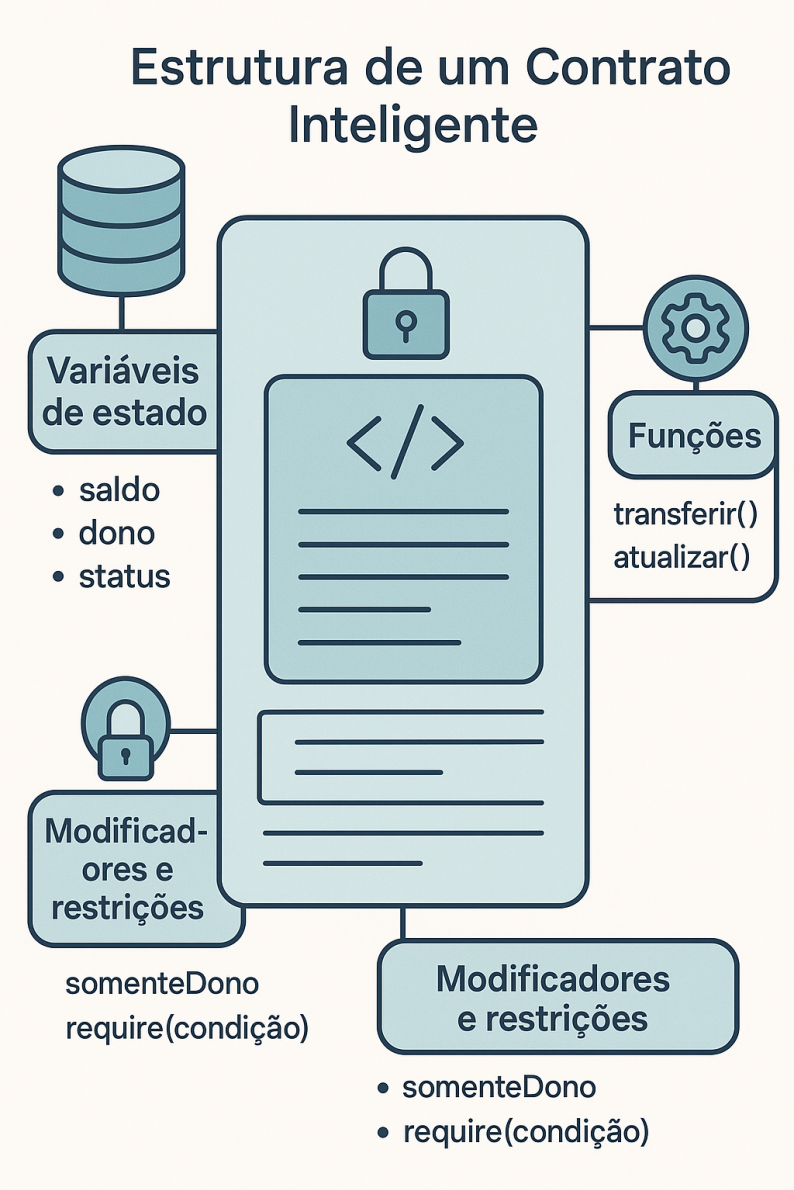


Figure 1.1. Estrutura Smart Contract

4. **Persistência**: o resultado da execução (como mudanças de estado) é armazenado de forma imutável.

Diferente de programas comuns que rodam em servidores, os contratos inteligentes são executados por todos os nós da rede blockchain, garantindo consenso e confiabilidade. Essa execução redundante assegura que o resultado será sempre o mesmo, independentemente de quem interaja com o contrato.

1.3.3. Modelo de Armazenamento e Custos

Toda operação realizada por um contrato inteligente consome recursos computacionais da rede. No Ethereum, esse custo é medido em *gas*, uma unidade que representa a quantidade

de trabalho necessário para realizar uma operação. O usuário que executa uma função do contrato deve pagar o custo correspondente em criptomoeda (por exemplo, ETH).

Além disso, os dados armazenados em contratos inteligentes ocupam espaço na blockchain, o que também gera custos de armazenamento. Por isso, desenvolvedores são incentivados a escrever códigos otimizados, que minimizem tanto o consumo de *gas* quanto o uso de armazenamento.

1.3.4. Oráculos e Interação com o Mundo Externo

Contratos inteligentes, por padrão, operam apenas com dados internos à blockchain. Para acessar informações externas como cotação de moedas, clima, ou resultados esportivos é necessário o uso de **oráculos**, que são serviços confiáveis responsáveis por trazer dados do mundo real para o ambiente blockchain. Sem os oráculos, os contratos ficariam isolados e incapazes de reagir a eventos externos.

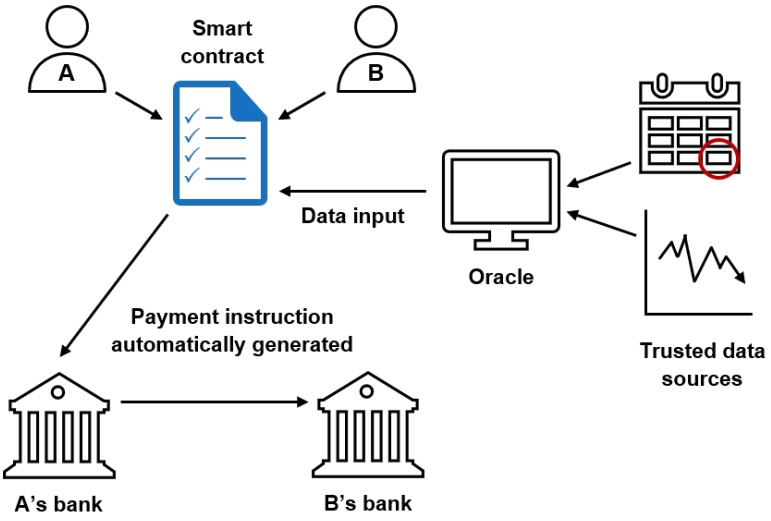


Figure 1.2. Oráculo Blockchain

O Universal Market Acess (UMA) é um importante oráculo que fornece uma infraestrutura para a criação de contratos inteligentes e aplicações financeiras descentralizadas. Ele utiliza uma combinação de dados on-chain e off-chain para alimentar contratos inteligentes com informações externas confiáveis. A sua criptomoeda se chama UMA.

1.3.5. Limitações Técnicas

Apesar de sua robustez, os contratos inteligentes ainda enfrentam algumas limitações:

- **Imutabilidade**: erros de código não podem ser corrigidos após o deploy.
- **Escalabilidade**: a execução por todos os nós pode gerar gargalos em redes congestionadas.
- **Complexidade no desenvolvimento**: exige conhecimento técnico específico e revisão rigorosa de segurança.

Apesar desses desafios, os contratos inteligentes continuam sendo uma das inovações mais promissoras da era blockchain, permitindo a construção de sistemas automatizados, confiáveis e transparentes.

1.4. Plataformas Populares para Contratos Inteligentes

A implementação de contratos inteligentes depende diretamente das plataformas de blockchain que oferecem suporte à criação, execução e validação desses contratos. Desde o surgimento do Ethereum, diversas blockchains surgiram com propostas específicas para otimizar desempenho, reduzir custos e melhorar a experiência de desenvolvedores e usuários. Nesta seção, destacam-se algumas das principais plataformas utilizadas atualmente.

1.4.1. Ethereum

O **Ethereum** é, sem dúvida, a plataforma mais consolidada e amplamente adotada para o desenvolvimento de contratos inteligentes. Lançado em 2015, o Ethereum introduziu o conceito de *Ethereum Virtual Machine* (EVM), um ambiente de execução que permite que qualquer nó da rede execute os contratos de forma padronizada.

A linguagem principal para programação no Ethereum é a **Solidity**, que permite escrever contratos com funcionalidades complexas. A plataforma também adota um sistema de taxas baseado em *gas*, o que torna sua utilização transparente e economicamente regulada.

Entre suas vantagens estão:

- Forte comunidade de desenvolvedores;
- Ferramentas maduras (Remix, Truffle, Hardhat);
- Compatibilidade com uma vasta gama de carteiras e dApps;
- Suporte a padrões como ERC-20 (tokens fungíveis) e ERC-721 (NFTs).

No entanto, o Ethereum enfrenta críticas relacionadas à escalabilidade e ao custo elevado de transações em momentos de congestionamento.

1.4.2. Binance Smart Chain (BSC)

A **Binance Smart Chain** surgiu como uma alternativa ao Ethereum, oferecendo maior velocidade e taxas mais baixas. É compatível com a EVM, o que facilita a migração de contratos existentes do Ethereum para a BSC sem necessidade de reescrita do código.

Seu foco principal está em aplicações DeFi e em permitir transações rápidas com baixa latência. Por ser uma blockchain mais centralizada, sua governança é frequentemente questionada, mas sua eficiência e integração com o ecossistema Binance são grandes atrativos.

1.4.3. Solana

A **Solana** destaca-se por sua arquitetura baseada em *Proof of History* (PoH), que permite processar milhares de transações por segundo com taxas extremamente baixas. Ao

contrário do Ethereum, a Solana não utiliza a EVM, e seus contratos são escritos majoritariamente em **Rust** ou **C**.

Essa plataforma tem sido amplamente utilizada para aplicações de alto desempenho, como marketplaces de NFTs e jogos em blockchain. Apesar disso, a complexidade do ambiente de desenvolvimento e as interrupções da rede em momentos críticos ainda são desafios a serem superados.

1.4.4. Cardano

A **Cardano** adota uma abordagem científica e baseada em revisão por pares. Seu modelo de contrato inteligente é alimentado pela linguagem funcional **Plutus**, que oferece segurança formal através de verificação matemática. Embora seja mais recente na implementação de contratos inteligentes, Cardano apresenta um ambiente promissor, com foco em sustentabilidade e escalabilidade.

Seu diferencial está no rigor acadêmico aplicado ao desenvolvimento da plataforma, o que a torna atrativa para projetos institucionais e de longo prazo.

1.4.5. Outras plataformas

Além das citadas acima, diversas outras blockchains vêm ganhando espaço, como:

- **Avalanche**: alta escalabilidade e compatibilidade com EVM;
- **Tezos**: contratos escritos em *Michelson*, com foco em governança on-chain;
- **Polygon**: solução de segunda camada para Ethereum, que reduz custos e aumenta a velocidade;
- **Near Protocol**: voltada à simplicidade para desenvolvedores e à eficiência energética.

Cada plataforma apresenta suas particularidades, vantagens e desafios, sendo a escolha ideal dependente dos requisitos do projeto, linguagem preferida, custo de transação, desempenho desejado e nível de descentralização necessário.

1.4.6. Principais Aplicações

Os contratos inteligentes têm se destacado como uma das tecnologias mais inovadoras no ecossistema blockchain, desempenhando um papel crucial na evolução de serviços digitais descentralizados. Sua capacidade de automatizar processos, garantir transparência, reforçar a segurança e eliminar intermediários vem impulsionando seu uso em uma ampla gama de setores econômicos e sociais. A seguir, detalham-se as principais aplicações dessa tecnologia, evidenciando seu impacto e potencial transformador.

- **Finanças Descentralizadas (DeFi)**: Uma das áreas que mais se beneficiou da implementação de contratos inteligentes é a das finanças descentralizadas (DeFi), que propõe um ecossistema financeiro aberto, sem a necessidade de intermediários

tradicionais como bancos ou corretoras. Plataformas DeFi utilizam contratos inteligentes para oferecer serviços como empréstimos colateralizados, onde usuários podem garantir ativos digitais como garantia e tomar empréstimos de forma automatizada; sistemas de staking, que recompensam usuários pela sua participação na segurança e manutenção da rede; yield farming, que permite a otimização de rendimentos através do fornecimento de liquidez a pools; e exchanges descentralizadas (DEX), que viabilizam a negociação peer-to-peer de criptoativos sem a necessidade de uma entidade central. Esses serviços não só ampliam a acessibilidade a produtos financeiros, especialmente em regiões sub-bancarizadas, como também promovem a autonomia dos usuários sobre seus próprios recursos. Um exemplo de DeFi é a Binance Smart chain(BSC): Embora a Binance seja amplamente conhecida como exchange centralizada, sua Binance Smart Chain (BSC) é uma blockchain pública que suporta contratos inteligentes e uma vasta gama de aplicações DeFi. Na BSC, projetos como *PancakeSwap* (DEX descentralizada) e *Venus Protocol* (empréstimos descentralizados) têm ganhado destaque. A BSC oferece taxas de transação mais baixas e alta velocidade em comparação com o Ethereum, o que atraiu um grande volume de usuários. A plataforma também suporta *yield farming*, staking e várias outras funcionalidades típicas de DeFi, consolidando-se como um ecossistema híbrido entre centralização e descentralização.

- **NFTs (Tokens Não Fungíveis)**: No campo da arte digital, entretenimento e propriedade intelectual, os NFTs (Non-Fungible Tokens) têm se consolidado como uma inovação disruptiva. Diferente dos tokens fungíveis como Bitcoin ou Ethereum, os NFTs representam ativos digitais únicos e indivisíveis, permitindo autenticar a originalidade e a propriedade de itens digitais. Contratos inteligentes são fundamentais nesse ecossistema, pois gerenciam a criação (minting), transferência e rastreamento de NFTs de forma segura e transparente. Além disso, possibilitam a implementação de regras automatizadas, como royalties programáveis, garantindo que criadores recebam uma porcentagem automaticamente a cada nova venda secundária. Essa funcionalidade inaugura um novo modelo de remuneração contínua, fortalecendo a sustentabilidade econômica para artistas e desenvolvedores de conteúdo. O *OpenSea* é atualmente a maior e mais popular plataforma de NFTs do mundo. Lançada em 2017, ela permite que qualquer usuário crie, compre, venda e explore ativos digitais tokenizados, incluindo arte digital, itens de jogos, música e colecionáveis virtuais. A plataforma opera principalmente na rede Ethereum e foi pioneira ao oferecer um marketplace abrangente para NFTs, promovendo milhões de transações e abrindo portas para artistas digitais e criadores de conteúdo em escala global.

- **Cadeia de Suprimentos (Supply Chain)**: A aplicação de contratos inteligentes na gestão da cadeia de suprimentos (supply chain) tem revolucionado a forma como produtos são monitorados ao longo de todo o seu ciclo de vida. Com a tecnologia blockchain, é possível registrar de maneira imutável e auditável cada etapa da jornada de um produto — desde a produção, passando pelo transporte, até a entrega ao consumidor final. Os contratos inteligentes automatizam processos como liberações de pagamento e verificação de conformidade em tempo real, tornando o sistema mais eficiente e confiável. Isso não apenas reduz riscos de fraudes e erros,

mas também melhora a transparência e a rastreabilidade, fatores cada vez mais valorizados por consumidores e parceiros comerciais preocupados com práticas éticas e sustentabilidade. A *VeChain* é uma das principais plataformas especializadas em rastreamento e gerenciamento de cadeias de suprimentos. Utilizando sensores IoT e blockchain, a VeChain permite registrar cada etapa da cadeia logística, desde a produção até a entrega final. Empresas de setores como alimentos, saúde e indústria automotiva já utilizam VeChain para garantir autenticidade, prevenir fraudes e melhorar a eficiência operacional. Um exemplo notável é a parceria da VeChain com a empresa de luxo *LVMH* para autenticar produtos de alta-costura.

- **Governança Descentralizada (DAOs)**: As Organizações Autônomas Descentralizadas (DAOs) representam um modelo inovador de governança corporativa e comunitária. Baseadas em contratos inteligentes, as DAOs operam sem estruturas hierárquicas tradicionais, permitindo que regras operacionais e mecanismos de tomada de decisão sejam codificados e executados automaticamente na blockchain. Membros dessas organizações podem propor mudanças, votar em decisões importantes e contribuir para a evolução da entidade de forma democrática e transparente. Esse modelo tem ganhado ampla adoção em projetos open source, plataformas DeFi e iniciativas de crowdfunding, onde a confiança é construída não através de líderes centrais, mas por meio da transparência e imutabilidade do código. As DAOs demonstram o potencial de criar sistemas de governança mais inclusivos, resilientes e responsivos às necessidades da comunidade. A plataforma *Aragon* permite a criação e gestão de Organizações Autônomas Descentralizadas (DAOs) de forma simples e segura. Fundada em 2016, Aragon fornece ferramentas para governança comunitária, votação, gestão de tesouraria e criação de regras organizacionais codificadas em contratos inteligentes. Usada amplamente em projetos de código aberto e iniciativas comunitárias, Aragon visa democratizar a governança digital, eliminando estruturas hierárquicas rígidas e promovendo maior transparência e participação dos membros.

1.4.7. Desafios e Limitações

Embora os smart contracts ofereçam benefícios significativos, sua adoção e implementação em larga escala ainda enfrentam diversos desafios e limitações. Tais obstáculos precisam ser cuidadosamente considerados por desenvolvedores, usuários e legisladores para garantir a segurança e a viabilidade dessa tecnologia. A seguir, são apresentados os principais desafios associados aos contratos inteligentes:

- **Vulnerabilidades e hacks**: Apesar da robustez criptográfica das blockchains, os smart contracts não estão imunes a falhas de programação. Um exemplo emblemático foi o *DAO Hack*, ocorrido em 2016, quando uma vulnerabilidade no código de um contrato inteligente permitiu que um atacante drenasse cerca de US$ 60 milhões em Ether da DAO (Decentralized Autonomous Organization). Esse incidente destacou a importância de auditorias rigorosas e práticas de desenvolvimento seguro, evidenciando que um erro no código pode ter consequências financeiras devastadoras.

- **Escalabilidade e custos de gas**: À medida que o uso de smart contracts cresce, redes blockchain — especialmente aquelas baseadas no modelo Proof of Work, como a Ethereum até sua transição para Proof of Stake — enfrentam problemas de congestionamento. Esse aumento de demanda resulta em elevação dos custos de transação (taxas de *gas*), tornando a execução de contratos complexos economicamente inviável para muitos usuários. A escalabilidade continua sendo um dos maiores desafios para viabilizar aplicações em massa, motivando o desenvolvimento de soluções como redes de camada 2 (*Layer 2*) e blockchains mais eficientes.

- **Aspectos regulatórios e incertezas legais**: A natureza descentralizada e global dos smart contracts gera incertezas quanto ao seu enquadramento jurídico. Questões relacionadas à validade legal dos contratos inteligentes, jurisdição em casos de disputa, e obrigações fiscais ainda estão em debate em muitos países. A ausência de regulamentações claras pode desencorajar empresas e instituições a adotarem plenamente essa tecnologia, especialmente em setores que exigem forte compliance legal, como finanças e saúde. Além disso, a falta de mecanismos para reverter contratos executados automaticamente levanta discussões sobre como proteger consumidores e partes vulneráveis.

1.5. Estudo de Caso: Sistema de Contratos Inteligentes para Aluguel

Figure 1.3. Tela principal

1.6. Contexto

Imaginemos uma plataforma onde proprietários (locadores) e inquilinos (locatários) gerenciam contratos de aluguel de forma transparente usando a blockchain Ethereum. O sistema permite:

- Registrar contratos entre locador e locatário;
- Depositar caução (depósito de segurança);
- Pagar aluguel mensal;
- Sacar caução ao fim do contrato;

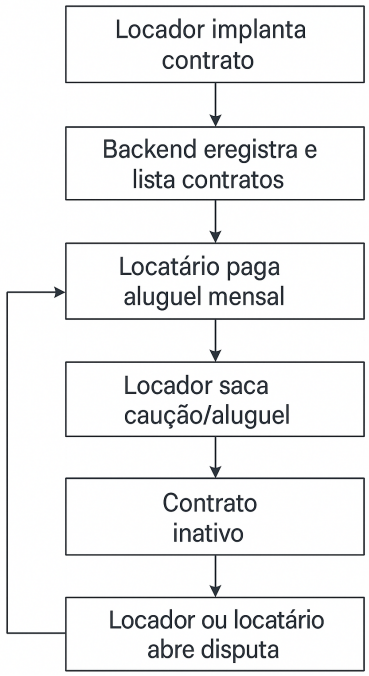


Figure 1.4. Fluxo de execução

- Abrir disputas em caso de desacordos;
- Encerrar contratos.

A vantagem do uso do contrato inteligente é garantir regras imutáveis e automação financeira segura, sem intermediários.

1.7. Parte 1 — Contrato Inteligente em Solidity

1.7.1. Objetivo

Criar um contrato inteligente que:

- Armazene as partes envolvidas (locador e locatário);
- Controle o depósito de caução e o pagamento de aluguel;
- Permita ações específicas apenas para locador ou locatário;
- Controle o status do contrato (ativo, encerrado, em disputa).

1.7.2. Funcionamento

O contrato é criado pelo locador, que passa o endereço do locatário e o valor do aluguel mensal. O funcionamento segue as seguintes etapas:

1. O locatário deposita a caução (em ether);

2. O locatário paga o aluguel mensal, enviando o valor exato;
3. O locador pode sacar a caução apenas após o encerramento do contrato;
4. O contrato pode ser encerrado pelo locador;
5. Qualquer uma das partes pode abrir uma disputa para resolução fora da blockchain.

1.8. Parte 2 — Backend Django com Web3.py

1.8.1. Objetivo

- Conectar-se a um nó Ethereum local (ex: Ganache);
- Consultar o estado do contrato (locador, locatário, status);
- Construir transações para as ações solicitadas pelo frontend;
- Executar ou simular transações.

1.8.2. Desafios Técnicos

- Assinatura e envio das transações com chaves privadas, que não devem permanecer no frontend;
- Atualizar o status do contrato exibido no frontend após cada ação;
- Segurança e validação dos dados fornecidos pelos usuários.

1.8.3. Estrutura Básica

- Uma lista (ou banco de dados) de contratos disponíveis;
- Função para renderizar a lista de contratos com seus respectivos status;
- Endpoint para receber ações do usuário, construir a transação correspondente e enviá-la para a blockchain.

1.9. Parte 3 — Frontend Django com Bootstrap

1.9.1. Objetivo

- Apresentar a lista de contratos com seus dados;
- Exibir status legível (*Ativo*, *Encerrado*, *Em Disputa*);
- Disponibilizar formulários para executar ações como:
  - Depositar caução;
  - Pagar aluguel;
  - Sacar caução;
  - Encerrar contrato;
  - Abrir disputa.
- Interface limpa e responsiva utilizando Bootstrap.

1.9.2. Fluxo de Uso

1. O usuário visualiza a lista de contratos;

2. Preenche o endereço Ethereum da carteira que será usada na transação e o valor (se aplicável);

3. Clica no botão da ação desejada;

4. O backend processa a solicitação, envia a transação e atualiza o estado exibido no frontend.

1.9.3. Fluxo Completo de uma Interação Típica

- **Criação do contrato:** o locador implanta o contrato passando o endereço do locatário e o valor do aluguel mensal;

- **Listagem no frontend:** o backend (Django) lê os dados do contrato via Web3.py e mostra o contrato com status *ativo*;

- **Locatário deposita caução:** o usuário insere seu endereço e valor, clica em *Depositar Caução*; o backend cria e envia a transação, que atualiza a variável `caucao`;

- **Locatário paga aluguel:** mensalmente, o locatário paga o aluguel por meio do botão *Pagar Aluguel*;

- **Locador encerra contrato:** ao final, o locador chama a função `encerrarContrato`, mudando o status para *encerrado*;

- **Locador saca caução:** com o contrato encerrado, o locador pode sacar a caução depositada;

- **Disputa:** em caso de desacordo, qualquer parte pode abrir uma disputa para resolução fora da blockchain.

1.10. Benefícios Práticos do Sistema

- **Transparência:** todos os pagamentos e ações são registrados na blockchain pública (ou testnet), proporcionando rastreabilidade;

- **Segurança:** regras imutáveis garantem que nenhuma das partes aja fora dos termos previamente acordados;

- **Automação:** transferências financeiras e controle de status são automatizados e não dependem de intermediários;

- **Facilidade:** com um frontend amigável, os usuários não precisam interagir diretamente com a blockchain.

1.10.1. Conclusão

O **Sistema de Contratos Inteligentes para Aluguel** ilustra de maneira prática como contratos inteligentes podem ser integrados a sistemas web modernos, promovendo segurança e descentralização em aplicações financeiras. Esse estudo de caso serve como uma excelente introdução aos conceitos de blockchain, Ethereum, smart contracts e integração com carteiras digitais como a MetaMask.

Considerações Finais

O minicurso **"Introdução aos Smart Contracts"** teve como proposta principal aproximar os participantes do universo da tecnologia blockchain por meio de experiências práticas e contextualizadas. Ao longo do curso, foram apresentados os fundamentos técnicos dos contratos inteligentes, suas aplicações reais e os principais desafios associados ao seu desenvolvimento.

Por meio do estudo de caso do **Sistema de Contratos Inteligentes para Aluguel**, os participantes puderam vivenciar todas as etapas de criação de um contrato inteligente: desde a definição de regras de negócio, a implementação em *Solidity*, até a construção de um sistema web funcional com *Django* e *Web3.py*. O uso do *Ganache* como ambiente local de testes e da carteira *MetaMask* para autenticação reforçou a integração entre o front-end e a blockchain, proporcionando um cenário completo e realista.

Espera-se que os conhecimentos adquiridos durante o minicurso sirvam como base sólida para que os participantes avancem em seus estudos e desenvolvam soluções descentralizadas mais robustas, seguras e inovadoras. A descentralização representa uma mudança de paradigma tecnológica, e a compreensão prática dos contratos inteligentes é um passo essencial para qualquer profissional que deseja atuar nessa nova fronteira da computação.

References

[1] H. Taherdoost, "Smart contracts in blockchain technology: A critical review," *Information*, vol. 14, no. 2, p. 117, 2023.

[2] S. Nakamoto, "Bitcoin whitepaper," 2008. [Online]. Available: *https://bitcoin.org/bitcoin.pdf*. Accessed: Jul. 17, 2019.

[3] H. Watanabe, S. Fujimura, A. Nakadaira, Y. Miyazaki, A. Akutsu, and J. Kishigami, "Blockchain contract: Securing a blockchain applied to smart contracts," in *Proc. 2016 IEEE Int. Conf. on Consumer Electronics (ICCE)*, pp. 467–468, 2016.

[4] A. M. Antonopoulos and G. Wood, *Mastering Ethereum: Building Smart Contracts and DApps*. O'Reilly Media, 2018.

[5] W. Metcalfe *et al.*, "Ethereum, smart contracts, DApps," *Blockchain and Crypt Currency*, vol. 77, pp. 77–93, 2020. Springer Singapore.

[6] N. Szabo, "Smart contracts: Building blocks for digital markets," 1996.

[7] Buterin, V. (2013). *Ethereum Whitepaper: A Next-Generation Smart Contract and Decentralized Application Platform*.

[8] Wood, G. (2014). *Ethereum: A Secure Decentralized Generalized Transaction Ledger*.

[9] Carvalho, C. A., & Ávila, L. V. (2020). *A Tecnologia Blockchain Aplicada aos Contratos Inteligentes*.

[10] Antonopoulos, A. M., & Wood, G. (2018). *Mastering Ethereum: Building Smart Contracts and DApps*. O'Reilly Media.

[11] Binance Smart Chain. *https://www.bnbchain.org*, acesso em: Maio de 2025.

[12] OpenSea. *https://opensea.io*, acesso em: Maio de 2025.

[13] VeChain. *https://www.vechain.org*, acesso em: Maio de 2025.

[14] Aragon. *https://aragon.org*, acesso em: Maio de 2025.

[15] Universal Market Access. *https://uma.xyz/*, acesso em: Maio de 2025.

[16] Blockchain Ganache. *https://www.blockchain-council.org/blockchain/ganache-blockchain-all-you-need-to-know/*, acesso em: Maio de 2025.

[17] Metamask. *https://criptonizando.com/aprenda/metamask-o-que-e/*, acesso em: Maio de 2025.

[18] Solidity. *https://docs.soliditylang.org/en/v0.8.30/*, acesso em: Maio de 2025.

[19] Web3 python. *https://web3py.readthedocs.io/en/stable/*, acesso em: Maio de 2025.

[20] Django frammework. *https://docs.djangoproject.com/pt-br/5.2/*, acesso: em Maio de 2025

Chapter 2

Desenvolvimento de Experiência Digitais Imersivas através do Mapeapento de Vídeo em Tempo Real com Redes Neurais

Héder Pereira Rodrigues Silva, Iallen Gábio de Sousa Santos

Abstract

Gesture-based interactivity, powered by technologies such as Kinect and computer vision systems, allows users to control digital content without touch, making it useful for applications like interactive storefronts and public games. To enable this, computer vision algorithms and neural networks detect and interpret body movements in real time, often accelerated by WebGL. Tools like ml5.js, Matter.js, and P5.js facilitate the development of these web applications, enabling the creation of immersive and accessible experiences with gesture recognition, physics simulation, and integrated visual rendering.

Resumo

A interatividade com gestos permite que usuários controlem conteúdos digitais sem toque, tornando-se útil em aplicações como vitrines interativas e jogos públicos. Para viabilizar isso, algoritmos de visão computacional e redes neurais detectam e interpretam movimentos corporais em tempo real, geralmente com aceleração via WebGL. Ferramentas como ml5.js, Matter.js e P5.js facilitam o desenvolvimento web dessas aplicações, permitindo a criação de experiências imersivas acessíveis, com reconhecimento de gestos, simulação física e visualização gráfica integrada. Neste capítulo é apresentado um panorama geral sobre a aplicação deste tipo de tecnologia na construção de experiências digitais imersivas.

2.1. Introdução

Desde a década de 1980, a utilização de *Digital Signage* - ou Sinalização Digital - vem sendo priorizada em relação aos avisos tradicionais para a divulgação de marcas empresariais, pois ela os supera ao facilitar atualizações frequentes e oportunas, aumentando a precisão e permitindo o fornecimento de informações altamente dinâmicas que, de outra forma, não seriam disponibilizadas [Davies et al. 2022].

Estudos indicam que a sinalização digital pode aumentar significativamente o engajamento do público e influenciar positivamente as decisões de compra, destacando-se como uma ferramenta poderosa no arsenal do marketing moderno [Asif et al. 2024]. Na atualidade, essa tecnologia evoluiu para uma abordagem que tem ganhado destaque: o uso de sinalização digital interativa, como monitores, totens e painéis de LED, amplamente utilizados em ambientes como shoppings, ruas comerciais e supermercados (Figura 2.1). Essa interatividade vem sendo implementada por meio de telas *touchscreen* e/ou QR Codes, porém já existem formas de interação mais imersivas, como a utilização de rastreamento ocular [Chung 2017] e leitura de gestos de mão [Chen et al. 2009].

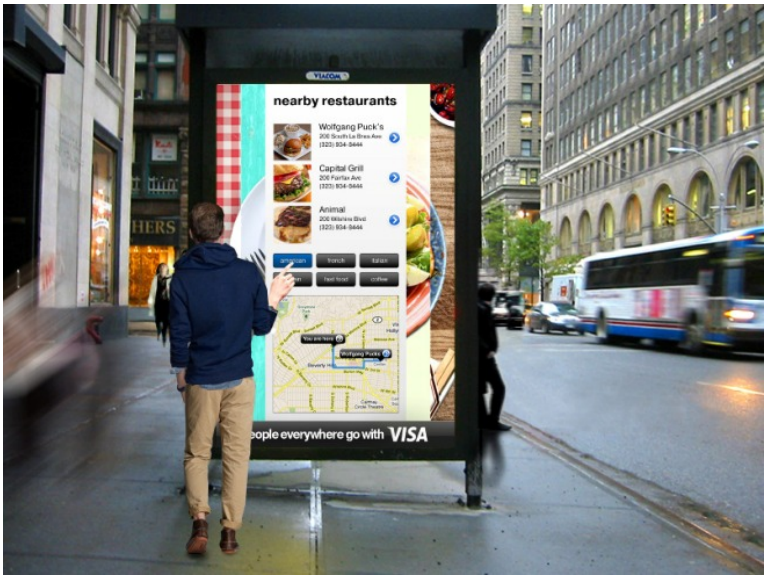


Figure 2.1. Representação de um Totem Digital com uma aplicação interativa. [Digital 2025]

O desenvolvimento dessas aplicações imersivas de sinalização digital atualmente é possível devido à utilização de algoritmos de aprendizado de máquina - para que o sistema de reconhecimento de gestos atenda aos requisitos em termos de precisão, desempenho em tempo real e robustez [Chen et al. 2009] - juntamente com ferramentas como o WebGL [Parisi 2012] que permitem a renderização de gráficos 3D interativos diretamente nos navegadores, sem a necessidade de plugins adicionais. Um exemplo de plataforma que utiliza o WebGL para o desenvolvimento de softwares de *digital signage* é a BrightSign [BrightSign LLC 2025].

Além disso, o desenvolvimento de aplicações interativas se tornou mais acessível para desenvolvedores menos experientes por meio de bibliotecas de JavaScript que facilitam a criação de gráficos, animações, objetos com propriedades físicas e uso de ferramentas de IA.

Diante desse panorama, este capítulo tem como objetivo apresentar as diferentes maneiras de se utilizar as aplicações interativas, introduzir ao leitor conceitos de *Machine Learning*, Visão Computacional e *design* de aplicações comerciais. Neste contexto, introduzir as bibliotecas **Ml5.js**, **Matter.js** e **P5.js** que serão utilizadas em um estudo de caso de criação de uma Experiência Imersiva utilizando o Mapeamento de posições de mão

para a interação com o usuário.

As próximas seções estão organizadas da seguinte forma:

Seção 2 Apresenta a evolução e os fundamentos da interação gestual em sinalização digital, abordando desde tecnologias pioneiras até exemplos práticos de aplicação.

Seção 3 Discute a arquitetura e os requisitos de software para sistemas de digital signage interativo, incluindo criação de conteúdo, agendamento, display e dispositivos móveis.

Seção 4 Introduz o conjunto de bibliotecas ml5.js, Matter.js e P5.js, mostrando como elas facilitam o reconhecimento de gestos, a simulação física e a renderização gráfica em aplicações web.

Seção 5 Demonstra, na forma de um tutorial prático, a integração das três bibliotecas para criar um jogo interativo baseado em gestos, desde a configuração do ambiente até a lógica de vitória.

2.2. Aplicações Interativas no contexto da sinalização digital

A interação com o público tem sido explorada no contexto de telas públicas há muitos anos. De fato, o trabalho com telas sensíveis ao toque começou na década de 1960 e tem continuado em ritmo acelerado desde então [Davies et al. 2022].

Uma explicação para a eficiência dessa estratégia pode ser explicada pelo fenômeno chamado *Honeypot Effect*(Efeito Pote de Mel) [Brignull and Rogers 2003], que descreve o efeito de atração que as pessoas sentem ao ver outras interagindo com um dispositivo de sinalização interativo, como pode ser observado na Figura 2.2.

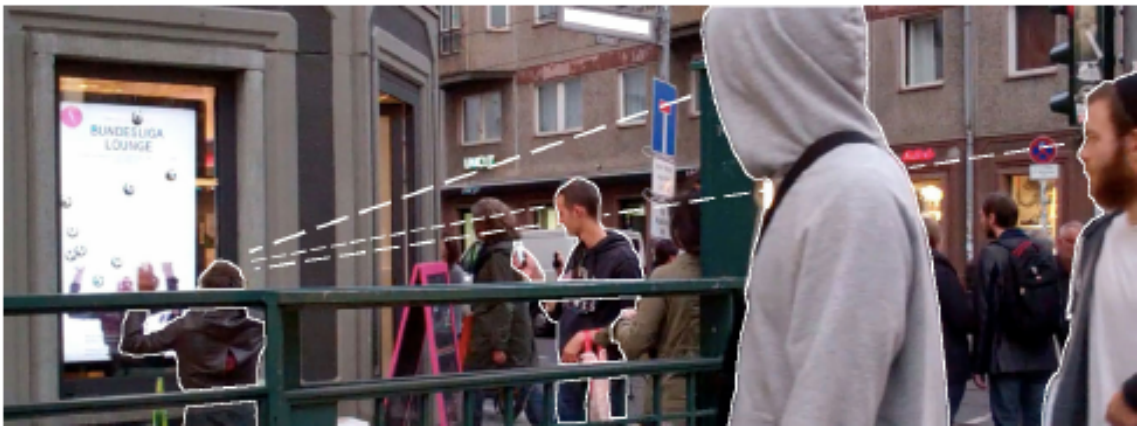


Figure 2.2. "O Efeito Pote de Mel: Quando as pessoas percebem alguém fazendo gestos incomuns, elas se posicionam de forma que tanto a tela quanto a pessoa que está interagindo sejam vistas. Muitas vezes, elas também se posicionam de forma que não sejam representadas na tela". [Müller et al. 2012]

Este fenômeno ocorre principalmente em locais públicos: um indivíduo, ao observar outra pessoa interagindo com uma tela publicitária, tem mais chances de ter a sua

atenção desviada para o local e se dirigir para mais perto da sinalização digital. Esse efeito continua e aumenta quanto mais pessoas estiverem próximas do dispositivo, tornando-o eficiente para chamar a atenção do público para um anúncio.

Um exemplo claro do efeito "pote de mel" pode ser observado no *CityWall* [HIIT 2025], um painel multitouch instalado no centro de Helsinque, capital da Finlândia. Peltonen et al. verificaram que, quando alguém já está interagindo com a tela, cerca de 19% dos passantes a percebem; em contrapartida, na ausência de um usuário ativo, alguns pedestres não identificam sua capacidade interativa [Peltonen et al. 2008].

2.2.1. Interatividade com Toque

Por muito tempo a interatividade nas aplicações interativas foi implementada por meio de toque. Um exemplo recente disso são os Quiosques de autoatendimento: são painéis digitais sensíveis ao toque instalados em pontos de venda ou restaurantes, permitindo que o cliente faça pedidos, escolha opções e pague sem auxílio de um atendente. Por exemplo, o McDonald's introduziu quiosques touchscreen em suas lojas, onde o cliente visualiza o cardápio, personaliza itens e confirma o pedido na própria tela, reduzindo erros e filas [McDonald's Corporation 2020].

Outra aplicação dessa forma de interatividade são estações interativas de personalização de produtos em *showrooms*. Por exemplo, numa concessionária Seat [SEAT S.A. 2021], foram criados pedestais com telas touchscreen ao lado de cada carro, onde o visitante pode "brincar" de configurador: escolher cores, acessórios e até registrar um depósito no próprio painel.

2.2.2. Interatividade com Gestos

Com o surgimento de tecnologias de visão computacional como o *Kinect* [Microsoft 2010], a utilização de interação com gestos começou a ser utilizada como alternativa ao toque, em situações em que a liberdade de movimento é mais conveniente ou permite interações que não são possíveis apenas por meio de *touchscreens*.

Um exemplo de aplicação utilizada publicamente é o projeto *StrikeAPose* [Walter et al. 2013], que criou um jogo baseado em simulação física para motivar os transeuntes a interagir. Os passantes viam sua imagem espelhada na tela e podiam interagir com ela para manipular cubos virtuais. Ao posicionar os cubos em áreas-alvo específicas, eles acumulavam pontos. Este trabalho foi utilizado para explorar formas de ensinar novos gestos de interação aos usuários.

Outras formas de explorar a imersividade da interatividade com gestos são apontadas pela OnSign TV [TV 2023]: uma vitrine pode exibir um banner "chegue mais perto" e, ao detectar uma pessoa dentro de um raio definido (por sensor de movimento), mudar automaticamente o layout para mostrar informações detalhadas ou opções interativas. Por exemplo, ao detectar um cliente perto da tela, o sistema pode exibir especificações do produto em destaque ou ativar um menu de toque.

Após explorar as aplicações e o potencial do *Digital Signage* interativo, a seção seguinte se dedicará a detalhar os processos de implementação dessas soluções, bem como os principais recursos de software que viabilizam a criação de experiências imersivas.

2.3. Como é desenvolvida uma sinalização imersiva?

Um erro frequente é subestimar a complexidade dos sistemas de digital signage interativos [Davies et al. 2022]. Esses ambientes exigem um conjunto robusto de funcionalidades, que vão desde a criação e publicação de conteúdo até o gerenciamento remoto de dispositivos, passando pelo agendamento automatizado de exibições e pelo monitoramento em tempo real da interação dos usuários.

[Davies et al. 2022] representam a arquitetura de softwares de sinalização em 4 segmentos: Criação de conteúdo, Agendamento, Display e Dispositivos móveis - representados na Figura 2.3. Dado o contexto e o foco nas aplicações interativas, todo o conteúdo deste capítulo estará incluído nos segmentos de **Criação de conteúdo** e de **Display**. Estes segmentos são, respectivamente, responsáveis por fornecer as ferramentas necessárias para a criação do conteúdo que irá ser exibido e pelas operações relacionadas à interatividade.

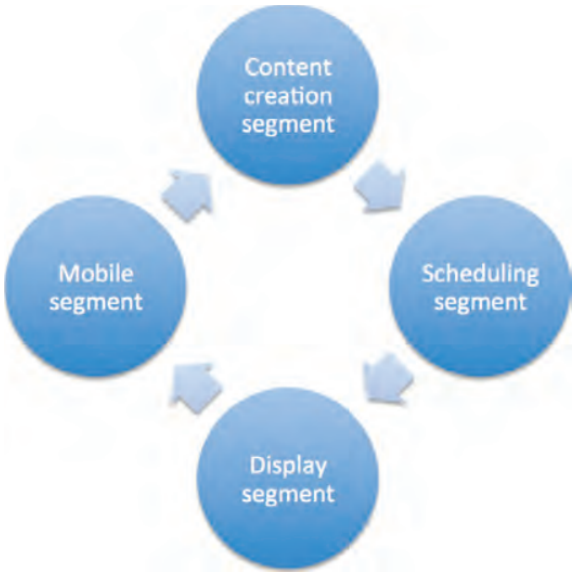


Figure 2.3. Entendendo a arquitetura de sistemas de sinalização. [Davies et al. 2022]

2.3.1. Criação de Conteúdo

Tradicionalmente, os sistemas consistiam em um software dedicado instalado em computadores conectados ao display ou diretamente incorporados neles. Além disso, também precisava-se de um software associado para a criação e gerenciamento do conteúdo a ser exibido [Davies et al. 2022].

Com o avanço da Web, especialmente em relação à reprodução de vídeos em alta qualidade e à capacidade de renderizar aplicações complexas diretamente no navegador, os sistemas de sinalização digital passaram a adotar um novo modelo: os displays agora executam navegadores que renderizam a aplicação localmente ou se conectam a um servidor dedicado para obter o conteúdo e a lógica do sistema.

Dessa forma, na atualidade, a produção de conteúdo para sinalização digital assemelha-se à de aplicações web. Por isso, não são necessárias ferramentas específicas para esse fim: os desenvolvedores costumam empregar soluções genéricas para criar

imagens, sons, vídeos, páginas e interações. As ferramentas que serão demonstradas neste capítulo serão aprofundadas na Seção 2.4.

2.3.2. Display

O conteúdo que é produzido precisa ser devidamente exibido na tela de sinalização; além disso, as interações que a aplicação irá utilizar devem ser gerenciadas, para que elas ocorram de forma precisa. Essa é a função do segmento de display, ele se encarrega de como funcionará a exibição do conteúdo.

No caso abordado na seção anterior, em que o display se conecta a um servidor dedicado, essa exibição será simplesmente a execução de um navegador, porém, em sistemas onde os nós de exibição têm mais autonomia local, algumas decisões de agendamento e exibição podem ser realizadas localmente, por exemplo, respondendo a eventos de sensores locais, e o conteúdo pode ser obtido de um cache local [Davies et al. 2022].

Esse segmento também oferece suporte à interatividade com o usuário; como visto na Seção 2.2, ela ocorre principalmente de duas formas: via toque em telas touch — cujo input é capturado como clique de mouse e repassado à aplicação — ou via gestos, em que poses corporais, manuais ou faciais são mapeadas e convertidas em comandos específicos para o sistema. Com o fim de se aprofundar nas experiências imersivas, trataremos com mais detalhes de como é feito esse processo de mapeamento e leitura de gestos.

A interação por gestos em sistemas de sinalização digital depende fundamentalmente de visão computacional. Esse campo da inteligência artificial surgiu nos anos 1960 (com trabalhos pioneiros como o de Lawrence Roberts no MIT em 1963 [de Oliveira and de Melo 2022]) e visa ensinar máquinas a interpretar imagens e vídeos do mesmo modo que o olho humano. Em termos simples, define-se visão computacional como um sistema computacional treinado para captar e analisar imagens do mundo real por meio do reconhecimento de padrões.

Na prática, algoritmos de visão computacional extraem características visuais de quadros de vídeo ou fotos (como contornos, texturas ou regiões de interesse) e as usam para tarefas de reconhecimento. Por exemplo, a estimativa de pose identifica pontos-chave do corpo humano (juntas, mãos, etc.) em imagens para entender a postura e o movimento.

No contexto de gestos, uma câmera captura sequências de quadros com o usuário realizando movimentos; algoritmos de detecção (baseados em segmentação, cor de pele ou filtros especializados) processam cada quadro, identificam as regiões correspondentes às mãos ou partes do corpo e calculam suas coordenadas espaciais (x,y em cada frame), como pode ser observado na Figura 2.4. Assim, gestos visuais são convertidos em pontos rastreáveis ou vetores de posição que podem ser usados em aplicações interativas. Essa transformação de vídeo em coordenadas de cursor ou em variáveis de controle é viabilizada justamente pelos métodos de visão computacional implantados no software de sinalização.

Para interpretar corretamente esses padrões visuais, o uso de aprendizado de máquina torna-se essencial. Redes neurais profundas são treinadas para reconhecer gestos a partir de grandes conjuntos de imagens etiquetadas, dispensando regras manuais de extração de características. Durante o treinamento, essas redes aprendem hierarquias de recursos (de bordas simples até formas complexas) que distinguem diferentes gestos. Como

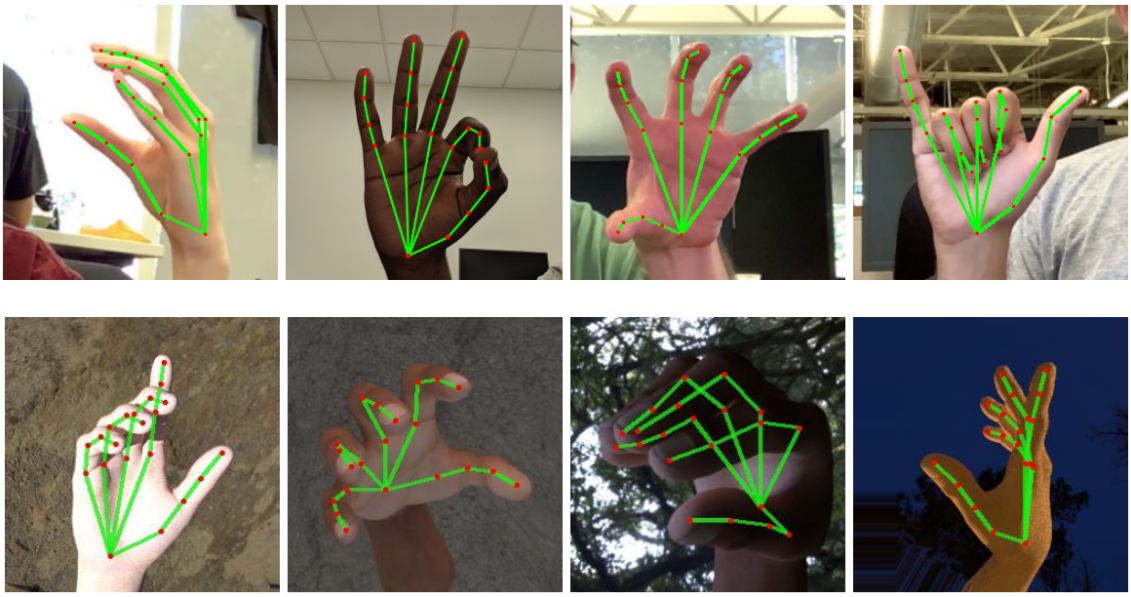


Figure 2.4. Mapeamento de gestos de mão realizado por um algoritmo de visão computacional. [Coldewey 2019]

observado por IBM, redes neurais convolucionais são computacionalmente intensivas e normalmente exigem GPUs para treinar modelos de visão computacional eficientemente [IBM Brasil 2023]. Mesmo para inferência em tempo real, deve-se contar com capacidade de processamento considerável, pois a rede precisa processar dezenas de frames por segundo. Em resumo, o aprendizado de máquina fornece os modelos que detectam e classificam gestos, mas demanda alto poder computacional (memória e processamento) tanto no treinamento quanto, em certa medida, na execução desses modelos.

Uma solução para prover capacidade computacional no ambiente web é o uso de WebGL [Parisi 2012]. WebGL é uma API JavaScript que expõe a GPU do dispositivo via contexto de desenho no navegador. Inicialmente criado para gráficos 3D, o WebGL vem sendo usado por bibliotecas de Machine Learning (ML) em JavaScript para computação numérica pesada - alguns exemplos são o **Tensorflow.js** [TensorFlow.js Team 2023], o **ml5.js** [ml5.js contributors 2018] e o **ONNX.js** [Microsoft 2019]. Na prática, tensores (arranjos multidimensionais de dados) são armazenados como texturas na GPU e operações matemáticas (como multiplicação de matrizes) são implementadas como shaders em WebGL.

Dessa forma, o processamento de redes neurais que rodariam originalmente na CPU passa a ser acelerado pela GPU, que é otimizada para cálculos paralelos. Segundo a documentação do TensorFlow.js, o backend WebGL pode ser até cem vezes mais rápido que a implementação em CPU, uma vez que delega os cálculos aos recursos gráficos.

Um exemplo prático desse conceito é o **TensorFlow.js** [TensorFlow.js Team 2023]. Essa biblioteca de código aberto do Google permite treinar e executar modelos de aprendizado de máquina inteiramente em JavaScript, tanto no navegador quanto em Node.js. Em seu modo de uso no browser, o TensorFlow.js seleciona automaticamente o backend WebGL: ele armazena tensores como texturas de GPU e compila operações tensorais em

shaders acelerados. Isso acelera significativamente tarefas de inferência de redes neurais (como reconhecimento de gestos ou classificação de imagens) sem depender de servidores externos. Em suma, essa ferramenta demonstra como técnicas de aprendizado de máquina podem ser integradas em aplicações web interativas, aproveitando os recursos gráficos do dispositivo cliente para obter desempenho.

Essa abordagem baseada em navegador traz vantagens específicas para sinalização digital interativa. Como navegadores web são universais e baseados em padrões (HTML, CSS, JavaScript), não há necessidade de software proprietário ou hardware dedicado, o que simplifica a escalabilidade e a manutenção do sistema. Conteúdo e lógica podem ser atualizados remotamente: por exemplo, um servidor central pode distribuir scripts e modelos de ML para múltiplos players. Além disso, atualmente, a sinalização digital, por meio da web, integra-se naturalmente aos sistemas existentes (APIs, bancos de dados, serviços em nuvem), permitindo criar conteúdo dinâmico nas páginas de sinalização.

Em termos práticos, essa arquitetura reduz custos operacionais (evita deslocamentos para atualizar cada dispositivo fisicamente), facilita a manutenção (por usar frameworks web comuns) e garante que o sistema seja escalável e compatível com o ecossistema web do ambiente corporativo.

2.4. Desenvolvimento de Aplicações Interativas Simplificado

Na seção anterior, foi discutido como aplicações com reconhecimento de gestos podem ser desenvolvidas para a web e utilizadas em contextos de sinalização digital. No entanto, esse tipo de desenvolvimento pode exigir um conhecimento mais aprofundado, especialmente em áreas como aprendizado de máquina e visão computacional. Considerando essa complexidade, esta seção apresentará três bibliotecas que se complementam e cuja simplicidade possibilita que desenvolvedores iniciantes — ou com pouca experiência em softwares de sinalização digital — consigam criar conteúdos interativos e imersivos de forma acessível.

Essas bibliotecas são: a **ml5.js**, a *Matter.js* e a *P5.js* — aqui referidas como o conjunto MMP — compartilham o propósito de oferecer, em suas áreas de especialização, ferramentas acessíveis e de fácil compreensão para desenvolvedores iniciantes. Com um conhecimento introdutório sobre o MMP, torna-se viável construir aplicações interativas simples, mas eficazes. Em poucos passos, é possível implementar reconhecimento de gestos, simulações físicas com objetos manipuláveis por movimentos corporais e a renderização visual desses elementos com o design definido pelo desenvolvedor. Esses três componentes formam uma base completa para a criação de experiências digitais imersivas e responsivas.

2.4.1. Ml5.js

Na Seção 2.3.2 o **TensorFlow.js** foi apresentado como uma ferramenta capaz de implementar modelos de machine learning para o ambiente web. No entanto, seu uso costuma exigir conhecimentos em álgebra linear, estatística e arquiteturas de redes neurais [Shiffman et al. 2018]. Assim, o **Ml5.js** surge como uma camada de abstração amigável sobre o TensorFlow.js para simplificar o uso de aprendizado de máquina em aplicações web. Ele foi projetado para tornar o ML acessível a artistas, estudantes e desenvolvedores não espe-

cialistas, oferecendo uma interface de alto nível com poucas linhas de código. Assim, em vez de manipular diretamente tensores e operações matemáticas, o programador carrega os modelos pré-treinados por meio de chamadas simples, permitindo focar no aspecto criativo do projeto, ainda com a possibilidade de realizar novos treinamentos para os modelos.

Entre os modelos disponibilizados pelo ml5.js, destacam-se especialmente aqueles voltados à visão computacional interativa. A biblioteca integra classes como HandPose, Pose/BodyPose (às vezes chamada de PoseDetection) e FaceMesh, correspondendo a modelos do MediaPipe/TensorFlow para rastreamento de mãos, detecção de pose corporal e de pontos faciais, respectivamente. Por exemplo, o modelo PoseNet (usado no BodyPose) estima a pose humana identificando as posições de pontos-chave do corpo (como ombros, cotovelos, quadris) em uma imagem, como pode ser visto na Figura 2.5.

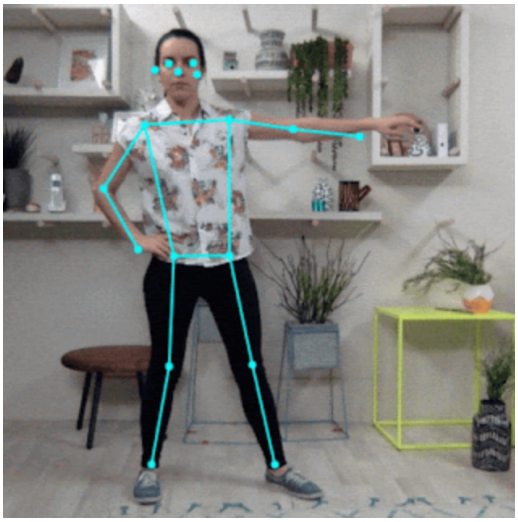


Figure 2.5. Mapeamento de pose de corpo por meio de pontos feito pelo modelo PoseNet, base para o BodyPose [TensorFlow.js Team 2023]

O HandPose realiza, em tempo real, a detecção de uma mão inteira: primeiro, um detector de palma localiza a mão, depois um modelo de landmarks calcula 21 pontos-chave nos dedos e na palma da mão (Figura 2.6).

E O FaceMesh estima 468 pontos de referência faciais em 3D, mapeando detalhadamente a geometria completa do rosto (olhos, nariz, boca e contornos). Cada uma dessas classes carrega internamente o modelo pré-treinado correspondente e expõe funções simples (como .predict() ou callbacks) que retornam as coordenadas dos landmarks detectados a cada frame de vídeo. Na prática, os modelos de visão do ml5.js operam a partir de um fluxo de vídeo (por exemplo, da webcam). A cada quadro, o modelo processa a imagem e devolve listas de coordenadas (normalmente normalizadas) dos pontos detectados. Essas coordenadas podem ser usadas para reconhecer gestos e posturas: por exemplo, identificando quando a mão de um usuário cruza determinada região da imagem ou se um usuário levantou um braço. Esse processamento em tempo real torna possível criar sinais digitais interativos. Usando FaceMesh, por exemplo, é possível inferir expressões faciais ou a direção do olhar de um espectador, enquanto HandPose facilita reconhecer gestos (como acenos ou pinças no ar).

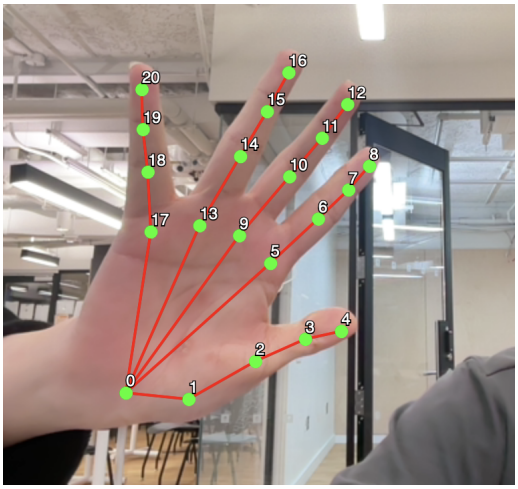


Figure 2.6. Mapeamento de pose de mão por meio de pontos feito pelo modelo Handpose [ml5.js contributors 2018].

A detecção de pose corporal permite determinar quando alguém está presente ou se movimentando em frente ao display, também é possível colocar uma condição para iniciar uma atividade, por exemplo, o usuário ter que fazer um sinal de 'Ok' para iniciar um jogo. Com essas informações, displays digitais podem adaptar o conteúdo exibido de acordo com o comportamento do público, ou criar aplicações em que o usuário ativamente interage com seus movimentos.

2.4.2. Matter.js

Uma vez estabelecido o mapeamento dos gestos do usuário, torna-se essencial disponibilizar elementos com os quais ele possa interagir. É nesse momento que se desenvolve a lógica funcional da aplicação, definindo quais interações desencadeiam eventos específicos e orientam o fluxo de uso. Para conferir maior dinamismo e fluidez à experiência interativa, é comum que os desenvolvedores incorporem elementos dotados de propriedades físicas, como colisão, velocidade e inércia. Nesse contexto, a biblioteca **Matter.js** [Brummit 2023] mostra-se uma ferramenta adequada, pois permite a criação de objetos e simulações que reproduzem, com precisão, comportamentos físicos diversos. Seu objetivo é permitir que desenvolvedores criem simulações interativas realistas, atribuindo a objetos gráficos propriedades físicas (massa, densidade, atrito etc.) e comportamentos dinâmicos.

Em particular, a Matter.js pode implementar detecção e resposta a colisões, gravidade uniforme e restrições mecânicas (como molas e juntas), recursos essenciais para adicionar realismo físico (queda, quicadas, empurrões) em jogos e aplicações de sinalização digital interativa.

A biblioteca funciona a partir de seus principais componentes: que incluem o *Engine*, o *World*, o *Composite* e as entidades *Body* e *Bodies*. O módulo *Matter.Engine* representa o motor de simulação física, responsável por atualizar o estado do mundo a cada quadro. Ao se criar um engine, é gerada uma instância de Engine que contém o atributo *engine.world*: a raíz do sistema, abrigando todos os corpos e restrições da simulação. Por sua vez, *Matter.Bodies* oferece fábricas para criação de formas geométricas comuns (círculos, retângulos, polígonos), enquanto *Matter.Body* designa cada objeto físico

individual criado através dessas fábricas. Além disso, o módulo *Matter.Composite* fornece métodos para agregar corpos em estruturas complexas, permitindo a composição de objetos complexos a partir de partes simples.

Uma vez criados, esses corpos podem ser manipulados dinamicamente. Por exemplo, o método *Matter.Body.applyForce* aplica uma força linear a um corpo a partir de um ponto no espaço, gerando aceleração e torque correspondentes. Em contrapartida, *Matter.Body.setVelocity* define diretamente a velocidade linear do corpo, e *Matter.Body.setPosition* atualiza instantaneamente a posição do objeto no mundo. O atributo body.torque acumula a força de rotação aplicada a cada atualização do motor (zerando a cada passo), indicando a tendência de giro do corpo em cada frame.

No navegador web, a integração do Matter.js é feita tipicamente via renderização em um canvas ou contexto similar. Para visualização, o módulo Matter.Render provê um renderizador básico em um elemento canvas, no entanto, a renderização desses objetos também pode ser feita por ferramentas externas, como será mostrado no próximo tópico.

Em cenários de produção, pode-se usar o laço de execução próprio *Matter.Runner* ou um loop *requestAnimationFrame* personalizado, chamando repetidamente Engine.update e, então, desenhando os corpos. Para entrada de usuário, Matter.js inclui o plugin *Matter.MouseConstraint*, que permite interagir com os corpos através de cliques ou toques na tela. Essa abordagem de restrição do ponteiro pode ser estendida para sinais gestuais, em que a posição rastreada da mão é mapeada para ações físicas na simulação.

2.4.3. P5.js

Como discutido na subseção anterior, para que os elementos simulados pelo Matter.js sejam visíveis ao usuário, é necessário renderizá-los em um elemento gráfico da tela, como um canvas. Embora a própria biblioteca forneça um renderizador básico por meio de seu módulo *Matter.Render*, esse recurso limita-se à exibição das formas geométricas simples presentes no "mundo" da simulação. Tal abordagem pode ser insuficiente para aplicações que demandam maior complexidade visual, como, por exemplo, a exibição de textos, imagens/fotos ou interfaces que respondam a toques e arrastos. Para atender a essas necessidades, o Matter.js permite integração com outros contextos gráficos mais versáteis, como a biblioteca **P5.js** [McCarthy 2025], que será apresentada nesta seção do minicurso.

A **P5.js** é uma biblioteca JavaScript - feita com base no *Processing.js* - voltada a artistas e iniciantes, que simplifica a programação criativa e a manipulação de elementos gráficos no navegador. Com p5.js, basta chamar *createCanvas()* dentro de uma função *setup()* para gerar o elemento canvas automaticamente e usar a função *draw()* para redesenhar a cena a cada frame, e também, é aqui que os elementos do matter também podem ser desenhados. A biblioteca conta com uma vasta quantidade de funções destinadas a desenho, desde desenho 2D até 3D, além de formas geométricas e também elementos DOM. Além disso, p5.js conta com uma comunidade ativa e diversas bibliotecas complementares (*p5.sound*, *p5.dom*, *p5.accessibility*), tutoriais e exemplos oficiais que aceleram o protótipo e a experimentação de interfaces interativas para desenvolvedores iniciantes.

2.5. Estudo de caso: Criação de uma aplicação interativa utilizando o conjunto MMP

Tendo sido apresentadas as capacidades e vantagens das bibliotecas que compõem o conjunto MMP, esta seção demonstrará, na prática, o uso dessas ferramentas por meio de um tutorial para o desenvolvimento de um projeto simples de jogo interativo. O objetivo é evidenciar o potencial dessas bibliotecas na criação de experiências digitais imersivas, mesmo em cenários introdutórios.

Neste tutorial, será desenvolvido um jogo interativo que visa a detecção de gestos com ml5.js, a renderização gráfica com p5.js e a simulação física com Matter.js. Ao final da aplicação, o jogador utilizará um gesto de pinça diante da webcam para "agarrar" palavras flutuantes, arrastá-las e posicioná-las corretamente sobre plataformas fixas, formando uma sequência coerente. O tutorial abordará desde a preparação do ambiente e a captura de vídeo, passando pela configuração do motor físico, até a criação, exibição e manipulação das entidades textuais, com lógica de colisão e validação automática da vitória quando todas as palavras estiverem corretamente organizadas. Com esta atividade, será possível compreender como integrar modelos de aprendizado de máquina para rastreamento de mãos, manipular corpos rígidos em um espaço bidimensional e renderizar conteúdos gráficos interativos em um canvas.

O jogo é uma demonstração de uma aplicação que seria executada em um totem interativo com câmera, instalado em um ambiente físico da empresa fictícia "DYB" (*Develop Your Dreams*). A dinâmica consiste em desafiar o jogador a organizar corretamente palavras, posicionando-as nos espaços designados (slots) na tela. Ao completar a tarefa com sucesso, a aplicação exibirá uma mensagem de vitória, informando que o jogador concluiu o desafio. Para reiniciar o jogo, será solicitado ao usuário que una todos os dedos diante da câmera, gesto que acionará a reinicialização da interface interativa.

2.5.1. Estrurura do Projeto Inicial da Página HTML

Comumente, os projetos que utilizam as bibliotecas do conjunto MMP utilizam o editor online do P5.js [McCarthy 2025], porém, haja vista que esta aplicação possui o intuito de ser publicada na web, utilizaremos a arquitetura mostrada na Figura 2.7, que é uma estrutura básica para uma página HTML.

O arquivo *index.html* deve ter uma estrutura da página web, com um campo reservado para o canvas em que será exibido o jogo. E também deverá ter as importações das bibliotecas por meio de CDN, assim como no Código 2.1.

```
<!DOCTYPE html>
<html lang="pt-BR">
<head>
  <!-- Links CDN para as bibliotecas -->
  <script src="https://cdnjs.cloudflare.com/ajax/libs/p5.js/1.11.1/p5.
    js"></script>
  <script src="https://cdnjs.cloudflare.com/ajax/libs/matter-js/0.20.0/
    matter.min.js"></script>
  <script src="https://unpkg.com/ml5@1/dist/ml5.min.js"></script>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0"
```

```
    >
</head>
<body>
  <main>
    <!-- local onde ficará o canvas da aplicação -->
    <div class="canvas-container"></div>
  </main>
  <script src="scripts/sketch.js"></script>
  <script src="scripts/boxes.js"></script>
  <script src="scripts/gameLogic.js"></script>
</body>
</html>
```

Código 2.1. Estrutura da página html

2.5.2. Setup do mapeamento de mão e do canvas do P5.js

Agora, para iniciar, faremos a configuração do modelo *handpose* do Ml5.js, que faz o mapeamento da mão do usuário. No arquivo *sketch.js* (Código 2.2), devem ser criadas as variáveis que serão utilizadas pelo ml5. Também são criadas as variáveis que estarão relacionadas às pontas dos dedos do usuário, pois cada ponto de mapeamento da mão recebe uma numeração própria, como pode ser observado na Figura 2.6. Após isso, dentro da função *preload* - específica do P5.js que é executada automaticamente - o modelo handpose é carregado com parâmetros que definem o número de mãos que serão mapeadas simultaneamente e com a captura flipada (por meio do atributo *Flipped*, que inverte a imagem horizontalmente), para que a imagem não fique invertida para o usuário. Após isso, na função *setup*, o canvas é criado e colocado na div reservada no HTML, e também é iniciada a captura de vídeo, que é inicialmente escondida pela função *video.hide()*, pois ela gera um elemento <video> no DOM, fora do canvas. Posteriormente, esse vídeo será exibido pelo próprio P5.js, na função *draw()*. A função *gotHands()* é uma callback que armazena results - que é um array de todos os pontos do mapeamento - na variável hands, para que possamos pegar as coordenadas dos pontos nos próximos passos.

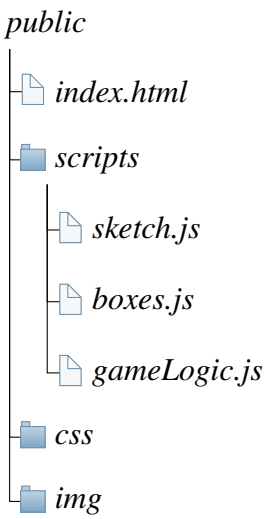


Figure 2.7. Estrutura de diretórios do projeto

```
let handPose, video, hands = [];

const INDEX_FINGER_TIP = 8; //Indice da ponta do indicador
const THUMB_TIP = 4; // Indice da ponta do polegar

function preload() {
  // Carrega o modelo handPose do Ml5.js
  handPose = ml5.handPose({ maxHands: 1, flipped: true });
}

function setup() {
  // Cria canvas e captura de vídeo (p5.js)
  let canvas = createCanvas(1280, 960);
  canvas.parent(document.querySelector(".canvas-container"));

  video = createCapture(VIDEO, { flipped: true });
  video.size(width, height);
  video.hide(); // Oculta o elemento <video> do DOM

  // Inicia detecção de mãos
  handPose.detectStart(video, gotHands);
}

function gotHands(results) {
  hands = results; // Atualiza keypoints das mãos
}
```

Código 2.2. Configuração do modelo Handpose e do Canvas.

2.5.3. Configurando o Motor de Física

Ainda no sketch, agora devemos realizar os preparos para a criação dos objetos do jogo (Código 2.3). Primeiro, são extraídos os componentes do Matter que serão utilizados neste projeto, eles são:

- **Engine** - Cria e controla o motor de física (tempo, atualização etc.)
- **Composite** - Está relacionado a agrupamento de corpos (compostos), neste projeto servirá principalmente adicionarmos elementos ao composto *Engine.world*, que abriga todos os elementos.
- **Bodies** - Fábrica de corpos: circle(), rectangle(), etc.
- **Body** - Manipula um corpo físico (mover, rotacionar, alterar massa...)

As aplicações desses componentes serão observadas no decorrer deste tutorial. Após isso, novamente na função *setup()*, o motor de simulação é criado por meio do *Engine*, e então, utilizando o *Bodies*, cria-se um chão para que os objetos não caiam para fora do canvas e dois círculos que servirão de ponteiros para os dedos; eles possuem a propriedade *isStatic*, pois não queremos que eles sejam afetados pela gravidade. Logo após isso, esses corpos são adicionados ao composto *Engine.world*, para que sejam incluídos no motor da simulação.

```
const { Engine, Bodies, Composite, Body } = Matter;

let engine, pointer, thumbPointer, ground;

function setup() {
  // ... (código de vídeo)

  // Cria o motor de física
  engine = Engine.create();

  // Cria um chão e pointeiros estaticos que serao presos nas pontas do
    dedo
  ground = Bodies.rectangle(width / 2, width * 3 / 4 - 50, width, 30, {
    isStatic: true });
  pointer     = Bodies.circle(0,0, 10, { isStatic: true });
  thumbPointer= Bodies.circle(0,0, 10, { isStatic: true });

  // Adiciona ao mundo
  Composite.add(engine.world, [ground, pointer, thumbPointer]);
}
```

Código 2.3. Configuração do motor de simulação Física do Matter e inclusão de elementos no mundo.

2.5.4. Criação das caixas de palavras e dos Slots do jogo

Agora, iremos os objetos principais para o jogo, as "caixas" que servirão de colisores para as palavras que irão formar as frases que o jogador deve formar, e os encaixes/plataformas em que ele deverá posicionar as caixas. Para isso, no arquivo *boxes.js* iremos montar um construtor para as caixas de colisão das palavras e das plataformas do jogo (Código 2.4).

O atributo *letters* recebe o array *gameWord* que iremos criar no *sketch.js*, ele contém a frase que deverá ser formada. O atributo *followingIndex* é um indicador de qual palavra o jogador está segurando no momento. E o atributo *slotsFilled* servirá para registrar quais encaixes foram preenchidos e o seu conteúdo; esses atributos servem para o controle da lógica do jogo, e em um próximo tópico, eles serão abordados.

Além disso, também é utilizado o componente *Bodies* para criarmos as colisões em formato de retângulo nas linhas 27 e 42 do Código 2.4.

```
class Boxes {
    constructor(num, size) {
        //quantidade e tamanho das caixas de colisão das palavras
        this.num = num;
        this.size = size;

        this.bodies = []; //array para as caixas das palavras do jogo
        this.platforms = [];//array para as caixas das plataformas das
    palavras

        this.letters = gameWord;  //palavras do jogo
        this.followingIndex = -1; //Variável que indica a caixa que est
    á seguindo o dedo indicador
        this.slotsFilled = new Array(num).fill(null);
        this.gameCompleted = false;

```

```
        this.initialize();
    }

    // Funçao que é chamada quando a classe Boxes é instanciada
    initialize() {
        // Criaçao das colisoes para cada palavra do jogo
        for (let i = 0; i < this.num; i++) {
            //posição de spawn de cada palavra e a largura(depende do tamanho da palavra)
            let wordWidth = textWidth(this.letters[i]) + 20;
            let x = random(100, 1100);
            let y = random(550, 750);

            let box = Bodies.rectangle(x, y, wordWidth * 2, this.size, {
                //propriedades físicas de cada caixa
                frictionAir: 0.1,
                restitution: 0.6,
                density: 0.002,
                angle: random(-0.1, 0.1)
            });

            this.bodies.push(box);
            Composite.add(engine.world, [box]); //adiciona as caixas para o mundo

            //Criação das plataformas para as palavras
            let platformX = width / 4 + i * 200;
            let platformY = 220;

            let platform = Bodies.rectangle(platformX, platformY + 10, this.size * 5, 17, { isStatic: true });
            this.platforms.push(platform);
            Composite.add(engine.world, [platform]); //adiciona as plataformas no mundo
        }
    }
}
```

Código 2.4. Construtor das caixas.

Com o construtor finalizado, vamos criar no *sketch.js* as palavras do jogo e instanciar a classe **Boxes** na função *Setup* (Código 2.5).

```
const gameWord = ["Develop", "Your", "Brain"];

function setup(){
    // ... restante do codigo ...
    boxes = new Boxes(gameWord.length, 40);
}
```

Código 2.5. Adição da gameWord e intanciação da classe Boxes.

2.5.5. Implementação das Regras do Jogo

Uma vez criados todos os objetos que iremos usar, agora iremos para a criação das mecânicas de jogabilidade. O jogo pode ser dividido em duas mecânicas principais:

- Pegar uma palavra - o jogador, fazendo um gesto de pinça (aproximar a ponta do dedo indicador com o polegar), poderá pegar uma das palavras ao aproximar os dedos da palavra. A palavra vai seguir os movimentos da mão do jogador se ele manter os dedos juntos. Se ele quiser soltar a caixa, precisa apenas separar os seus dedos.
- Colocar a palavra em um slot - uma vez que o jogador pegue uma palavra, ele deve movimentála até uma das plataformas, se ele posicionar na correta, a palavra irá parar de seguir a mão. Uma vez que todas os encaixes tenham sido preenchidos, o jogador ganhou o jogo.

Para implementar essas mecânicas, continuaremos incluindo funções na classe Boxes, então, para separarmos o código relacionado à construção das caixas e à lógica do jogo, iremos criar as funções no arquivo *gameLogic.js* (Código 2.6). A função *checkCollision()* é responsável por verificar a colisão entre os dedos e a caixa, e fazer com que ela os siga uma vez que tenha sido encostada. Enquanto isso, a função *checkPlatforms()* verifica a colisão entre a palavra e o encaixe, e armazena a palavra encaixada em *slotsFilled*. Uma vez que todas as palavras sejam encaixadas, a função *win* é chamada e executa um registro no console, indicando a vitória (numa aplicação real isso pode ser substituído por algum outro evento específico).

```
1  // Função que vai checar se o usuário está fazendo o gesto de pinça
       para pegar uma palavra
2  Boxes.prototype.checkCollision = function (pointer, thumb) {
3
4      let distance = dist(pointer.position.x, pointer.position.y, thumb.
       position.x, thumb.position.y);
5
6      //Primeiro verifica se tem alguma caixa seguindo o dedo
7      if (this.followingIndex === -1) {
8
9          // Verifica se o gesto de pinça foi feito e então checka a
       proximidade do dedo com a caixa
10         if (distance < 50) {
11             for (let i = 0; i < this.bodies.length; i++) {
12                 let box = this.bodies[i];
13                 let boxDistance = dist(box.position.x, box.position.y,
       pointer.position.x, pointer.position.y);
14
15                 if (boxDistance < this.size * 1.5 && !this.slotsFilled.
       includes(i)) {
16                     this.followingIndex = i;
17                     break;
18                 }
19             }
20         }
21     } else {
```

```
22          let box = this.bodies[this.followingIndex]; //pega a caixa que
    está seguindo o dedo
23
24          // Se os dedos se afastarem, parar de seguir
25          if (distance > 120) {  // Define um limite para soltar a caixa
    - ao afastar os dedos a caixa é solta
26              this.followingIndex = -1;
27          } else {
28              let newPosition = {
29                  x: lerp(box.position.x, pointer.position.x, 0.2), // o
    lerp é usado para dar suavidade no movimento de seguir da caixa, nã
    o obrigatório
30                  y: lerp(box.position.y, pointer.position.y, 0.2)
31              };
32              Body.setPosition(box, newPosition);
33          }
34      }
35  }
36  Boxes.prototype.checkPlatforms = function () {
37
38      for (let i = 0; i < this.bodies.length; i++) {
39          let box = this.bodies[i];
40          let platform = this.platforms[i];
41
42          //Traça uma distancia entre cada caixa e a sua respectiva
    plataforma
43          let distance = dist(box.position.x, box.position.y, platform.
    position.x, platform.position.y);
44
45          if (distance < this.size) {
46              //posiciona a caixa no mesmo lugar da plataforma ao
    encostar
47              Body.setPosition(box, { x: platform.position.x, y: platform
    .position.y - this.size / 2 });
48              Body.setStatic(box, true);
49
50              if (!this.slotsFilled[i]) { // Só armazena se ainda não
    estiver preenchido
51                  this.slotsFilled[i] = this.letters[i]; //armazena a
    palavra da caixa
52              }
53
54              if (this.followingIndex === i) {
55                  this.followingIndex = -1;
56              }
57          }
58      }
59
60      if (this.slotsFilled.every(slot => slot !== null)) {
61          this.win();
62      }
63
64      console.log(this.slotsFilled);
65  }
66
```

```
67 //função que define o que será feito quando o jogador ganha o jogo
68 Boxes.prototype.win = function () {
69     console.log("Você venceu!");
70     this.gameCompleted = true;
71 }
```

Código 2.6. Lógica das mecânicas do jogo.

2.5.6. Desenhando os elementos do jogo

Até este momento, todos os objetos e corpos que criamos até agora ainda não podem ser visualizados pelo usuário, pois nenhum deles foi renderizado pelo P5.js. Isto é feito por meio de funções de desenho que serão colocadas em uma função especial do P5 chamada *draw()*. Mas antes de voltarmos para o *sketch.js*, é preciso primeiro criar as funções de desenho das caixas no construtor da classe Boxes(Código 2.7. Fazemos isso criando uma função da classe Boxes chamada *display()* que depois será chamada no *draw()*. Nela, utilizamos as funções *rect()* - que criam retângulos - para fazermos os encaixes das palavras e *text()* para escrever as palavras nas caixas de colisão, que não serão visualizadas pelo usuário.

```
1  display() {
2
3      textAlign(CENTER, CENTER); //alinhar as palavras com as colisoes
4      textSize(24);
5
6      // desenho das plataformas
7      fill("#d3d3d3");
8      for (let i = 0; i < this.platforms.length; i++) {
9          let px = this.platforms[i].position.x;
10         let py = this.platforms[i].position.y - 20;
11         let wordWidth = textWidth(this.letters[i]) + 20;
12
13         rectMode(CENTER);
14         rect(px, py, wordWidth * 1.5, this.size * 1.3);
15     }
16
17     // "desenho" das palavras que serão posicionadas nas caixas
18     for (let i = 0; i < this.bodies.length; i++) {
19         let x1 = this.bodies[i].position.x;
20         let y1 = this.bodies[i].position.y;
21
22         textSize(40);
23         text(this.letters[i], x1, y1);
24     }
25 }
```

Código 2.7. Desenho das plataformas e das palavras.

Feito isso, só resta desenharmos o resto dos elementos no *sketch.js* por meio da função *draw()* 2.8. A função draw é executada repetidas vezes por segundo, por isso, além das funções de desenho, também está nela o *Engine.Update()*, para que o motor de simulação seja atualizado constantemente, assim permitindo a fluidez dos movimentos dos corpos. As funções de desenho presentes no draw incluem:

- **image()** - Desenha uma imagem no canvas, como ela está na função *draw()* e está recebendo "video" como parâmetro, essa função está servindo como um reprodutor da imagem capturada pela câmera.

- **ellipse()** - Desenha formas elípticas. No jogo esses círculos são usados para indicar as pontas dos dedos do usuário

- **rect() e text()** - presentes dentro da função *boxes.display()*, como citado anteriormente, desenham retângulos e textos, respectivamente.

E por fim, a função *CheckCollision()* está sendo chamada com os círculos das pontas dos dedos como parâmetro, para que a posição seja verificada, e também a função *checkPlatform()* é chamada para constantemente verificar o estado do jogo.

```
function draw() {

  background(220);

  Engine.update(engine);
  // Exibe a imagem da webcam ajustada ao tamanho do canvas
  image(video, 0, 0, width, height);


  // Desenha Círculos nas pontas dos dedos
  if (hands.length > 0) {
    index = hands[0].keypoints[INDEX_FINGER_TIP];
    thumb = hands[0].keypoints[THUMB_TIP];

    pointer.position.x = index.x;
    pointer.position.y = index.y;

    thumbPointer.position.x = thumb.x;
    thumbPointer.position.y = thumb.y;

    fill("#49E60F");
    ellipse(index.x, index.y, 10);
    ellipse(thumb.x, thumb.y, 10);
  }

  boxes.checkCollision(pointer, thumbPointer);
  boxes.checkPlatforms();
  boxes.display();

}
```

Código 2.8. Desenho das plataformas e das palavras.

Com todos esses passos concluídos, a aplicação imersiva está completa, possuindo funções de machine learning e de simulações físicas, sendo configurada apenas com lógica básica de JavaScript.

2.6. Conclusão

Ao longo deste capítulo, discutimos os fundamentos da interatividade por gestos em sinalização digital partindo do surgimento dessa tecnologia até como ela é aplicada atualmente. Em seguida, exploramos a importância do aprendizado de máquina para o treinamento e a inferência de modelos capazes de reconhecer gestos, e como o WebGL viabiliza essa computação no navegador. Na sequência, apresentamos o conjunto MMP — ml5.js, Matter.js e P5.js — detalhando, em cada subseção, como ml5.js fornece uma interface amigável para acesso a modelos de visão, como Matter.js adiciona realismo físico às interações e como P5.js enriquece a renderização gráfica. Por fim, demonstramos tudo isso na prática por meio de um tutorial que integra detecção de gestos, simulação 2D e desenho em canvas para criar uma aplicação interativa.

Espera-se que, com essa trajetória — do conceito à aplicação prática —, o leitor tenha uma visão clara não apenas das teorias envolvidas, mas também do potencial dessas ferramentas para se criar verdadeiras experiências digitais imersivas.

References

[Asif et al. 2024] Asif, R., Naveed, A., Farasat, M., and Khan, M. I. (2024). Impact of digital signage and social media advertising on consumer buying behavior: Mediating role of emotional processes. *Journal of Asian Development Studies*, 13(1):1147–1160.

[BrightSign LLC 2025] BrightSign LLC (2025). Brightsign®: Soluções de mídia para sinalização digital. Empresa líder global em players de mídia para sinalização digital, fundada em 2002 e sediada em Los Gatos, Califórnia.

[Brignull and Rogers 2003] Brignull, H. and Rogers, Y. (2003). Enticing people to interact with large public displays in public spaces. In *Interact*, volume 3, pages 17–24.

[Brummit 2023] Brummit, L. (2023). Matter.js is a 2d physics engine for the web. Disponível em: `https://brm.io/matter-js/`. Acesso em: 14 de março de 2025.

[Chen et al. 2009] Chen, Q., Malric, F., Zhang, Y., Abid, M., Cordeiro, A., Petriu, E. M., and Georganas, N. D. (2009). Interacting with digital signage using hand gestures. In Kamel, M. and Campilho, A., editors, *Image Analysis and Recognition*, pages 347–358, Berlin, Heidelberg. Springer Berlin Heidelberg.

[Chung 2017] Chung, S. Z. (2017). Smart digital signage with eye tracking system. Masters thesis, Universiti Sains Malaysia.

[Coldewey 2019] Coldewey, D. (2019). This hand-tracking algorithm could lead to sign language recognition. `https://techcrunch.com/2019/08/19/this-hand-tracking-algorithm-could-lead-to-sign-language-recognition/`. Acessado em 9 de maio de 2025.

[Davies et al. 2022] Davies, N., Clinch, S., and Alt, F. (2022). *Pervasive displays: understanding the future of digital signage*. Springer Nature.

[de Oliveira and de Melo 2022] de Oliveira, B. V. N. and de Melo, F. T. (2022). Fundamentos da visão computacional: Arcabouço teórico do reconhecimento artificial de imagens e vídeos. *Humanidades & Inovação*, 10(17):313–324.

[Digital 2025] Digital, T. (2025). Google lança recurso de publicidade para digital out-of-home. Disponível em: https://www.teoriadigital.com.br/google-publicidade-digital-dooh/. Acesso em: 22 de abril de 2025.

[HIIT 2025] HIIT (2025). Citywall. Disponível em: https://citywall.org/. Acesso em: 25 de abril de 2025. Helsinki Institute for Information Technology.

[IBM Brasil 2023] IBM Brasil (2023). O que são redes neurais convolucionais? https://www.ibm.com/br-pt/think/topics/convolutional-neural-networks. Acessado em maio de 2025.

[McCarthy 2025] McCarthy, L. L. (2025). P5.js - friendly tool for learning to code and make art. Disponível em: https://p5js.org/. Acesso em: 14 de março de 2025.

[McDonald's Corporation 2020] McDonald's Corporation (2020). Self-Ordering Kiosks Improve McDonald's Customer Experience. https://corporate.mcdonalds.com/corpmcd/scale-for-good/our-planet/kiosk-experience.html. Acesso em: 28 abr. 2025.

[Microsoft 2010] Microsoft (2010). Kinect for Windows Sensor. https://developer.microsoft.com/en-us/windows/kinect/. Acesso em: 28 abr. 2025.

[Microsoft 2019] Microsoft (2019). Onnx.js: Run onnx models in browser using javascript. https://github.com/microsoft/onnxjs. Acessado em: 2025-05-09.

[ml5.js contributors 2018] ml5.js contributors (2018). ml5.js: Friendly machine learning for the web. https://ml5js.org. Acessado em: 2025-05-09.

[Müller et al. 2012] Müller, J., Walter, R., Bailly, G., Nischt, M., and Alt, F. (2012). Looking glass: a field study on noticing interactivity of a shop window. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '12, page 297–306, New York, NY, USA. Association for Computing Machinery.

[Parisi 2012] Parisi, T. (2012). *WebGL: up and running*. " O'Reilly Media, Inc.".

[Peltonen et al. 2008] Peltonen, P., Kurvinen, E., Salovaara, A., Jacucci, G., Ilmonen, T., Evans, J., Oulasvirta, A., and Saarikko, P. (2008). It's mine, don't touch! interactions at a large multi-touch display in a city centre. In *Proceedings of the SIGCHI conference on human factors in computing systems*, pages 1285–1294.

[SEAT S.A. 2021] SEAT S.A. (2021). SEAT Creates Digital Showroom Experience. https://www.seat.com/company/news/corporate/digital-showroom-technology.html. Acesso em: 28 abr. 2025.

[Shiffman et al. 2018] Shiffman, D. et al. (2018). ml5.js: Friendly open source machine learning library for the web. https://itp.nyu.edu/adjacent/issue-3/ml5-friendly-open-source-machine-learning-library-for-the-web/. Acessado em: 12 de maio de 2025.

[TensorFlow.js Team 2023] TensorFlow.js Team (2023). Platform and environment. https://www.tensorflow.org/js/guide/platform_environment. Acessado em maio de 2025.

[TV 2023] TV, O. (2023). Using Proximity Sensors with Digital Signage. https://onsign.tv/blog/technology/motion-sensors-with-digital-signage/11. Acesso em: 28 abr. 2025.

[Walter et al. 2013] Walter, R., Bailly, G., and Müller, J. (2013). Strikeapose: revealing mid-air gestures on public displays. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 841–850.

Chapter 3

Como Criar seu Próprio Assistente de Pesquisa Científica com LangGraph

Larissa Souza do Nascimento e Ricardo Moura Sekeff Budaruiche

Abstract

This chapter discusses the development of a multi-agent system based on the LangGraph framework to automate scientific searches. The proposed architecture combines specialized agents, including arXiv querying, general web search via Tavily AI, and semantic storage using ChromaDB, coordinated by a supervising agent capable of delegating tasks and interpreting responses. Throughout development, the project faced limitations related to the reliability of AI services, the control of hallucinations, dependency on private API keys, and difficulties in explaining agent behavior. Despite these challenges, the results indicate promising potential for enhancing literature reviews and promoting faster scientific information retrieval. However, further work is needed to improve source filtering, integrate with institutional repositories, and define robust evaluation metrics. The initiative aims to spark new discussions on the responsible and effective deployment of intelligent agents in scientific workflows.

Resumo

Este capítulo apresenta o desenvolvimento de um sistema multiagente baseado no framework LangGraph, voltado à automatização de buscas científicas. A solução propõe a coordenação de agentes especializados, como consultas ao arXiv, buscas gerais na web via Tavily AI e armazenamento semântico com ChromaDB, supervisionados por um agente central que decide e direciona as ações. Durante o desenvolvimento, foram enfrentadas limitações quanto à estabilidade das ferramentas de IA utilizadas, controle de alucinações, necessidade de chaves privadas, além de desafios relacionados à explicabilidade do comportamento dos agentes. Embora os resultados iniciais demonstrem o potencial da abordagem para otimizar revisões bibliográficas e acelerar o acesso à informação científica, ainda são necessárias melhorias quanto à curadoria de fontes, integração com bancos de dados institucionais e definição de métricas confiáveis de desempenho. Espera-se que a prática apresentada inspire novas aplicações com maior robustez, confiabilidade e alinhamento ético no uso de agentes inteligentes na ciência.

3.1. Introdução

O advento da Inteligência Artificial (IA) tem provocado transformações profundas na maneira como o conhecimento científico é produzido, acessado e validado. Nos últimos anos, o volume de dados científicos disponíveis cresceu exponencialmente, exigindo novas abordagens computacionais para que pesquisadores possam lidar com a complexidade e a velocidade de geração dessas informações (FIORILLO; MEHTA, 2024). Nesse contexto, destaca-se o uso de agentes inteligentes, capazes de automatizar tarefas repetitivas, filtrar informações relevantes e auxiliar no processo decisório.

Agentes inteligentes são sistemas computacionais que percebem o ambiente por meio de sensores e reagem por meio de atuadores, conforme ilustrado na Figura 3.1. Sua aplicação no contexto da pesquisa científica é particularmente relevante, pois permite a execução de tarefas como a recuperação automatizada de literatura, a integração de dados provenientes de diferentes fontes e a análise preditiva baseada em grandes volumes de informação (MANZOOR et al., 2012). Tais capacidades ampliam o potencial analítico dos pesquisadores, ao mesmo tempo que reduzem o tempo despendido com atividades operacionais.

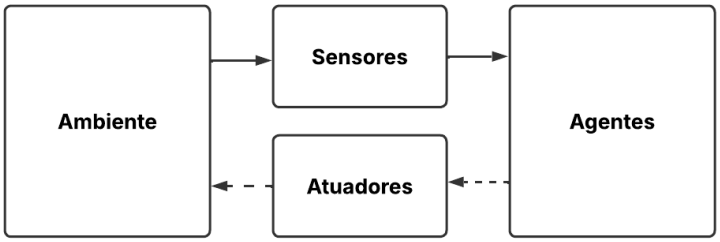


Figure 3.1: Fluxograma de agentes inteligentes

Diversos *frameworks* têm sido desenvolvidos para facilitar a construção desses agentes, destacando-se, entre eles, o *AutoGen* e o *CrewAI*. Essas ferramentas viabilizam a criação de agentes baseados em *Large Language Models* (LLMs) para tarefas específicas. Embora apresentem contribuições valiosas, frequentemente carecem de controle estrutural refinado e de flexibilidade nos fluxos de decisão, fatores cruciais quando se busca precisão e rastreabilidade em contextos científicos.

Diante dessa lacuna, o *LangGraph* surge como uma proposta inovadora. Desenvolvido pela *LangChain* (CHASE, 2024), o *LangGraph* adota uma arquitetura baseada em grafos direcionados, oferecendo suporte a ciclos de *feedback*, persistência de estado e controlabilidade em tempo real. Essas propriedades tornam o *framework* especialmente adequado para aplicações científicas, nas quais o refinamento iterativo das informações e a validação cruzada entre múltiplas fontes constituem práticas essenciais.

Este capítulo tem como objetivo apresentar uma abordagem prática e fundamentada para a construção de um assistente de pesquisa científica utilizando o *LangGraph*.

Por meio de uma combinação de fundamentação teórica, exemplos práticos e reflexões críticas, serão discutidos os seguintes aspectos:

1. A definição e a estrutura dos agentes inteligentes aplicados à ciência;
2. O funcionamento da arquitetura do *LangGraph* e suas vantagens comparativas;
3. O ambiente de desenvolvimento e a implementação prática de agentes com foco em pesquisa;
4. Estratégias de otimização e mitigação de falhas, como alucinações;
5. Aspectos éticos e responsáveis no uso da IA para fins científicos.

Ao longo do capítulo, será demonstrado como o *LangGraph* pode ser utilizado para criar agentes capazes de interagir com repositórios científicos, como o *arXiv*, integrando dados, refinando respostas e auxiliando na geração de conhecimento. Ressalta-se que o objetivo não é substituir o pesquisador humano, mas potencializar sua capacidade analítica, permitindo que ele se concentre em tarefas de maior valor intelectual.

3.2. Fundamentos Teóricos

A ascensão da IA tem transformado significativamente os métodos de investigação científica, promovendo avanços substanciais na automação, extração e interpretação de dados em larga escala. Dentre as soluções mais promissoras, destacam-se os agentes inteligentes baseados LLMs, cuja arquitetura permite a construção de assistentes autônomos voltados ao apoio direto à atividade de pesquisa. Este capítulo apresenta os conceitos teóricos que fundamentam o desenvolvimento desses agentes, com ênfase nas capacidades oferecidas pelo *framework LangGraph*, em comparação a alternativas contemporâneas como o *AutoGen* e o *CrewAI*.

3.2.1. Agentes Inteligentes: Conceito e Aplicações

Agentes inteligentes são definidos como entidades computacionais autônomas, capazes de perceber o ambiente, processar informações, tomar decisões e executar ações com o objetivo de atingir metas previamente estabelecidas. No contexto científico, esses agentes são utilizados para automatizar tarefas operacionais, como a busca em repositórios acadêmicos, a análise de documentos e a sumarização de dados, permitindo que os pesquisadores concentrem seus esforços na interpretação crítica e na formulação de hipóteses.

O *LangGraph* pode ser aplicado em uma ampla gama de cenários, como: desenvolvimento de *chatbots*, agentes autônomos, sistemas multiagentes, ferramentas de automação de fluxos de trabalho, ambientes personalizados de aprendizagem e sistemas de recomendação baseados em dados contextuais.

Além da autonomia, os agentes inteligentes distinguem-se por características como adaptabilidade, capacidade de interação com múltiplas fontes e habilidades de aprendizado contínuo. Quando organizados em sistemas multiagentes, esses agentes colaboram entre si, formando redes articuladas que permitem a resolução de problemas mais complexos de maneira cooperativa.

3.2.2. Estrutura de Agentes Baseados em LLMs

Com o avanço dos LLMs, emergiu uma nova geração de agentes capazes de realizar inferências complexas, interpretar linguagem natural com elevado grau de precisão e interagir de forma contextualizada com dados estruturados e não estruturados. *Frameworks* como o *LangGraph*, o *AutoGen* e o *CrewAI* têm popularizado a orquestração desses agentes, ao oferecerem interfaces modulares para integração com *Application Programming Interface* (APIs), bancos de dados e ferramentas externas.

A arquitetura geral desses agentes compreende os seguintes componentes:

- **Interface de entrada**: responsável por receber a demanda do usuário (pergunta ou tarefa);
- **Módulo de processamento**: operado por LLMs, encarregado de interpretar, executar e formular a resposta à solicitação;
- **Camada de integração**: responsável pela conexão com fontes externas de informação, como o repositório *arXiv*;
- **Sistema de retorno**: encarregado de organizar e apresentar a resposta final ao usuário de forma clara e estruturada.

Essa arquitetura viabiliza a automação de tarefas complexas e introduz uma nova abordagem à exploração científica mediada por IA, promovendo eficiência, escalabilidade e aprofundamento analítico no processo de produção do conhecimento.

3.2.3. Arquitetura do *LangGraph*

O *LangGraph* é um *framework* projetado para a construção de fluxos de agentes inteligentes utilizando grafos direcionados, oferecendo maior controle, modularidade e transparência na tomada de decisões executadas por esses agentes. Três propriedades fundamentais distinguem sua arquitetura:

- **Ciclos (feedback loops)**: permitem a retroalimentação das saídas para agentes anteriores, promovendo o refinamento contínuo das respostas por meio de iterações sucessivas;
- **Controlabilidade**: refere-se à capacidade de manipular dinamicamente os nós do grafo de execução, viabilizando decisões condicionais e ajustes em tempo de execução;
- **Persistência**: possibilita a manutenção do estado entre interações, preservando o raciocínio e o histórico de execução ao longo de uma sessão.

A Figura 3.2 ilustra o funcionamento conceitual do *LangGraph*, evidenciando o fluxo de dados entre os nós, as decisões condicionais que modulam o percurso de execução e a presença de ciclos que viabilizam iterações sucessivas. Este modelo torna o

LangGraph especialmente apropriado para aplicações científicas baseadas em agentes, nas quais a robustez do processo decisório depende da capacidade de revisão, persistência e controle granular sobre cada etapa do fluxo.

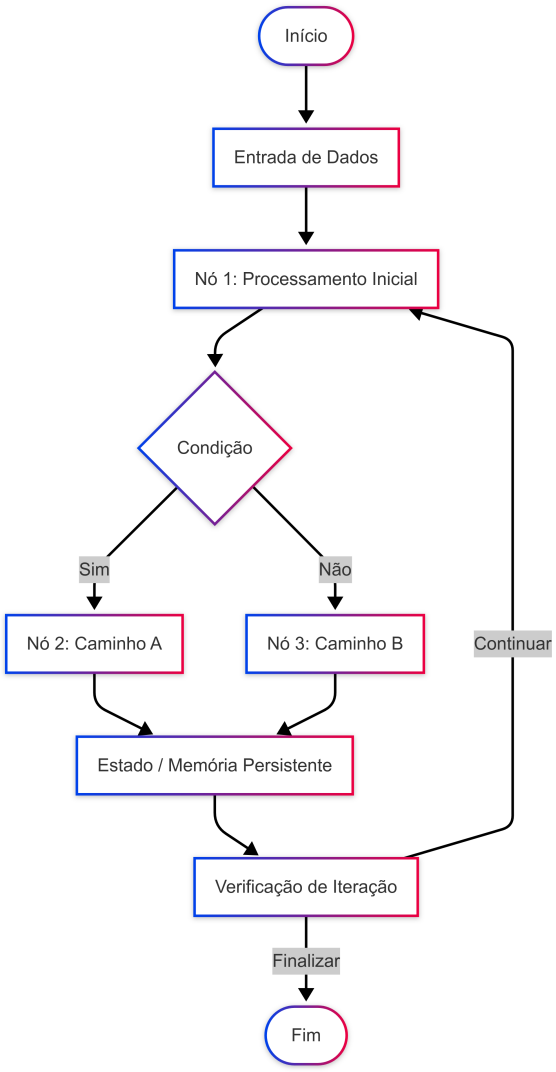


Figure 3.2: Funcionamento do LangGraph

Essas características tornam o *LangGraph* especialmente adequado para aplicações científicas, nas quais o refinamento progressivo, a validação iterativa e a integração com múltiplas fontes de informação são requisitos essenciais para garantir a robustez e a confiabilidade dos resultados.

3.2.4. Vantagens

O *LangGraph* permite a construção de aplicações com múltiplos atores e com gerenciamento de estado, utilizando LLMs de forma acessível e estruturada. O *framework* estende as funcionalidades do *LangChain* ao introduzir a capacidade de criar e gerenciar grafos cíclicos, o que se revela fundamental para a implementação de tempos de execução sofisticados em agentes inteligentes.

Seus principais conceitos incluem: (i) estrutura de grafo, (ii) gerenciamento de estado e (iii) mecanismos de coordenação. A partir desses fundamentos, o *LangGraph* oferece vantagens significativas para desenvolvedores que trabalham com aplicações complexas baseadas em LLMs, entre as quais destacam-se:

- **Desenvolvimento simplificado**: abstrações de alto nível facilitam a implementação de fluxos complexos;
- **Flexibilidade**: permite modificar os caminhos de execução em tempo real, adaptando-se a diferentes contextos e necessidades;
- **Escalabilidade**: suporta a construção de agentes com múltiplas etapas, integrados a diversas fontes de dados;
- **Tolerância a falhas**: possibilita retentativas e controle de estados intermediários, promovendo resiliência no processo computacional.

3.2.5. Comparação com *Frameworks* Semelhantes

Frameworks como ***LangChain, AutoGen*** e ***CrewAI*** oferecem abordagens eficazes para a construção de agentes baseados em LLMs. No entanto, diferem significativamente do *LangGraph* no que se refere ao grau de controle sobre o fluxo de execução. Enquanto os primeiros operam majoritariamente com sequências lineares ou estruturas do tipo árvore de decisão, o *LangGraph* permite a definição de grafos direcionados, com possibilidade de atualização dinâmica dos caminhos de execução, o que favorece a inclusão de ciclos de validação, refinamento iterativo e lógica condicional.

Um estudo recente (BARBARROXA; GOMES; VALE, 2024) analisou o desempenho de *frameworks* como *AutoGen*, *CrewAI* e TaskWeaver em tarefas de geração automatizada de código, evidenciando suas potencialidades e limitações. Embora o *LangGraph* não tenha sido incluído nessa análise, sua arquitetura modular representa uma evolução relevante em termos de flexibilidade, rastreabilidade e capacidade de adaptação em contextos científicos exigentes.

3.2.6. Aplicações Científicas e Relevância Prática

O uso de agentes baseados em IA tem se consolidado em diversas áreas da investigação científica, destacando-se em tarefas como a revisão automatizada de manuscritos, extração de conhecimento a partir de bases científicas (MELONI et al., 2023) e apoio a decisões clínicas e editoriais. A capacidade desses agentes de interagir com múltiplas fontes e processar grandes volumes de dados em linguagem natural os torna cada vez mais essenciais em um cenário caracterizado pela crescente complexidade e quantidade de informação disponível.

De acordo com a pesquisa de (LUO et al., 2019), arquiteturas baseadas em grafos promovem maior precisão na coordenação entre múltiplos agentes, o que acelera a obtenção de resultados relevantes. Nesse contexto, o *LangGraph* destaca-se como uma solução promissora para o desenvolvimento de sistemas inteligentes de apoio à pesquisa científica, permitindo que pesquisadores configurem agentes especializados, ajustáveis às particularidades e requisitos de seus projetos.

3.3. Arquitetura do Sistema

Um agente inteligente é um sistema que utiliza LLM para decidir o fluxo de controle de uma aplicação. À medida que esses sistemas evoluem, eles tendem a se tornar mais complexos, o que dificulta sua manutenção, escalabilidade e capacidade de resposta. Em determinados contextos, como na implementação de um agente voltado à busca bibliográfica e análise científica, a utilização de apenas um agente pode não ser suficiente.

Entre os desafios enfrentados em sistemas com agente único, destacam-se: a presença de múltiplas ferramentas que confundem a tomada de decisão sobre qual utilizar em cada momento; a complexidade crescente do contexto, que compromete a efetividade de um único raciocinador; e a necessidade de competências especializadas distintas (por exemplo, planejamento, recuperação de documentos, interpretação estatística, entre outras). Para lidar com essas limitações, uma abordagem eficiente é a decomposição da aplicação em múltiplos agentes menores, com funções bem definidas, compondo assim um sistema multiagente (DUAN; WANG, 2024).

Os principais benefícios da adoção de arquiteturas multiagentes incluem:

- **Modularidade**: a separação em agentes independentes facilita o desenvolvimento incremental, além de simplificar o teste e a manutenção dos componentes do sistema.
- **Especialização**: permite a criação de agentes especialistas focados em domínios ou funções específicas, o que contribui para a eficiência e precisão do sistema como um todo.
- **Controle explícito**: possibilita a orquestração detalhada da comunicação entre os agentes, reduzindo a dependência de mecanismos implícitos de chamada de funções e promovendo maior previsibilidade no comportamento do sistema.

Dessa forma, sistemas baseados em múltiplos agentes inteligentes representam uma evolução arquitetural relevante, especialmente em contextos acadêmicos e científicos, nos quais a complexidade das tarefas exige abordagens mais escaláveis, colaborativas e ajustáveis a fluxos de trabalho diversos.

3.3.1. Arquitetura Sistema Multiagente

O *LangGraph* oferece diferentes arquiteturas para a construção de sistemas multiagentes, permitindo que os desenvolvedores escolham a forma mais adequada de orquestrar a comunicação entre agentes. Entre as principais abordagens, destacam-se:

- ***Network***: todos os agentes podem se comunicar entre si. Qualquer agente pode decidir qual outro agente acionar em seguida.
- ***Supervisor***: todos os agentes se comunicam com um agente supervisor central, responsável por decidir qual agente será ativado em cada etapa.
- ***Hierárquico***: extensão da arquitetura de supervisor, onde há supervisores de supervisores, permitindo fluxos de controle mais complexos.

Neste trabalho, foi adotada uma variante da arquitetura do tipo ***Supervisor***, denominada *Supervisor com chamada de ferramentas (tool-calling)*. Nesse modelo, os agentes especializados são representados como ferramentas que ficam à disposição de um agente supervisor. Este utiliza um LLM com capacidade de raciocínio baseado em chamadas de ferramentas para decidir qual agente acionar, bem como quais argumentos serão repassados.

O agente supervisor segue uma lógica de execução baseada em laço (*while-loop*), realizando chamadas sucessivas até que uma condição de parada seja alcançada. A Figura 3.3 ilustra essa arquitetura, na qual o supervisor recebe uma entrada do usuário e, com base no conteúdo da solicitação, redireciona a tarefa ao agente mais apropriado.

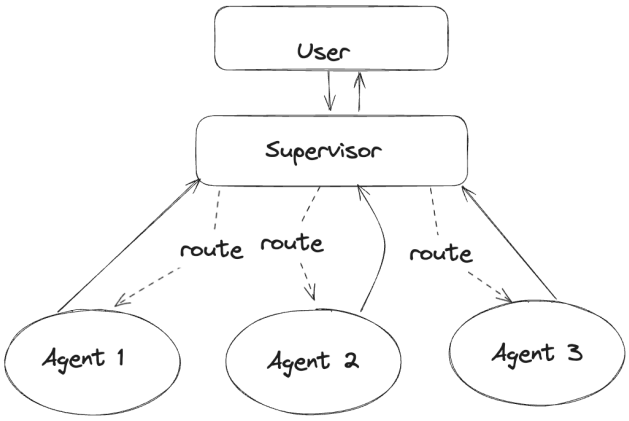


Figure 3.3: Arquitetura Agente Supervisor. Fonte: `https://github.com/langchain-ai/langgraph/blob/main/docs/docs/tutorials/multi_agent/agent_supervisor.ipynb`

No sistema desenvolvido, o agente supervisor coordena a atuação de três agentes especializados, descritos a seguir:

1. ***Tavily Agent***: realiza buscas na *web* por meio da API *Tavily*, adequado para consultas gerais e abertas;
2. ***arXiv Agent***: realiza buscas específicas em artigos científicos no repositório *arXiv*, além de realizar a ingestão dos dados no *ChromaDB*;
3. ***Scheduler Agent***: executa buscas científicas agendadas no *arXiv*, com controle de periodicidade, duração e número de resultados.

O fluxo de funcionamento é representado na Figura 3.4. Quando o usuário envia uma solicitação, o agente supervisor avalia seu conteúdo:

- Se a solicitação indicar a necessidade de uma pesquisa científica, o supervisor encaminha a demanda ao ***arXiv Agent***;

- Se a pergunta for mais ampla, como por exemplo - “qual a previsão do tempo?” -, o supervisor encaminha ao ***Tavily Agent***;
- Se o usuário desejar agendar uma pesquisa periódica — por exemplo, “pesquise artigos sobre transformers na IA durante 3 meses e me envie 10 artigos toda terça-feira” —, a tarefa é repassada ao ***Scheduler Agent***.

Caso a entrada do usuário não se enquadre em nenhum dos cenários esperados (por exemplo: “Oi, tudo bem?”), o supervisor simplesmente não aciona nenhum agente ou ferramenta, apenas responde de acordo a base de conhecimento pré-treinado do LLM.

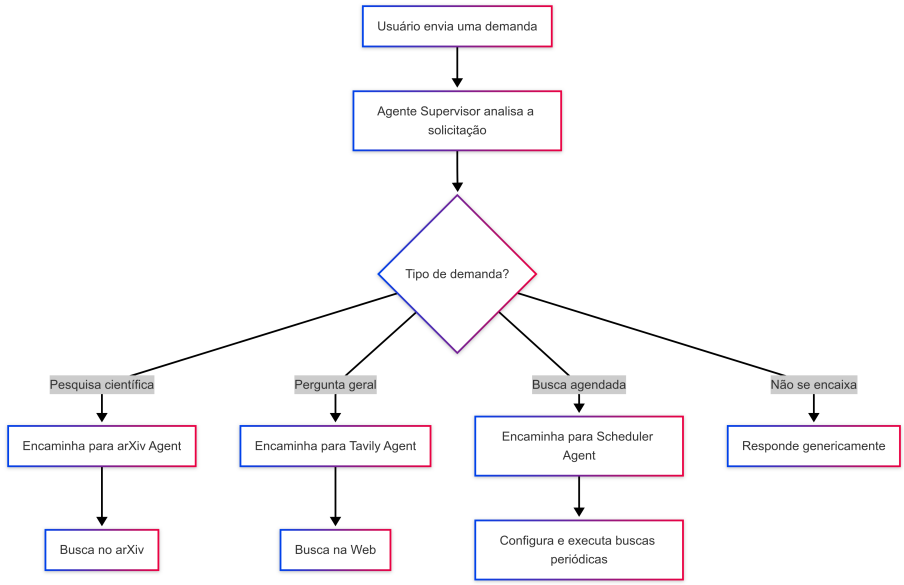


Figure 3.4: Fluxograma do funcionamento do sistema multiagente

Na Figura 3.5, é possível observar uma tela do sistema em operação, evidenciando a interação entre o usuário, o agente supervisor e os subagentes responsáveis por processar a solicitação.

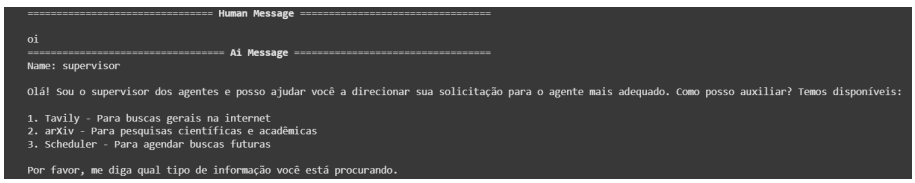


Figure 3.5: Funcionamento do agente

3.3.2. Tecnologias Utilizadas

Para o desenvolvimento do sistema multiagente voltado à realização de pesquisas científicas e buscas na *web*, foram empregadas tecnologias modernas e adequadas ao contexto de aplicações baseadas em modelos de linguagem. A seguir, apresentam-se as principais ferramentas utilizadas:

- ***Python***: utilizada como linguagem de programação principal devido à sua ampla adoção na comunidade de inteligência artificial, simplicidade sintática, vasta gama de bibliotecas voltadas ao desenvolvimento de agentes e suporte nativo a paradigmas como orientação a objetos e programação funcional. Além disso, o *Python* apresenta excelente integração com *frameworks* de IA, como *TensorFlow*, *PyTorch*, *LangChain* e *LangGraph*, facilitando tanto a prototipagem quanto o escalonamento de soluções complexas (KHANDARE et al., 2023).

- ***LangGraph***: *framework* de orquestração baseado em grafos direcionados, utilizado para a construção do sistema multiagente. Permite a criação de fluxos dinâmicos e cíclicos entre agentes, com controle refinado sobre os estados e transições, sendo ideal para aplicações que requerem tomada de decisão condicionada e persistência de estado ao longo de múltiplas interações.

- ***TavilySearch***: ferramenta de busca na *web* integrada via API, utilizada para fornecer respostas baseadas em fontes abertas e atualizadas. Sua principal função no sistema é atender às demandas informacionais gerais que extrapolam o escopo de bases científicas especializadas.

- ***ChromaDB***: banco de dados vetorial responsável pelo armazenamento de *embeddings* extraídos de documentos e artigos científicos. Distingue-se por sua eficiência, interface intuitiva, modelo de código aberto e desempenho otimizado para conjuntos de dados de pequena e média escala. Essas características tornam o *ChromaDB* particularmente apropriado para aplicações acadêmicas e sistemas multiagentes em fase de prototipagem. Além disso, apresenta elevada performance em tarefas de recuperação semântica, como demonstrado em estudos recentes (MATHUR; CHHABRA, 2024).

- **arXiv API**: fonte de dados especializada, utilizada para acesso estruturado a artigos científicos em formato XML. Permite consultas precisas por metadados (autores, palavras-chave, datas e resumos), sendo essencial para a operação do agente responsável por pesquisas bibliográficas.

- ***Anthropic Claude 3.5***: LLM de última geração utilizado para geração e interpretação de respostas dentro do sistema. Sua alta capacidade de raciocínio, compreensão de contexto extenso e desempenho superior em tarefas de múltiplos turnos o tornam adequado para funções de supervisão e coordenação de subagentes.

3.4. Ambiente de Desenvolvimento

Para que os leitores tenham uma experiência prática fluida, é fundamental estabelecer um *ambiente de desenvolvimento* padronizado. Nesta seção, são apresentados os requisitos de *software* e *hardware*, o passo a passo de instalação e configuração, bem como um exemplo inicial de uso do *LangGraph*. É necessário dispor de um editor de código, neste exemplo será utilizado o ***Visual Studio Code***, embora o leitor possa optar por outro de sua preferência e ter o **Python 3** previamente instalado na máquina.

3.4.1. Preparação e Configuração Inicial

1. **Clonar ou baixar o repositório:**

```
git clone https://github.com/larissaNa/LangGraphAgent.git
```

2. **Acessar a pasta do projeto:** No terminal, navegue até o diretório onde o repositório foi clonado:

```
cd <caminho-para-sua-pasta>
```

3. **Criar e ativar ambiente virtual Python**

```
python -m venv venv
source venv/bin/activate
venv scripts activate
```

4. **Instalar dependências:**

```
pip install -r requirements.txt
```

Para que o sistema multiagente funcione corretamente, é necessário obter e configurar as chaves de acesso (API keys) das ferramentas utilizadas.

- ***Anthropic API*** (ANTHROPIC, 2024): A *Anthropic* é uma empresa norte-americana especializada em IA, focada no desenvolvimento de LLMs responsáveis e confiáveis. Seu modelo *Claude-3.5* é utilizado neste sistema como supervisor do fluxo de agentes. A chave de API pode ser obtida no site oficial da empresa: `https://www.anthropic.com/`.

- ***Tavily API*** (TAVILY, 2024): O *Tavily* é uma ferramenta de busca baseada em IA voltada à realização de pesquisas rápidas e eficazes em diversas fontes da *web*. Permite a entrega de informações sintetizadas com alto grau de relevância. Sua API pode ser acessada via: `https://tavily.com/`.

3.4.2. Configuração do Arquivo `.env`

Após obter suas chaves, crie um arquivo `.env` na raiz do projeto e insira as informações da seguinte forma:

```
TAVILY_API_KEY="sua_chave"
ANTHROPIC_API_KEY="sua_chave"
```

Esse arquivo será utilizado pelo sistema para autenticar as chamadas às APIs externas de forma segura.

3.4.3. Execução do Sistema Multiagente

Com o ambiente devidamente configurado e as chaves de API fornecidas, o sistema pode ser executado com o seguinte comando no terminal:

```
python main.py
```

A partir desse ponto, o sistema estará operacional, permitindo ao usuário realizar pesquisas científicas automatizadas com suporte dos agentes especializados.

3.5. Implementação Prática

Nesta seção, será apresentado um guia passo a passo para desenvolver, testar e validar um agente de pesquisa científica utilizando o *framework LangGraph*. A solução integra buscas gerais na *web* com *Tavily*, consultas científicas ao *arXiv* e a capacidade de agendamento de pesquisas periódicas. Os trechos de código a seguir foram extraídos e adaptados do repositório apresentado na seção anterior.

3.5.1. Definindo Variáveis de Ambiente

A primeira etapa do sistema consiste em garantir que as chaves das APIs estejam acessíveis por variáveis de ambiente. O código abaixo solicita essas chaves via terminal caso ainda não estejam definidas:

Listing 3.1: Definindo as APIs Keys

```
{
    def _set_env(var: str):
        if not os.environ.get(var):
            os.environ[var] = getpass.getpass(f"{var}: ")

    _set_env("TAVILY_API_KEY")
    _set_env("ANTHROPIC_API_KEY")
}
```

Esse método permite maior segurança e flexibilidade, evitando o hardcode de credenciais sensíveis.

3.5.2. Inicialização do Modelo de Linguagem e do Banco Vetorial

O sistema utiliza o modelo `Claude 3.5 Sonnet` da *Anthropic* como núcleo dos agentes, além de um banco vetorial local com o `ChromaDB` para armazenamento e consulta semântica dos artigos do arXiv:

Listing 3.2: Configurando LLM e chromaDB

```
{
    # --- Modelo base ---
    llm = init_chat_model("anthropic:claude-3-5-sonnet-latest")

    # 3. Configura ChromaDB
    embeddings = HuggingFaceEmbeddings(model_name="sentence-
        transformers/all-MiniLM-L6-v2")
    vectorstore = Chroma(
        collection_name="artigos_arxiv",
        embedding_function=embeddings,
        persist_directory="./chroma_db"
)
}
```

3.5.3. Criação dos Agentes Especializados

A seguir, são definidos agentes especializados para diferentes tarefas, cada um com suas ferramentas e instruções específicas. Cada agente segue o padrão *ReAct*, ou seja, combina raciocínio com ações sobre ferramentas:

Listing 3.3: Configurando os agentes

```
{

agent = create_react_agent(model=llm, tools=tools)

    # Agente Tavily (busca geral)
tavily_agent = create_react_agent(
    model=llm,
    tools=[tavily_tool],
    prompt= 'You perform web searches',
    name="tavily_agent"
)

# Agente arXiv (pesquisa cient fica)
arxiv_agent = create_react_agent(
    model=llm, tools=[arxiv_search_and_ingest], name="arxiv_agent",
    prompt="You search, ingest into ChromaDB and report arXiv papers."
)

sched_agent  = create_react_agent(
    llm,
    tools=[schedule_research, cancel_research],
    prompt="You schedule or cancel periodic arXiv searches.",
    name="scheduler_agent"
)
)
}
```

3.5.4. Supervisor e Orquestração Multiagente

O supervisor é o componente central do sistema. Ele analisa a solicitação do usuário e decide qual agente deverá ser acionado.

Listing 3.4: Configurando e compilando o supervisor

```
{

supervisor_graph = create_supervisor(
    model=llm,
    agents=[tavily_agent, arxiv_agent, sched_agent],
    prompt=(
        "Voc     um supervisor que delega ao agente mais adequado
           entre:\n"
        "- Tavily (buscas gerais)\n"
        "- arXiv (buscas imediatas)\n"
        "- Scheduler (buscas agendadas)\n"
        "Ap s obter a resposta, sempre gere UMA mensagem final clara
           ao usu rio."
    ),
    add_handoff_messages=True, #permite rastrear a comunica  o entre
       supervisor e agentes, tornando a execu  o mais interpret vel
       .
    add_handoff_back_messages=True,
    output_mode="full_history"
)

compiled_supervisor = supervisor_graph.compile()
}
```

3.5.5. Compilando o Grafo de Estado com LangGraph

Após definir os componentes, o sistema é estruturado como um grafo de estado que inicia no supervisor. A estrutura abaixo permite escalar o sistema com mais nós e ramificações no futuro, mantendo a modularidade e o controle sobre os estados e transições.

Listing 3.5: Monta StateGraph e compila

```
{

class StateSchema(TypedDict):
    messages: Annotated[list[BaseMessage], add_messages]

graph = StateGraph(StateSchema)
graph.add_node("supervisor", compiled_supervisor)
graph.add_edge(START, "supervisor")
compiled = graph.compile(checkpointer=MemorySaver())
}
```

3.5.6. Execução Interativa do Sistema

Por fim, é definido um loop de execução interativa, permitindo ao usuário conversar diretamente com o sistema no terminal:

Listing 3.6: Loop interativo

```
{

if __name__ == "__main__":
    while True:
        ui = input("User: ")
        if ui.lower() in ["exit","quit","q"]:
            print("Encerrando...")
            break
        run(ui)
}
```

Esse loop inicia a inferência do grafo de agentes com base na entrada do usuário, realizando as delegações e execuções necessárias.

A Figura 3.6 ilustra a arquitetura interna do sistema multiagente implementado com o *framework LangGraph*. Diferentemente do diagrama inicial, que apresenta uma visão geral do sistema proposto, este diagrama detalha o fluxo de interação entre o usuário, o supervisor e os agentes especializados *(Tavily, arXiv e Scheduler)*, bem como sua integração com o banco vetorial *ChromaDB*. Essa estrutura possibilita a delegação inteligente de tarefas de busca e agendamento, promovendo uma execução dinâmica e modular baseada em agentes autônomos coordenados por um nó supervisor.

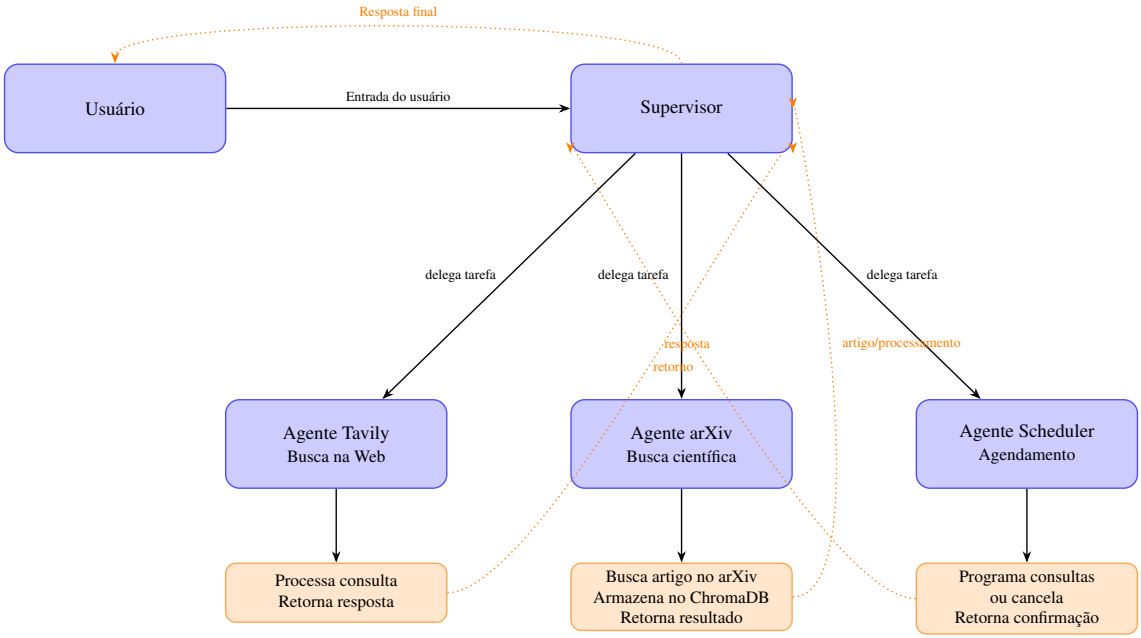


Figure 3.6: Arquitetura do sistema multiagente com supervisor e subagentes especializados.

3.6. Otimização e Melhoria

A construção de agentes baseados em modelos de linguagem de última geração requer não apenas boas práticas de desenvolvimento, mas também estratégias que assegurem confiabilidade, interpretabilidade e desempenho contínuo. Esta seção discute técnicas

aplicáveis à redução de alucinações, métodos para elevar a robustez das respostas geradas e caminhos para evolução futura do sistema proposto.

3.6.1. Redução de Alucinações por LLMs

Embora os LLMs sejam altamente eficazes na geração de texto natural, eles estão sujeitos a um fenômeno conhecido como alucinação, em que informações incorretas ou inventadas são apresentadas como verdadeiras. Para mitigar esse problema no agente proposto, é possível aplicar duas estratégias complementares:

- ***Re-ranking* de Resultados**: Após a obtenção de múltiplas respostas a partir de fontes externas (como buscas na *web* ou no *arXiv*), um mecanismo de reclassificação pode ser utilizado para ordenar os resultados com base em critérios como similaridade semântica ao *prompt* original, confiabilidade da fonte e presença de citações verificáveis.

- **Checagem Cruzada (*Cross-checking*)**: A verificação cruzada entre diferentes agentes ou ferramentas, como o *Tavily* (busca *web*) e o agente do arXiv (pesquisa científica), permite confirmar a consistência das informações recuperadas. Essa abordagem fortalece a confiança do sistema ao validar evidências em múltiplas fontes independentes (ALMEIDA DA SILVA et al., 2024).

Além disso, a utilização de *retrieval-augmented generation* (RAG) com base vetorial, como *ChromaDB*, também contribui para a fundamentação das respostas, ancorando-as em documentos reais e previamente armazenados.

3.6.2. Aumento da Confiabilidade do Agente

Para garantir maior confiabilidade ao longo do uso do agente, propõem-se medidas como:

- **Auditoria de Histórico de Conversas**: Armazenar os diálogos completos permite reanalisar decisões, treinar classificadores de erros e melhorar respostas futuras.

- ***Logs* Explicáveis por Estado**: Como o agente é construído com *LangGraph*, que segue uma estrutura baseada em grafos de estado, cada transição pode ser registrada de forma transparente, oferecendo interpretabilidade das ações realizadas pelo sistema.

- **Avaliação Contínua por Métricas Customizadas**: Métricas específicas podem ser desenvolvidas para avaliar a precisão das respostas científicas, considerando fatores como uso de fontes primárias, atualidade da informação e cobertura temática.

A combinação dessas práticas reforça a confiança dos usuários no sistema, promovendo maior adoção e segurança em aplicações reais, especialmente no domínio acadêmico.

3.6.3. Perspectivas de Evolução do Projeto

Após a realização do minicurso, diversas direções podem ser exploradas para ampliar o escopo e a sofisticação do agente:

- **Integração com bases de dados institucionais**: Permitir buscas internas em repositórios acadêmicos locais (como o SUAP ou repositórios do IFPI) ampliaria a utilidade do agente para estudantes e pesquisadores da própria instituição.

- **Personalização por Perfil de Usuário**: Implementar perfis adaptativos (aluno, professor, pesquisador) permitiria ajustar o comportamento do agente de acordo com as necessidades específicas de cada grupo.

- **Avaliação Colaborativa por Comunidade**: A criação de mecanismos para que os usuários avaliem as respostas, sugiram melhorias ou reportem inconsistências contribuiria para uma melhoria contínua baseada em dados reais de uso.

- **Multimodalidade e Explicações Visuais**: Com o avanço de modelos capazes de interpretar e gerar imagens, gráficos e vídeos, o agente poderia incorporar respostas multimodais para enriquecer a experiência de busca científica.

A incorporação dessas melhorias visa não apenas ampliar a funcionalidade do sistema, mas também consolidá-lo como uma ferramenta de apoio confiável e adaptável em contextos educacionais e de pesquisa.

3.7. Considerações Éticas

O desenvolvimento e a aplicação de agentes inteligentes para apoio à pesquisa científica exigem atenção rigorosa às implicações éticas, especialmente no que se refere à confiabilidade das informações geradas, à transparência das decisões tomadas e ao impacto social de seu uso. Esta seção discute aspectos centrais relacionados ao viés algorítmico, à explicabilidade dos modelos e à responsabilidade no uso da IA em contextos científicos.

3.7.1. Viés Algorítmico

Os modelos de LLMs, como os utilizados neste projeto, são treinados com vastos volumes de dados disponíveis na internet, muitos dos quais refletem desigualdades sociais, estereótipos e preconceitos históricos. Isso pode resultar em comportamentos enviesados, mesmo em tarefas técnicas ou aparentemente neutras. No contexto da ciência, esses vieses podem influenciar, por exemplo, o tipo de artigo recuperado, as fontes priorizadas ou as interpretações sugeridas pelos agentes. Como discutido por (BENDER et al., 2021), o viés não é apenas um problema técnico, mas também epistemológico, uma vez que afeta a forma como o conhecimento é produzido e difundido.

Portanto, é fundamental incorporar mecanismos de mitigação de viés, como *re-ranking* supervisionado, validação cruzada com múltiplas fontes e auditoria contínua dos resultados gerados pelos agentes.

3.7.2. Transparência e Explicabilidade

Outro aspecto crítico diz respeito à transparência dos sistemas baseados em IA. A opacidade de modelos como o *Claude* ou o *GPT-4.5* dificulta a compreensão do processo pelo qual uma determinada resposta é produzida. Isso compromete a capacidade do usuário de avaliar a confiabilidade das informações e dificulta a rastreabilidade de erros ou desvios.

Para mitigar essa limitação, é necessário empregar abordagens de RAG, que permitem rastrear a origem dos conteúdos apresentados. Além disso, o uso de *logs* detalhados do agente supervisor oferece maior visibilidade sobre o fluxo de decisões internas, ampliando a explicabilidade do sistema (LEWIS et al., 2021). A transparência é não apenas um requisito técnico, mas também ético, especialmente quando os agentes são aplicados em contextos científicos e educacionais.

3.7.3. Responsabilidade no Uso da IA em Ciência

A introdução de agentes autônomos para apoiar a produção de conhecimento científico impõe desafios significativos em termos de responsabilidade. Embora esses sistemas ampliem a capacidade de busca e análise, eles não substituem o papel crítico do pesquisador humano na interpretação dos dados e na formulação de hipóteses.

É necessário estabelecer limites claros de uso, bem como orientações éticas para a adoção responsável dessas tecnologias. Além disso, reforça-se a importância da formação dos usuários quanto às limitações dos sistemas de IA, para evitar uma confiança cega nas respostas geradas (PEROV; PEROVA, 2024).A responsabilização deve ser compartilhada entre desenvolvedores, usuários e instituições que promovem o uso de IA em ambientes acadêmicos.

Por fim, propõe-se que iniciativas como esta incluam diretrizes explícitas de ética em IA, como parte dos materiais educacionais do projeto, promovendo uma cultura de uso crítico, consciente e transparente da inteligência artificial.

3.8. Conclusão

Este capítulo apresentou, de forma prática e fundamentada, o desenvolvimento de um sistema multiagente de apoio à pesquisa científica, utilizando a biblioteca ***LangGraph***. A proposta buscou demonstrar como agentes inteligentes, quando bem arquitetados e integrados a ferramentas específicas como o ***Tavily AI*** e o ***arXiv***, podem auxiliar pesquisadores em tarefas como levantamento bibliográfico, busca em bases científicas e agendamento automatizado de pesquisas periódicas.

Ao longo da implementação, foram abordadas desde questões de configuração do ambiente de desenvolvimento até a estruturação do grafo supervisor, que permite a coordenação entre agentes especializados. A abordagem modular e interpretável do LangGraph mostrou-se eficaz para a construção de fluxos interativos mais transparentes, além de facilitar a extensibilidade do sistema.

Entre os principais aprendizados, destaca-se a importância de pensar a inteligência artificial não como substituta da atividade científica, mas como ferramenta complementar que amplifica a capacidade humana de análise, filtragem e interpretação de informações. A combinação entre grandes modelos de linguagem, bancos vetoriais e mecanismos de

agendamento configura um cenário promissor para o uso de IA em atividades de apoio à ciência, especialmente em tempos de sobrecarga informacional.

O impacto esperado da aplicação prática do *LangGraph* vai além da automação. Espera-se que essa tecnologia contribua para a democratização do acesso à informação científica, otimize processos de revisão de literatura e reduza barreiras técnicas para pesquisadores iniciantes ou com menor familiaridade com ferramentas computacionais.

Como caminhos futuros, vislumbra-se a integração do sistema com outras bases científicas e plataformas de gestão científica, o aperfeiçoamento dos mecanismos de avaliação da relevância dos artigos encontrados e a adoção de técnicas de *fine-tuning* para adequar os agentes aos domínios específicos de diferentes áreas do conhecimento. Além disso, propõe-se o avanço em estratégias de mitigação de vieses e a incorporação de métricas de confiabilidade nos fluxos de decisão dos agentes.

Por fim, acredita-se que a formação de pesquisadores sobre o uso ético, crítico e transparente da IA será determinante para o sucesso de iniciativas como esta, que buscam alinhar inovação tecnológica e integridade científica.

References

ALMEIDA DA SILVA, Wildemarkes de et al. Mitigation of Hallucinations in Language Models in Education: A New Approach of Comparative and Cross-Verification. In: 2024 IEEE International Conference on Advanced Learning Technologies (ICALT). [S.l.: s.n.], 2024. P. 207–209. DOI: 10.1109/ICALT61570.2024.00066.

ANTHROPIC. **Anthropic key**. [S.l.: s.n.], 2024. Acessado em 20 de março de 2025. Available from: <https://www.anthropic.com/>.

BARBARROXA, Rafael; GOMES, Luis; VALE, Zita. Benchmarking Large Language Models for Multi-agent Systems: A Comparative Analysis of AutoGen, CrewAI, and TaskWeaver. In_____. **Advances in Practical Applications of Agents, Multi-Agent Systems, and Digital Twins: The PAAMS Collection**. Cham: Springer Nature Switzerland, 2024. P. 39–48. ISBN 978-3-031-70415-4.

BENDER, Emily M. et al. On the Dangers of Stochastic Parrots: Can Language Models Be Too Big? In: PROCEEDINGS of the 2021 ACM Conference on Fairness, Accountability, and Transparency. Virtual Event, Canada: Association for Computing Machinery, 2021. (FAccT '21), p. 610–623. ISBN 9781450383097. DOI: 10.1145/3442188.3445922. Available from: <https://doi.org/10.1145/3442188.3445922>.

CHASE, Harrison. **LangGraph Documentation**. [S.l.: s.n.], 2024. Acessado em 14 de março de 2025. Available from: <https://langchain-ai.github.io/langgraph/>.

DUAN, Zhihua; WANG, Jialin. **Exploration of LLM Multi-Agent Application Implementation Based on LangGraph+CrewAI**. [S.l.: s.n.], 2024. arXiv: 2411.18241 [cs.MA]. Available from: <https://arxiv.org/abs/2411.18241>.

FIORILLO, Luca; MEHTA, Vini. Accelerating editorial processes in scientific journals: Leveraging AI for rapid manuscript review. **Oral Oncology Reports**, v. 10, p. 100511, 2024. ISSN 2772-9060. DOI: https://doi.org/10.1016/j.oor.2024.100511. Available from: <https://www.sciencedirect.com/science/article/pii/S2772906024003571>.

KHANDARE, Anand et al. Analysis of Python Libraries for Artificial Intelligence. In_____. **Intelligent Computing and Networking**. Singapore: Springer Nature Singapore, 2023. P. 157–177. ISBN 978-981-99-0071-8.

LEWIS, Patrick et al. **Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks**. [S.l.: s.n.], 2021. arXiv: 2005.11401 [cs.CL]. Available from: <https://arxiv.org/abs/2005.11401>.

LUO, Tianze et al. Multi-Agent Collaborative Exploration through Graph-based Deep Reinforcement Learning. In: 2019 IEEE International Conference on Agents (ICA). [S.l.: s.n.], 2019. P. 2–7. DOI: 10.1109/AGENTS.2019.8929168.

MANZOOR, Umar et al. Automating Academic Tasks (AAT) - An Agent Based Approach. In: 2012 IEEE 14th International Conference on High Performance Computing and Communication 2012 IEEE 9th International Conference on Embedded Software and Systems. [S.l.: s.n.], 2012. P. 1770–1775. DOI: 10.1109/HPCC.2012.266.

MATHUR, Srushti; CHHABRA, Aayush. Vector Search Algorithms: A Brief Survey. In: 2024 4th International Conference on Ubiquitous Computing and Intelligent Information Systems (ICUIS). [S.l.: s.n.], 2024. P. 365–371. DOI: 10.1109/ICUIS64676.2024.10866377.

MELONI, Antonello et al. Integrating Conversational Agents and Knowledge Graphs Within the Scholarly Domain. **IEEE Access**, v. 11, p. 22468–22489, 2023. DOI: 10.1109/ACCESS.2023.3253388.

PEROV, Vadim; PEROVA, Nina. AI Hallucinations: Is “Artificial Evil” Possible? In: 2024 IEEE Ural-Siberian Conference on Biomedical Engineering, Radioelectronics and Information Technology (USBEREIT). [S.l.: s.n.], 2024. P. 114–117. DOI: 10.1109/USBEREIT61901.2024.10584048.

TAVILY. **Tavily key**. [S.l.: s.n.], 2024. Acessado em 20 de março de 2025. Available from: <https://www.tavily.com/>.

Capítulo 4

Deep Learning na Prática: Classificando Imagens Faciais com Redes Neurais Convolucionais

Caio Pereira, Rodrigo Borges, José Rodrigues, Rodrigo Veras e Kelson Aires

Abstract

The advancement of Deep Learning techniques has transformed the way we analyze images, especially in the healthcare field, where computer vision holds great potential for assisting in diagnostics. This short course presents the fundamentals of image classification with Deep Learning, demonstrated in practice through an application involving facial images in the health domain. Participants will learn how to plan, conduct, and evaluate the model development process, from data preprocessing and augmentation to result analysis, with a focus on methodological rigor and reproducibility. Techniques involving pre-trained networks, support tools, and best evaluation practices will be discussed, preparing students and professionals for solid projects in Computer Vision.

Resumo

O avanço das técnicas de Deep Learning tem transformado o modo como analisamos imagens, especialmente na área da saúde, onde a visão computacional tem potencial para auxiliar em diagnósticos. Este minicurso apresenta os fundamentos de classificação de imagens com Deep Learning, mostrados na prática com uma aplicação em imagens faciais relacionadas à área da Saúde. Os participantes aprenderão a planejar, conduzir e avaliar o processo de desenvolvimento de um modelo, desde o pré-processamento e aumento de dados até a análise de resultados, com foco no rigor metodológico e na reprodutibilidade. Serão discutidas técnicas de uso de redes pré-treinadas, ferramentas de apoio e boas práticas de avaliação, preparando estudantes e profissionais para projetos sólidos em Visão Computacional.

4.1. Introdução

Nos últimos anos, a área de Visão Computacional tem sido significativamente influenciada pelo avanço das técnicas de *Deep Learning* (Aprendizado Profundo). O uso de

Redes Neurais Convolucionais (CNNs) tem possibilitado progressos importantes em tarefas como identificação de objetos, reconhecimento facial e interpretação de expressões, aproximando o desempenho computacional ao nível humano em diversos contextos [Goodfellow et al. 2016].

A análise de imagens faciais, em particular, emergiu como um dos campos mais promissores para aplicações de *Deep Learning* na área da saúde. As faces humanas contêm inúmeras informações que vão além da identidade: podem revelar emoções, estados cognitivos e, potencialmente, indicadores sutis de condições neurológicas e psiquiátricas [Alves Rodrigues et al. 2023].

As CNNs, diferentemente dos métodos tradicionais que dependiam de extração manual de características, possuem a capacidade de aprender automaticamente as características mais relevantes diretamente das imagens. Essa propriedade é particularmente valiosa quando lidamos com características faciais sutis que podem não ser facilmente percebidas pelo olho humano ou descritas por métodos convencionais [Pereira et al. 2024].

4.1.1. Representação Digital de Imagens

Para compreender como as CNNs processam imagens, é fundamental entender como as imagens são representadas digitalmente. Uma imagem digital pode ser definida como a representação visual de um objeto através de uma função bidimensional f(x,y), onde x e y são coordenadas espaciais e o valor de f em qualquer ponto (x,y) corresponde à intensidade ou nível de cinza naquele ponto. Na figura 4.1, temos um exemplo da representação digital de uma imagem [Gonzalez and Woods 2002].

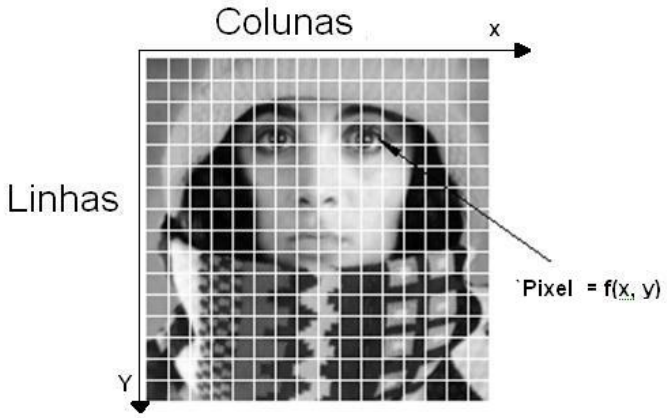


Figura 4.1. Representação digital de uma imagem. Fonte: [Claro et al. 2020]

Na prática, uma imagem digital é representada por uma matriz de elementos discretos chamados *pixels*. Para imagens em escala de cinza, cada *pixel* é representado por um único valor numérico que indica sua intensidade, geralmente variando de 0 (preto) a 255 (branco). Já em imagens coloridas, cada *pixel* é tipicamente representado por três valores correspondentes às intensidades dos canais vermelho (R), verde (G) e azul (B), o chamado modelo RGB. A Figura 4.2 ilustra essa representação matricial de uma imagem digital [Marques Filho and Neto 1999].

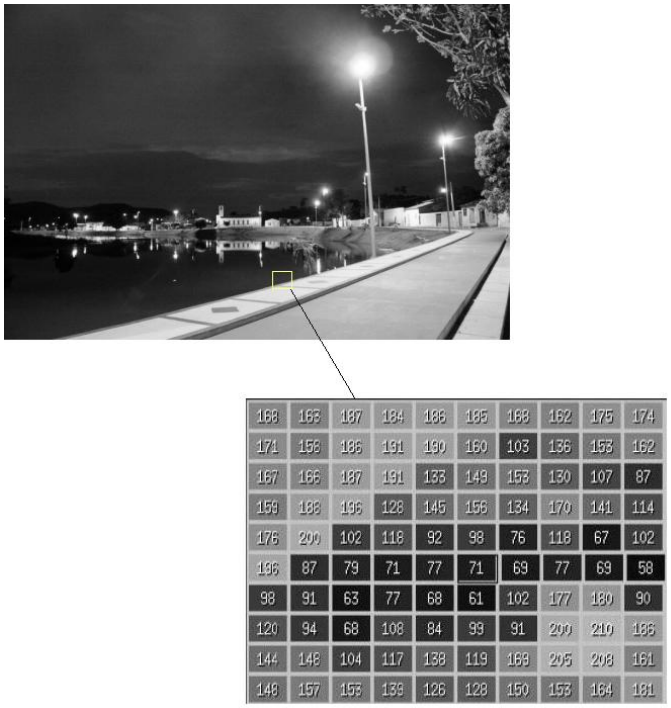


Figura 4.2. Representação matricial de uma imagem digital em escala de cinza com valores de intensidade para cada pixel. Fonte: [Claro et al. 2020]

4.1.2. Aprendizado de Máquina, Redes Neurais Artificiais e *Deep Learning*

O Aprendizado de Máquina (*Machine Learning*) é uma subárea da Inteligência Artificial que desenvolve algoritmos capazes de aprender padrões a partir de dados, sem necessidade de programação explícita. Estes algoritmos utilizam princípios de indução para generalizar a partir de exemplos, criando hipóteses aplicáveis a novos dados [Lorena et al. 2021].

No contexto de classificação de imagens, o Aprendizado Supervisionado é particularmente relevante: o algoritmo aprende a mapear entradas (imagens) para saídas (classes ou rótulos) a partir de um conjunto de dados rotulados. Por exemplo, em um sistema de classificação facial para Transtorno do Espectro Autista (TEA), o modelo seria treinado com imagens faciais rotuladas como 'com TEA' ou 'sem TEA' [Alves Rodrigues et al. 2023].

As Redes Neurais Artificiais (RNAs) são modelos matemáticos inspirados na estrutura e funcionamento do cérebro humano. Uma RNA consiste em unidades de processamento interconectadas (neurônios artificiais) organizadas em camadas. Cada conexão entre neurônios possui um peso associado que é ajustado durante o processo de aprendizagem [Haykin 2007]. A Figura 4.3 ilustra um exemplo de uma rede neural artificial, destacando sua estrutura típica composta por três tipos principais de camadas:

- Camada de entrada: Recebe os dados (no caso de imagens, os valores dos pixels);
- Camadas intermediárias ou ocultas: Realizam transformações não-lineares nos dados;

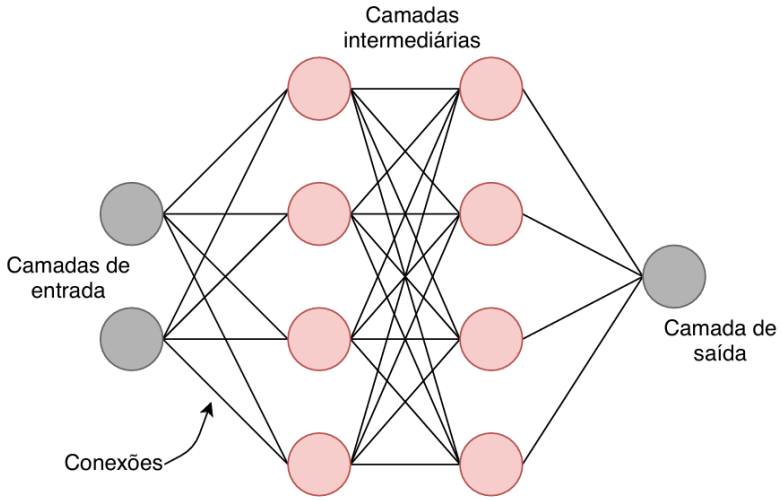


Figura 4.3. Exemplo de uma rede neural artificial. Fonte: [Vogado et al. 2019]

- Camada de saída: Produz a classificação final.

O processo de aprendizagem em RNAs envolve o ajuste gradual dos pesos das conexões para minimizar a diferença entre as previsões da rede e os valores reais. Isso é feito através de algoritmos como o *backpropagation*, que propaga o erro da camada de saída para as camadas anteriores [Rezende 2003].

O *Deep Learning* (Aprendizado Profundo) representa uma evolução das RNAs tradicionais, caracterizado pelo uso de múltiplas camadas intermediárias – daí o termo "profundo"– que permitem a extração automática de características em diferentes níveis de abstração [Goodfellow et al. 2016].

4.1.3. Aplicações em Imagens Faciais

A análise de imagens faciais através de *Deep Learning* tem encontrado aplicações diversas e impactantes em múltiplos campos, com destaque especial para a área da saúde. Esses sistemas têm demonstrado capacidade de detectar padrões sutis que podem estar associados a condições neurológicas, genéticas e psiquiátricas.

No contexto específico do Transtorno do Espectro Autista (TEA), pesquisas recentes têm explorado o potencial das CNNs para identificar biomarcadores faciais que possam auxiliar no diagnóstico precoce. O TEA é um transtorno do neurodesenvolvimento caracterizado por dificuldades na comunicação social e padrões restritos e repetitivos de comportamento, interesses ou atividades [American Psychiatric Association 2013].

O diagnóstico precoce do TEA é crucial para intervenções terapêuticas oportunas, que podem significativamente melhorar o prognóstico. No entanto, o diagnóstico tradicional baseia-se principalmente em observações comportamentais, podendo ser tardio e subjetivo. A identificação de biomarcadores objetivos, como alterações sutis na morfologia facial, representa uma abordagem promissora para complementar os métodos diagnósticos existentes [Alves Rodrigues et al. 2023].

Outras aplicações relevantes de *Deep Learning* em imagens faciais na área da saúde incluem a identificação de síndromes genéticas, por meio da detecção de características faciais associadas a condições como síndrome de Down, Williams e DiGeorge. Também é possível realizar a análise de expressões faciais para avaliar respostas emocionais em transtornos como depressão e esquizofrenia. Além disso, essas técnicas permitem estimar a idade biológica, identificando sinais de envelhecimento precoce relacionados a condições metabólicas, e avaliar a simetria facial, detectando assimetrias sutis que podem ser indicativas de problemas neurológicos.

É importante destacar que sistemas baseados em *Deep Learning* para análise facial em contextos médicos devem ser considerados ferramentas complementares e não substitutivas ao julgamento clínico. Nos próximos capítulos, exploraremos em detalhes a teoria e a prática de desenvolvimento de CNNs para classificação de imagens faciais, com foco em aplicações relacionadas à saúde, particularmente na detecção de características associadas ao TEA.

4.2. Redes Neurais Convolucionais

As Redes Neurais Convolucionais (CNNs) representam uma classe especializada de redes neurais particularmente eficaz para processamento e análise de imagens. Diferentemente das redes neurais tradicionais, as CNNs são projetadas para explorar a estrutura espacial presente em dados como imagens, aproveitando propriedades como localidade espacial e invariância à translação [LeCun et al. 1998].

O conceito de CNN foi inicialmente apresentado por Yann LeCun e Fukushima na década de 90, mas só ganhou ampla popularidade no século XXI com o aumento do poder computacional e a disponibilidade de grandes conjuntos de dados [Fukushima 1988, LeCun et al. 1998]. Hoje, as CNNs constituem o estado da arte em praticamente todas as tarefas de visão computacional, desde classificação e detecção de objetos até segmentação semântica e reconhecimento facial.

A principal vantagem das CNNs sobre outras abordagens é sua capacidade de aprender automaticamente hierarquias de características. Nas primeiras camadas, a rede aprende características básicas como bordas e texturas; em camadas intermediárias, padrões mais complexos como olhos, narizes e bocas; e nas camadas finais, representações abstratas que permitem a classificação. Este aprendizado hierárquico é particularmente valioso para análise facial, onde diferentes níveis de características são relevantes [Goodfellow et al. 2016].

4.2.1. Componentes

As CNNs são compostas por diversos tipos de camadas, cada uma com funções específicas no processo de extração de características e classificação. Vamos explorar os principais componentes:

Camadas Convolucionais: A operação de convolução é o coração das CNNs. Nesta operação, filtros (ou *kernels*) deslizam sobre a imagem de entrada, realizando multiplicações elemento a elemento e somas, produzindo mapas de características (*feature maps*) que destacam padrões específicos na imagem [Goodfellow et al. 2016]. Em aplicações de

análise facial, as camadas convolucionais são cruciais para extrair características morfológicas e texturais. Filtros nas primeiras camadas detectam bordas e texturas fundamentais, enquanto filtros em camadas mais profundas capturam estruturas faciais complexas [Alves Rodrigues et al. 2023]. A Figura 4.4 ilustra um exemplo de operação de convolução, onde um filtro 3×3 é aplicado sobre a imagem de entrada, gerando um mapa de características.

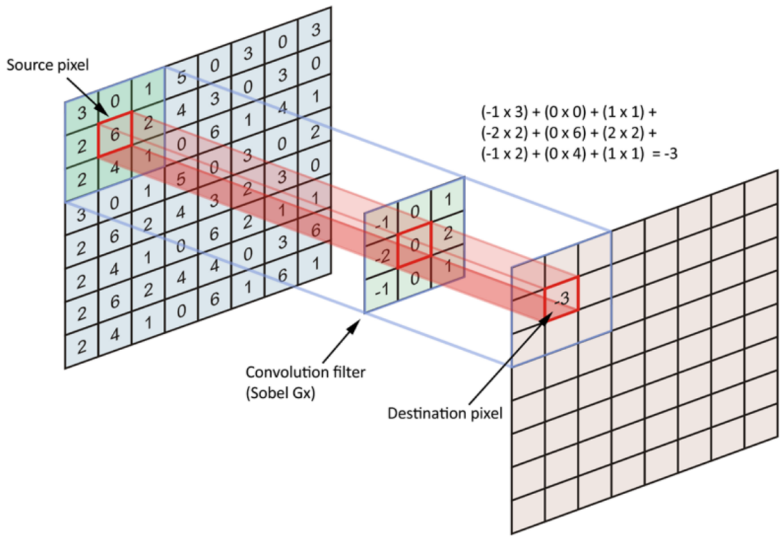


Figura 4.4. Ilustração da operação de convolução aplicada a uma imagem de entrada utilizando um filtro Sobel (Gx). O filtro 3×3 realiza multiplicações elemento a elemento com uma região da imagem, e os resultados são somados para gerar um valor no mapa de características. Esse processo permite a detecção de bordas horizontais, destacando padrões estruturais importantes na imagem. Fonte: [Hachilif et al. 2019]

Camadas de *Pooling*: Após as camadas convolucionais, geralmente são aplicadas camadas de *pooling*, que reduzem a dimensionalidade espacial dos mapas de características. Esta redução diminui o custo computacional e ajuda a rede a se tornar invariante a pequenas translações e distorções [Scherer et al. 2010]. Os dois tipos mais comuns de *pooling* são o *Max Pooling* e o *Average Pooling*. O *Max Pooling*, por exemplo, seleciona o valor máximo em cada região do mapa de características, preservando as informações mais proeminentes, como ilustrado na Figura 4.5. Nela, uma janela de 2x2 percorre a matriz de entrada e extrai o maior valor de cada quadrante para formar uma nova matriz de saída com dimensões reduzidas. Já o *Average Pooling* calcula a média dos valores em cada região, o que resulta em uma suavização das características. Na prática, o *Max Pooling* é mais utilizado, pois tende a preservar informações discriminativas importantes enquanto descarta detalhes menos relevantes [Boureau et al. 2010].

Camadas de Ativação: As funções de ativação introduzem não-linearidades no modelo, permitindo que a rede aprenda representações complexas. Sem estas não-linearidades, a rede se comportaria como uma única transformação linear, limitando severamente sua capacidade de aprendizado [Glorot et al. 2011]. As principais funções de ativação incluem:

- ReLU (*Rectified Linear Unit*): A mais popular devido à sua simplicidade e eficácia.

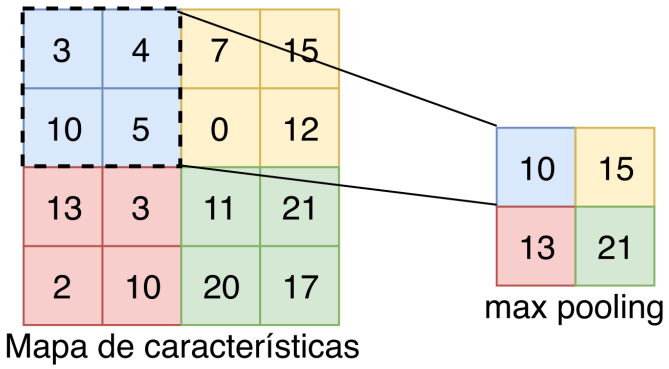


Figura 4.5. Operação de *Max Pooling* com uma janela 2×2. Fonte: [Vogado et al. 2019]

Substitui valores negativos por zero, evitando o problema do desaparecimento do gradiente em redes profundas;

- Sigmoid: Mapeia os valores para o intervalo [0,1], útil para a camada de saída em problemas de classificação binária;
- Softmax: Usada na camada de saída para problemas de classificação multiclasse, convertendo os valores em probabilidades que somam 1;
- LeakyReLU: Uma variação da ReLU que permite um pequeno gradiente negativo, evitando "neurônios mortos".

A escolha da função de ativação pode impactar significativamente o desempenho do modelo. Na prática, ReLU é frequentemente a escolha padrão para camadas intermediárias, enquanto Softmax é utilizada na camada de saída para classificação multiclasse [Nair and Hinton 2010].

Camadas *Flatten* e *Fully Connected*: Após várias camadas convolucionais e de *pooling*, os mapas de características são convertidos em um vetor unidimensional através da camada *Flatten*. Este vetor é então processado por uma ou mais camadas totalmente conectadas (*Fully Connected* ou *Dense*), onde cada neurônio está conectado a todos os neurônios da camada anterior.

As camadas *Fully Connected* realizam o raciocínio de alto nível, combinando todas as características extraídas pelas camadas convolucionais para realizar a classificação final. A última camada *Fully Connected* geralmente contém um número de neurônios igual ao número de classes, com uma função de ativação Softmax para gerar probabilidades de classificação [Goodfellow et al. 2016].

Em modelos para classificação do TEA, por exemplo, a camada final teria dois neurônios (com TEA/sem TEA), ou mais neurônios se estiver classificando diferentes subtipos ou níveis de severidade [Alves Rodrigues et al. 2023].

Estas camadas contêm a maior parte dos parâmetros da rede e, portanto, são mais propensas ao overfitting. Técnicas de regularização como Dropout são frequentemente aplicadas nestas camadas para melhorar a generalização do modelo [Srivastava et al. 2014].

4.2.2. Hiperparâmetros

Os hiperparâmetros são variáveis que controlam o processo de treinamento e a arquitetura da CNN. Diferentemente dos parâmetros (pesos e vieses) que são aprendidos durante o treinamento, os hiperparâmetros são definidos antes do início do treinamento e permanecem fixos durante todo o processo. A seleção adequada de hiperparâmetros é crucial para o desempenho do modelo [Bergstra and Bengio 2012].

Número e Tamanho das Camadas: A profundidade da rede (número de camadas) e o tamanho de cada camada (número de filtros ou neurônios) determinam a capacidade de representação do modelo. Redes mais profundas podem aprender representações mais complexas, mas também requerem mais dados e são mais propensas ao *overfitting* [Szegedy et al. 2015].

Estudos empíricos mostram que, em geral, é melhor usar redes mais profundas com camadas menores do que redes rasas com camadas maiores, pois isso permite aprender hierarquias de características mais ricas [He et al. 2016]. No entanto, a escolha ideal depende da complexidade do problema e da quantidade de dados disponíveis.

Taxa de Aprendizado: A taxa de aprendizado controla o tamanho dos passos durante a otimização. Uma taxa muito pequena resulta em convergência lenta, enquanto uma taxa muito grande pode impedir a convergência ou levar a mínimos locais subótimos [Ruder 2016].

Existem várias estratégias para definir a taxa de aprendizado. Uma delas é utilizar um valor fixo ao longo de todo o treinamento, como 0.001. Outra abordagem é a redução programada, na qual a taxa é diminuída em intervalos predefinidos ou quando o desempenho do modelo estagna. Também existem métodos de redução adaptativa, que ajustam automaticamente a taxa com base no desempenho, como o `ReduceLROnPlateau` no Keras. Além disso, programações cíclicas, como as propostas por [Smith 2017], fazem a taxa variar ciclicamente entre valores mínimo e máximo.

A escolha da taxa de aprendizado é frequentemente considerada o hiperparâmetro mais crítico em *Deep Learning*. Por isso, uma prática recomendada é realizar testes preliminares com diferentes valores, como 1×10^{-2}, 1×10^{-3}, 1×10^{-4}, para identificar o intervalo mais promissor para o problema específico [Bengio 2012].

Número de Épocas e Tamanho do *Batch*: Uma época representa uma passagem completa pelo conjunto de dados de treinamento. O número ideal de épocas depende da complexidade do problema e do tamanho do conjunto de dados. O treinamento deve continuar até que o modelo convirja, mas deve ser interrompido antes que ocorra *overfitting* [Goodfellow et al. 2016].

O tamanho do *batch* determina quantos exemplos são processados antes de atualizar os parâmetros do modelo. *Batches* maiores proporcionam estimativas de gradiente mais estáveis, mas requerem mais memória e podem levar a soluções com menor capacidade de generalização. *Batches* menores introduzem mais ruído no treinamento, o que pode ajudar a escapar de mínimos locais, mas pode tornar a convergência mais difícil [Keskar et al. 2017].

Em aplicações com conjuntos de dados limitados, como frequentemente ocorre em

estudos clínicos de TEA, técnicas como validação cruzada e *early stopping* são essenciais para determinar o número ideal de épocas e evitar *overfitting* [Alves Rodrigues et al. 2023].

Funções de Perda: A função de perda (*loss function*) quantifica a diferença entre as previsões do modelo e os valores reais, orientando o processo de otimização. A escolha da função de perda depende do tipo de problema [Janocha and Czarnecki 2017]. Para problemas de classificação, as funções mais comuns incluem:

- *Binary Cross-Entropy*: Para classificação binária (ex: com TEA/sem TEA).
- *Categorical Cross-Entropy*: Para classificação multiclasse (ex: diferentes subtipos de TEA).
- *Focal Loss*: Uma modificação da cross-entropy que atribui mais peso a exemplos difíceis, útil para conjuntos de dados desbalanceados [Lin et al. 2017].

Otimizadores: Os otimizadores controlam como os parâmetros do modelo são atualizados com base nos gradientes calculados. Cada otimizador tem características distintas que afetam a velocidade de convergência e a qualidade da solução final [Ruder 2016].

Na prática, Adam é frequentemente a escolha inicial devido ao seu bom desempenho geral e baixa necessidade de ajuste. No entanto, SGD com *momentum* às vezes atinge melhor generalização em tarefas de classificação de imagens [Wilson et al. 2017].

4.2.3. Arquiteturas

A evolução das arquiteturas de CNNs tem sido impulsionada principalmente pela competição *ImageNet Large Scale Visual Recognition Challenge* (ILSVRC), que desafiou pesquisadores a desenvolver modelos cada vez mais precisos para classificação e detecção de objetos em larga escala [Russakovsky et al. 2015].

Estas arquiteturas representam marcos importantes no desenvolvimento das CNNs e são frequentemente utilizadas como base para transferência de aprendizado em problemas específicos, como a classificação de imagens faciais para detecção de TEA.

DenseNet: Proposta por Huang et al. (2017), a *Densely Connected Convolutional Network* (DenseNet) estendeu a ideia de conexões de atalho, conectando cada camada a todas as camadas subsequentes dentro de um bloco denso. Essa arquitetura apresenta blocos densos nos quais cada camada recebe inputs de todas as camadas anteriores, permitindo a reutilização de características ao longo da rede. Além disso, *DenseNet* utiliza menos parâmetros, o que a torna menos propensa ao *overfitting*, e possibilita uma propagação eficiente do gradiente.

MobileNet: Desenvolvida por Howard et al. (2017), a *MobileNet* foi projetada especificamente para aplicações móveis e embarcadas com restrições de recursos computacionais. Essa arquitetura utiliza convoluções separáveis em profundidade (*depthwise separable convolutions*) e inclui hiperparâmetros que permitem controlar o tamanho e a latência do modelo. Comparada a arquiteturas similares, a *MobileNet* possui significativamente menos parâmetros e operações, garantindo bom desempenho para aplicações em tempo real.

A eficiência da *MobileNet* a torna particularmente útil para implementações em dispositivos móveis, potencialmente permitindo ferramentas de triagem de TEA acessíveis em regiões com recursos limitados [Alves Rodrigues et al. 2023].

Em aplicações de classificação facial para detecção de TEA, a escolha da arquitetura depende de vários fatores, incluindo tamanho do conjunto de dados, recursos computacionais disponíveis e requisitos de desempenho. Estudos recentes têm demonstrado bons resultados com arquiteturas como *DenseNet* e *MobileNet*, especialmente quando combinadas com técnicas de transferência de aprendizado [Alves Rodrigues et al. 2023, Pereira et al. 2024].

4.3. Construção de uma CNN

Após compreender os componentes e arquiteturas das CNNs, vamos explorar como construir uma rede convolucional básica utilizando Keras com TensorFlow. Para fins didáticos, desenvolveremos uma CNN genérica que segue a estrutura clássica introduzida por [LeCun et al. 1998] e refinada em trabalhos posteriores.

Nossa CNN será projetada para classificação binária de imagens faciais com dimensões 64×64×3, possuindo uma estrutura sequencial de camadas convolucionais, *pooling*, e camadas densas. Essa arquitetura é suficientemente complexa para demonstrar os princípios fundamentais, mas simples o bastante para facilitar a compreensão.

O Keras simplifica consideravelmente a implementação através de sua API intuitiva. O código 4.1 ilustra a construção da nossa CNN.

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D,
   Flatten, Dense, Dropout, Activation, BatchNormalization

# Definicao do modelo
model = Sequential()

# Blocos convolucionais
model.add(Conv2D(32, kernel_size=(3, 3), padding='same',
   input_shape=(64, 64, 3)))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(64, kernel_size=(3, 3), padding='same'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

# Classificacao
model.add(Flatten())
model.add(Dense(128))
model.add(BatchNormalization())
```

```
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(1))
model.add(Activation('sigmoid'))

# Compilacao
model.compile(
    optimizer='adam',
    loss='binary_crossentropy',
    metrics=['accuracy']
)
```

Listing 4.1. Definição e compilação de uma CNN moderna em Keras para classificação binária

A primeira camada convolucional utiliza 32 filtros 3×3 para extrair características básicas da imagem de entrada. A seguir, a camada *BatchNormalization* normaliza as ativações, o que estabiliza e acelera o processo de treinamento. A função de ativação ReLU introduz não-linearidade, e a camada *MaxPooling2D* reduz as dimensões espaciais pela metade. O segundo bloco convolucional segue o mesmo padrão, mas com 64 filtros, aumentando a capacidade de representação enquanto as dimensões espaciais continuam diminuindo.

Após extrair características através das camadas convolucionais, a operação *Flatten* transforma o mapa de características tridimensional em um vetor unidimensional. Este vetor alimenta uma camada densa com 128 neurônios, que também é seguida por *BatchNormalization* e ReLU. A camada de *Dropout* que desativa aleatoriamente 50% dos neurônios durante o treinamento para prevenir *overfitting*. A camada final utiliza ativação sigmoid para produzir a probabilidade da classe positiva.

Na etapa de compilação, definimos o otimizador *Adam* e a função de perda *binary_crossentropy*, que é a escolha padrão e matematicamente adequada para problemas de classificação binária. A acurácia foi selecionada como a métrica para monitorar o desempenho do modelo durante o treinamento e a avaliação.

Ao projetar CNNs para problemas específicos como a detecção de TEA em imagens faciais, algumas considerações importantes incluem a resolução das imagens, a profundidade da rede, e técnicas de regularização. Imagens faciais geralmente beneficiam-se de resoluções maiores (128×128 ou 224×224) para preservar características sutis. A arquitetura pode ser ajustada adicionando mais camadas convolucionais para aumentar a capacidade de representação, ou incorporando técnicas como *Batch Normalization*, já incluída em nosso exemplo para garantir um treinamento mais estável e eficiente.

4.4. Transferência da Aprendizado

A transferência de aprendizado representa uma das técnicas mais importantes no arsenal do *deep learning* moderno, especialmente quando lidamos com conjuntos de dados limitados, como frequentemente ocorre em aplicações médicas. Esta abordagem permite aproveitar o conhecimento adquirido por uma rede neural em uma tarefa para melhorar seu desempenho em outra tarefa relacionada, mesmo quando os domínios são diferentes.

[Pan and Yang 2010] definem transferência de aprendizado como a situação onde o que foi aprendido em um cenário é explorado para melhorar a generalização em outro cenário.

No contexto da classificação de imagens faciais para detecção de TEA, a transferência de aprendizado é particularmente valiosa. Treinar uma CNN do zero geralmente requer dezenas de milhares de imagens rotuladas, um volume raramente disponível em estudos clínicos. Além disso, o treinamento completo demanda recursos computacionais significativos e tempo considerável. A transferência de aprendizado oferece uma alternativa eficiente, permitindo que modelos pré-treinados em grandes conjuntos de dados como ImageNet (com mais de um milhão de imagens) sejam adaptados para tarefas específicas com muito menos dados [Tajbakhsh et al. 2016].

Existem duas abordagens principais para a transferência de aprendizado em CNNs: extração de características e ajuste fino (*fine-tuning*). Cada abordagem tem seus méritos e é adequada para diferentes cenários, dependendo da similaridade entre as tarefas de origem e destino, e da quantidade de dados disponíveis para a nova tarefa.

4.4.1. Extração de Características

A extração de características é uma abordagem direta de transferência de aprendizado que aproveita a capacidade das CNNs de aprender representações hierárquicas dos dados. Nas primeiras camadas, as CNNs aprendem características genéricas como bordas e texturas; nas camadas intermediárias, padrões mais complexos; e nas camadas finais, representações específicas para a tarefa original [Yosinski et al. 2014]. A ideia fundamental é utilizar uma rede pré-treinada como um extrator fixo de características, removendo as camadas de classificação final e mantendo o restante da rede congelado (pesos não treináveis).

Nesta abordagem, as imagens são processadas pela CNN pré-treinada até uma camada intermediária específica, gerando vetores de características que são então utilizados para treinar um classificador simples como SVM (*Support Vector Machine*) ou até mesmo uma rede neural mais simples. Esta técnica é particularmente eficaz quando os dados disponíveis são limitados e a tarefa alvo é significativamente diferente da tarefa original.

Para implementar a extração de características usando Keras, podemos utilizar um modelo pré-treinado como a *VGG16* ou *ResNet50*, removendo as camadas de classificação e congelando os pesos das camadas convolucionais. Vemos um exemplo no código 4.2.

```
from tensorflow.keras.applications import VGG16
from tensorflow.keras.models import Model

# Carregar modelo pre-treinado sem as camadas de classificacao
base_model = VGG16(weights='imagenet', include_top=False,
   input_shape=(224, 224, 3))

# Congelar todas as camadas (tornar nao treinaveis)
for layer in base_model.layers:
    layer.trainable = False

# Funcao para extrair caracteristicas de um conjunto de imagens
```

```
def extract_features(images):
    features = base_model.predict(images)
    # Reshape para vetor unidimensional
    features = features.reshape(features.shape[0], -1)
    return features
```

Listing 4.2. Extração de características com VGG16 pré-treinada

4.4.2. Ajuste Fino

O ajuste fino (*fine-tuning*) representa uma abordagem mais avançada de transferência de aprendizado, onde algumas ou todas as camadas da rede pré-treinada são retreinadas com uma taxa de aprendizado reduzida. Esta técnica permite que o modelo se adapte mais especificamente à nova tarefa, mantendo o conhecimento geral adquirido durante o pré-treinamento.

O processo geralmente começa com um modelo pré-treinado. Em seguida, substitui-se sua camada de saída por uma nova camada adaptada ao problema específico, e a rede é treinada com uma taxa de aprendizado menor. Normalmente, congela-se a maior parte das camadas da rede e treina-se inicialmente apenas as camadas adicionadas. Posteriormente, mais camadas podem ser "descongeladas" progressivamente para um refinamento adicional. Uma implementação básica de ajuste fino com Keras pode ser vista no código 4.3.

```
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.layers import Dense,
   GlobalAveragePooling2D
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam

# Carregar modelo base pre-treinado
base_model = ResNet50(weights='imagenet', include_top=False,
   input_shape=(224, 224, 3))

# Adicionar novas camadas de classificacao
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(256, activation='relu')(x)
predictions = Dense(1, activation='sigmoid')(x)

# Modelo final
model = Model(inputs=base_model.input, outputs=predictions)

# Congelar camadas do modelo base
for layer in base_model.layers:
    layer.trainable = False

# Primeira fase: treinar apenas as camadas adicionadas
model.compile(optimizer='adam', loss='binary_crossentropy',
   metrics=['accuracy'])
```

```
model.fit(train_data, train_labels, epochs=5, validation_data=(
    val_data, val_labels))

# Segunda fase: ajuste fino (fine-tuning)
# Descongelar algumas camadas do modelo base
for layer in base_model.layers[-10:]:
    layer.trainable = True

# Recompilar com taxa de aprendizado menor
model.compile(optimizer=Adam(learning_rate=1e-5), loss='
    binary_crossentropy', metrics=['accuracy'])
model.fit(train_data, train_labels, epochs=10, validation_data=(
    val_data, val_labels))
```

Listing 4.3. Modelo com saída sigmoid e perda binária

O ajuste fino geralmente produz modelos com melhor desempenho do que a simples extração de características, especialmente quando o conjunto de dados de destino tem tamanho moderado a grande. [Yosinski et al. 2014] demonstraram que mesmo quando as tarefas são aparentemente distantes, o ajuste fino pode melhorar significativamente o desempenho em comparação com o treinamento do zero.

Na prática, a estratégia de ajuste fino deve considerar diversos fatores. O número de camadas a descongelar depende do tamanho do conjunto de dados disponível e da similaridade entre as tarefas. Com conjuntos de dados pequenos, descongelar muitas camadas pode levar ao *overfitting*. A taxa de aprendizado deve ser significativamente menor (geralmente 10 a 100 vezes) que a utilizada para treinamento do zero, para evitar destruir o conhecimento previamente adquirido.

Para problemas de análise facial como a detecção de TEA, o ajuste fino tem demonstrado resultados promissores. [Alves Rodrigues et al. 2023] aplicaram esta técnica com a arquitetura DenseNet para classificar imagens faciais de indivíduos com e sem TEA, alcançando acurácias superiores a 90%. Os autores observaram que o ajuste fino de apenas as camadas finais produziu os melhores resultados, porque as características genéricas aprendidas nas camadas iniciais da rede já eram adequadas para capturar estruturas faciais relevantes.

4.5. Estudo de Caso

Para demonstrar a aplicação prática dos conceitos apresentados, desenvolvemos um estudo de caso focado na detecção do Transtorno do Espectro Autista (TEA) através de imagens faciais utilizando redes neurais convolucionais. Este caso ilustra como as técnicas de *deep learning* podem ser aplicadas em problemas reais da área da saúde, apresentando tanto as potencialidades quanto os desafios inerentes a essas aplicações.

4.5.1. Problema

O Transtorno do Espectro Autista é um distúrbio do neurodesenvolvimento caracterizado por dificuldades na comunicação social e interpessoal, retardo de linguagem, comportamento repetitivo e disfunção sensorial. Estima-se que uma em cada 44 crianças nos Esta-

dos Unidos seja acometida por esse transtorno, com números similares sendo observados globalmente [Centers for Disease Control and Prevention 2023].

A intervenção precoce desempenha papel crucial no desenvolvimento de indivíduos com TEA devido à plasticidade cerebral característica dos primeiros anos de vida. Os ganhos decorrentes da intervenção precoce podem reduzir consideravelmente os gastos familiares no tratamento das crianças com TEA, bem como os custos dos sistemas de saúde pública. Além disso, a intervenção precoce é de grande valor para reduzir os impactos do TEA sobre as famílias e a sociedade [Alves Rodrigues et al. 2023].

4.5.2. Base de Dados

Para este estudo de caso, utilizamos o "*Autism Image Data*"disponibilizado na plataforma Kaggle, que representa o único conjunto de dados publicamente disponível com imagens faciais bidimensionais estáticas de indivíduos com TEA. Este *dataset* foi criado especificamente para pesquisas em detecção automática de TEA utilizando técnicas de visão computacional.

O conjunto de dados é composto por 2.938 imagens faciais no formato JPEG com dimensões padronizadas de 224×224x3 *pixels*. As imagens foram coletadas através de buscas sistemáticas na web e processadas para focar nas características faciais, com remoção de fundos e recorte centrado no rosto. A distribuição do *dataset* é equilibrada entre as classes e organizada em conjuntos de treinamento (2.538 imagens, 86%), validação (200 imagens, 7%) e teste (200 imagens, 7%), com proporção igual entre indivíduos com TEA e desenvolvimento típico em cada divisão. Na figura 4.6, observamos exemplos das imagens contidas nesse *dataset*.

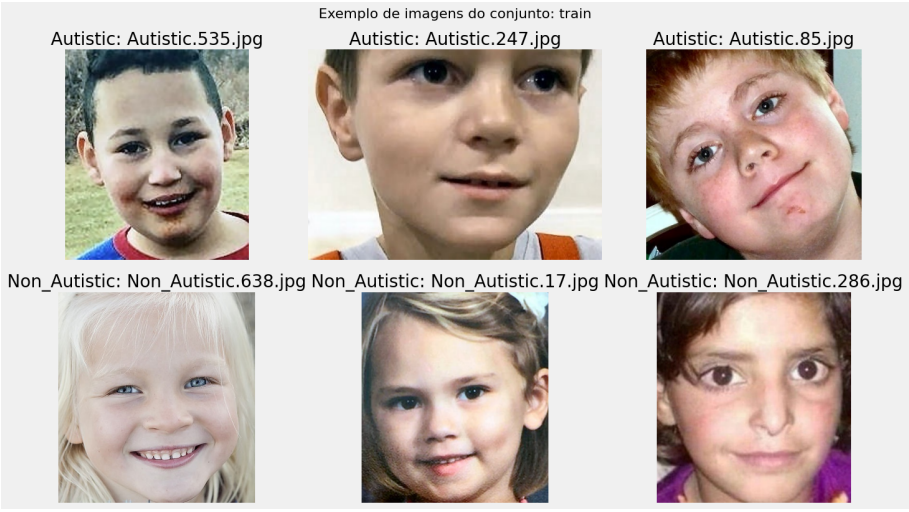


Figura 4.6. Exemplos de imagens do conjunto de treino, com amostras de indivíduos com (linha superior) e sem TEA (linha inferior). Fonte: https://www.kaggle.com/datasets/cihan063/autism-image-data

O pré-processamento incluiu normalização dos valores dos *pixels* para o intervalo [0, 1] e verificação da qualidade das imagens. Todas as imagens são coloridas e possuem qualidade adequada para preservar características faciais relevantes. É importante reconhecer as limitações inerentes a este conjunto de dados, incluindo variabilidade em

qualidade, iluminação e pose devido à coleta web, além da ausência de controle rigoroso sobre verificação independente dos diagnósticos. Essas limitações são típicas de estudos exploratórios e devem ser consideradas na interpretação dos resultados.

Apesar das limitações, este dataset tem sido amplamente utilizado na literatura científica como *benchmark* para avaliação de métodos de detecção automática de TEA, permitindo comparações consistentes entre diferentes abordagens metodológicas.

4.5.3. Aumento de Dados

O aumento de dados (*data augmentation*) representa uma técnica fundamental para melhorar a generalização de modelos de *deep learning*, especialmente quando trabalhamos com conjuntos de dados limitados. Esta técnica gera novos exemplos de treinamento a partir dos dados existentes através de transformações que preservam a classe da imagem, efetivamente aumentando o tamanho do conjunto de dados de forma artificial [Mumuni and Mumuni 2022].

Em análise de imagens faciais para detecção de TEA, o aumento de dados requer cuidado especial para evitar transformações que possam alterar características morfológicas importantes. Transformações como rotações extremas ou distorções perspectivas podem modificar relações espaciais cruciais entre características faciais, potencialmente removendo ou criando artificialmente os biomarcadores que estamos tentando detectar.

Para este estudo de caso, como mostra a figura 4.7, aplicamos apenas o espelhamento horizontal (*horizontal flip*), uma transformação que preserva todas as características morfológicas importantes enquanto dobra efetivamente o tamanho do conjunto de dados. Esta escolha conservadora é justificada pela natureza específica do problema, onde características faciais sutis podem ser fundamentais para a classificação.

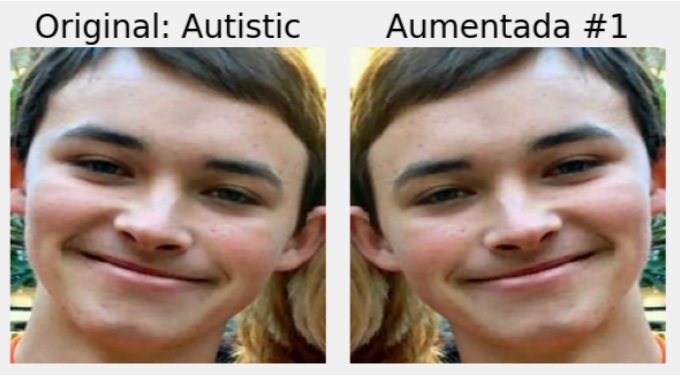


Figura 4.7. Exemplo de aumento de dados aplicado a uma imagem do *dataset*. À esquerda, a imagem original; à direita, a versão aumentada por espelhamento horizontal.

Testes preliminares com outras transformações como rotação, *zoom* e brilho revelaram que essas modificações tendiam a degradar o desempenho do modelo, provavelmente porque alteravam características faciais relevantes para a detecção de TEA. Este resultado reforça a importância de considerar a natureza específica do problema ao escolher estratégias de aumento de dados.

A técnica de aumento de dados não apenas aumenta a quantidade de dados dis-

poníveis, mas também melhora a capacidade de generalização do modelo ao expô-lo a variações naturais que podem ocorrer em aplicações reais. No contexto da detecção de TEA, onde eventualmente o modelo pode ser aplicado a imagens capturadas em condições variadas através de dispositivos móveis, essa robustez adicional é particularmente valiosa.

4.5.4. Implementação

A implementação prática do modelo para detecção de TEA utilizou técnicas de transferência de aprendizado com múltiplas arquiteturas CNN pré-treinadas. Foram avaliadas cinco arquiteturas: *MobileNet*, *MobileNetV2*, *DenseNet201*, *VGG19* e *ResNet50V2*, todas inicializadas com pesos pré-treinados na *ImageNet* e adaptadas para o problema específico de classificação binária de TEA, como mostra o código 4.4.

```
from tensorflow.keras.applications import DenseNet201, MobileNet
   , VGG19, ResNet50V2
from tensorflow.keras.layers import Dense,
   GlobalAveragePooling2D, Dropout
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam

def create_model(base_architecture='DenseNet201', input_shape
   =(224, 224, 3)):
    # Selecao da arquitetura base
    if base_architecture == 'DenseNet201':
        base_model = DenseNet201(weights='imagenet', include_top
           =False, input_shape=input_shape)
    elif base_architecture == 'MobileNet':
        base_model = MobileNet(weights='imagenet', include_top=
           False, input_shape=input_shape)
    elif base_architecture == 'VGG19':
        base_model = VGG19(weights='imagenet', include_top=False
           , input_shape=input_shape)
    elif base_architecture == 'ResNet50V2':
        base_model = ResNet50V2(weights='imagenet', include_top=
           False, input_shape=input_shape)

    # Congelar camadas do modelo base
    for layer in base_model.layers:
        layer.trainable = False

    # Adicionar camadas de classificacao personalizadas
    x = base_model.output
    x = GlobalAveragePooling2D()(x)
    x = Dense(1024, activation='relu')(x)
    x = Dropout(0.4)(x)
    predictions = Dense(1, activation='sigmoid')(x)  #
       Classificacao binaria
```

```
    # Modelo final
    model = Model(inputs=base_model.input, outputs=predictions)

    # Compilacao para classificacao binaria
    model.compile(
        optimizer=Adam(learning_rate=0.001),
        loss='binary_crossentropy',
        metrics=['accuracy', 'precision', 'recall']
    )

    return model, base_model
```

Listing 4.4. Implementação do modelo para detecção de TEA usando transferência de aprendizado com arquiteturas pré-treinadas na ImageNet, adaptadas para classificação binária.

Para classificação binária, utilizamos uma única saída com ativação *sigmoid*, que mapeia os valores para o intervalo [0,1] representando a probabilidade de pertencer à classe TEA. Esta abordagem é mais direta e eficiente para problemas de duas classes, utilizando a função de perda *binary_crossentropy* que é otimizada especificamente para este cenário.

No código 4.5, observa-se que o carregamento dos dados foi adaptado para o formato binário, onde as classes são representadas como 0 (desenvolvimento típico) e 1 (TEA):

```
from tensorflow.keras.preprocessing.image import
   ImageDataGenerator

# Configuracao dos geradores para classificacao binaria
train_datagen = ImageDataGenerator(
    rescale=1./255,
    horizontal_flip=True,
    validation_split=0.2
)

test_datagen = ImageDataGenerator(rescale=1./255)

# Carregamento dos dados com class_mode='binary'
train_generator = train_datagen.flow_from_directory(
    'autism_dataset/train',
    target_size=(224, 224),
    batch_size=32,
    class_mode='binary',  # Modo binario
    subset='training'
)

validation_generator = train_datagen.flow_from_directory(
    'autism_dataset/train',
    target_size=(224, 224),
    batch_size=32,
```

```
    class_mode='binary',
    subset='validation'
)

test_generator = test_datagen.flow_from_directory(
    'autism_dataset/test',
    target_size=(224, 224),
    batch_size=32,
    class_mode='binary',
    shuffle=False
)
```

Listing 4.5. Preparação dos geradores de dados para treino, validação e teste com classificação binária

O treinamento foi conduzido em duas fases seguindo as melhores práticas de transferência de aprendizado como mostra o código 4.6. Na primeira fase, apenas as camadas de classificação adicionadas foram treinadas por 20 épocas, mantendo todas as camadas do modelo base congeladas.

```
# Primeira fase treinamento das camadas superiores
model, base_model = create_model('DenseNet201')

history_phase1 = model.fit(
    train_generator,
    epochs=20,
    validation_data=validation_generator,
    callbacks=[
        tf.keras.callbacks.EarlyStopping(patience=5,
           restore_best_weights=True),
        tf.keras.callbacks.ReduceLROnPlateau(patience=3, factor
           =0.5)
    ]
)

# Segunda fase fine tuning opcional
# Descongelar algumas camadas superiores do modelo base
for layer in base_model.layers[-10:]:
    layer.trainable = True

# Recompilar com taxa de aprendizado menor
model.compile(
    optimizer=Adam(learning_rate=0.0001),  # Taxa 10x menor
    loss='binary_crossentropy',
    metrics=['accuracy', 'precision', 'recall']
)

history_phase2 = model.fit(
    train_generator,
    epochs=10,
```

```
    validation_data=validation_generator,
    callbacks=[
        tf.keras.callbacks.EarlyStopping(patience=3,
           restore_best_weights=True)
    ]
)
```

Listing 4.6. Treinamento e Fine tuning com DenseNet201

Durante os experimentos, foram implementadas técnicas de regularização e monitoramento para prevenir *overfitting*. O *Early Stopping* interrompe o treinamento quando a perda de validação para de melhorar, enquanto o `ReduceLROnPlateau` ajusta automaticamente a taxa de aprendizado quando o modelo atinge um platô de desempenho.

Os resultados experimentais demonstraram que as arquiteturas *DenseNet201* e *MobileNet* obtiveram os melhores desempenhos. A *DenseNet201* alcançou acurácia média de 90,7% com desvio padrão de 1,64%, chegando a 93,5% nos melhores casos. A *MobileNet*, embora com acurácia máxima ligeiramente inferior (92,5%), apresentou maior estabilidade com desvio padrão de apenas 0,68%, indicando comportamento mais consistente entre diferentes execuções.

Para avaliar o desempenho final do modelo e analisar detalhadamente seus resultados, implementamos uma rotina de avaliação abrangente que calcula métricas de desempenho, gera a matriz de confusão e analisa a distribuição das probabilidades preditas. O código 4.7 demonstra essa análise.

```
# Avaliacao do modelo no conjunto de teste
test_results = model.evaluate(test_generator)
print(f"Acuracia no teste {test_results[1]:.3f}")
print(f"Precisao no teste {test_results[2]:.3f}")
print(f"Recall no teste {test_results[3]:.3f}")

# Predicoes para analise detalhada
predictions = model.predict(test_generator)
# Para classificacao binaria aplicamos threshold de 05
predicted_classes = (predictions > 0.5).astype(int).flatten()
true_classes = test_generator.classes

# Matriz de confusao
from sklearn.metrics import confusion_matrix,
   classification_report
cm = confusion_matrix(true_classes, predicted_classes)
print("Matriz de Confusao")
print(cm)
print("\nRelatorio de Classificacao")
print(classification_report(true_classes, predicted_classes,
                            target_names=['Desenvolvimento Tipico'
                               , 'TEA']))

# Analise das probabilidades preditas
```

```
print(f"\nEstatisticas das predicoes")
print(f"Probabilidade media para classe TEA {predictions.mean()
    :.3f}")
print(f"Desvio padrao {predictions.std():.3f}")
print(f"Predicoes com alta confianca maior que 08 ou menor que
    02 {((predictions > 0.8) | (predictions < 0.2)).sum()}/{len(
    predictions)}")
```

Listing 4.7. Avaliação do modelo no conjunto de teste

4.5.5. Explicabilidade com GradCAM

A interpretabilidade de modelos de *deep learning* é crucial em aplicações médicas, onde a confiança e compreensão das decisões do modelo são tão importantes quanto sua precisão. Para este fim, implementamos o *Gradient-weighted Class Activation Mapping* (GradCAM), uma técnica que permite visualizar quais regiões da imagem são mais importantes para as decisões do modelo [Selvaraju et al. 2017].

O GradCAM funciona calculando os gradientes da classe prevista em relação aos mapas de características de uma camada convolucional específica. Estes gradientes são então utilizados como pesos para combinar os mapas de características, produzindo um mapa de calor que destaca as regiões mais influentes para a classificação. O código 4.8 apresenta uma implementação simplificada dessa técnica, incluindo a geração e visualização do mapa de calor sobre a imagem de entrada.

```
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras.models import Model
import cv2

def grad_cam(model, img_array, layer_name, class_idx):
    grad_model = Model([model.inputs], [model.get_layer(
        layer_name).output, model.output])

    with tf.GradientTape() as tape:
        conv_outputs, predictions = grad_model(img_array)
        loss = predictions[:, class_idx]

    grads = tape.gradient(loss, conv_outputs)
    pooled_grads = tf.reduce_mean(grads, axis=(0, 1, 2))
    heatmap = tf.reduce_sum(tf.multiply(pooled_grads,
        conv_outputs[0]), axis=-1)
    heatmap = tf.maximum(heatmap, 0) / tf.math.reduce_max(
        heatmap)

    return heatmap.numpy()

def visualize_gradcam(model, image, true_label, predicted_label,
    class_names):
```

```
    for layer in reversed(model.layers):
        if 'conv' in layer.name.lower():
            last_conv_layer = layer.name
            break
    else:
        base_model = model.layers[0]
        for layer in reversed(base_model.layers):
            if 'conv' in layer.name.lower():
                last_conv_layer = layer.name
                break

    img_array = np.expand_dims(image, axis=0)
    heatmap = grad_cam(model, img_array, last_conv_layer,
       predicted_label)

    heatmap_resized = cv2.resize(heatmap, (image.shape[1], image
       .shape[0]))
    heatmap_colored = cv2.applyColorMap(np.uint8(255 *
       heatmap_resized), cv2.COLORMAP_JET)
    superimposed = cv2.addWeighted(np.uint8(255 * image), 0.6,
       heatmap_colored, 0.4, 0)

    fig, axes = plt.subplots(1, 3, figsize=(15, 5))
    axes[0].imshow(image)
    axes[0].set_title(f'Original\nVerdadeiro {class_names[
       true_label]}')
    axes[1].imshow(heatmap_resized, cmap='jet')
    axes[1].set_title(f'Mapa de Calor\nPredito {class_names[
       predicted_label]}')
    axes[2].imshow(superimposed)
    axes[2].set_title('Sobreposicao')

    for ax in axes:
        ax.axis('off')
    plt.tight_layout()
    plt.show()
```

Listing 4.8. Implementação simplificada do GradCAM

A técnica GradCAM fornece uma visualização das regiões da imagem que influenciam as decisões do modelo, mas seus resultados devem ser interpretados com cautela. As áreas destacadas não necessariamente correspondem a biomarcadores clínicos reais do TEA. Essa interpretabilidade permite verificar possíveis vieses, como atenção a elementos irrelevantes, e avaliar se o modelo está focando em regiões anatomicamente plausíveis, além de facilitar a comunicação com profissionais da saúde ao tornar o processo de decisão mais transparente.

Neste estudo, o GradCAM foi utilizado para explorar o comportamento de uma CNN na detecção do TEA por meio de imagens faciais, demonstrando desempenho pro-

missor. No entanto, destaca-se que esses modelos devem atuar como apoio ao julgamento clínico, e não como substitutos. A implementação prática ainda requer atenção a fatores como viés nos dados, capacidade de generalização e validação clínica rigorosa.

Referências

[Alves Rodrigues et al. 2023] Alves Rodrigues, J. N., Teixeira Aires, K. R., Branco Soares, A. C., Machado, V. P., and de Melo Souza Veras, R. (2023). Classification of facial images using deep learning models to support asd identification. In *2023 XLIX Latin American Computer Conference (CLEI)*, pages 1–9. IEEE.

[American Psychiatric Association 2013] American Psychiatric Association (2013). *Diagnostic and statistical manual of mental disorders (DSM-5)*. American Psychiatric Pub.

[Bengio 2012] Bengio, Y. (2012). Practical recommendations for gradient-based training of deep architectures. In *Neural networks: Tricks of the trade*, pages 437–478. Springer.

[Bergstra and Bengio 2012] Bergstra, J. and Bengio, Y. (2012). Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(2):281–305.

[Boureau et al. 2010] Boureau, Y.-L., Ponce, J., and LeCun, Y. (2010). A theoretical analysis of feature pooling in visual recognition. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 111–118.

[Centers for Disease Control and Prevention 2023] Centers for Disease Control and Prevention (2023). Data and statistics on autism spectrum disorder. Accessed: 2025-05-25.

[Claro et al. 2020] Claro, M. L., Vogado, L. H. S., Santos, J. D., and Veras, R. M. S. (2020). Utilização de técnicas de data augmentation em imagens: Teoria e prática. In Teles, A. S., Calçada, D. B., and Veras, N. L., editors, *Livro de Minicursos - VIII Escola Regional de Computação Ceará, Maranhão e Piauí (ERCEMAPI 2020)*, chapter 3, pages 47–71. Sociedade Brasileira de Computação, Porto Alegre.

[Fukushima 1988] Fukushima, K. (1988). Neocognitron: A hierarchical neural network capable of visual pattern recognition. *Neural Networks*, 1(2):119–130.

[Glorot et al. 2011] Glorot, X., Bordes, A., and Bengio, Y. (2011). Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 315–323. JMLR Workshop and Conference Proceedings.

[Gonzalez and Woods 2002] Gonzalez, R. C. and Woods, R. E. (2002). *Digital Image Processing*. Prentice Hall.

[Goodfellow et al. 2016] Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press.

[Hachilif et al. 2019] Hachilif, R., Baghdadi, R., and Benhamida, F. (2019). *Graduation Thesis: Implementing and Optimizing Neural Networks using Tiramisu*. Phd thesis, Unknown Institution.

[Haykin 2007] Haykin, S. (2007). *Redes neurais: princípios e prática*. Bookman Editora.

[He et al. 2016] He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, pages 770–778.

[Janocha and Czarnecki 2017] Janocha, K. and Czarnecki, W. M. (2017). On loss functions for deep neural networks in classification. *arXiv preprint arXiv:1702.05659*.

[Keskar et al. 2017] Keskar, N. S., Nocedal, J., Tang, D., Mudigere, D., and Smelyanskiy, M. (2017). On large-batch training for deep learning: Generalization gap and sharp minima. *arXiv preprint arXiv:1609.04836*.

[LeCun et al. 1998] LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

[Lin et al. 2017] Lin, T.-Y., Goyal, P., Girshick, R., He, K., and Dollár, P. (2017). Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision (ICCV)*, pages 2980–2988.

[Lorena et al. 2021] Lorena, A., Faceli, K., Almeida, T., de Carvalho, A., and Gama, J. (2021). *Inteligência Artificial: uma abordagem de Aprendizado de Máquina*. LTC, 2 edition.

[Marques Filho and Neto 1999] Marques Filho, O. and Neto, H. V. (1999). *Processamento digital de imagens*. Brasport.

[Mumuni and Mumuni 2022] Mumuni, A. and Mumuni, F. (2022). Data augmentation: A comprehensive survey of modern approaches. *Array*, 16:100258.

[Nair and Hinton 2010] Nair, V. and Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814.

[Pan and Yang 2010] Pan, S. J. and Yang, Q. (2010). A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359.

[Pereira et al. 2024] Pereira, C. B. A. A., Barros, P., Rodrigues, J. N., Araújo, P., Borges, R., Almeida, K., and Veras, R. (2024). Identificação de parkinson em imagens faciais usando modelos de deep learning pré-treinados. In *Anais da XII Escola Regional de Computação do Ceará, Maranhão e Piauí*, pages 169–178. SBC.

[Rezende 2003] Rezende, S. O. (2003). *Sistemas inteligentes: fundamentos e aplicações*. Editora Manole Ltda.

[Ruder 2016] Ruder, S. (2016). An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*.

[Russakovsky et al. 2015] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., et al. (2015). Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252.

[Scherer et al. 2010] Scherer, D., Müller, A., and Behnke, S. (2010). Evaluation of pooling operations in convolutional architectures for object recognition. In *International Conference on Artificial Neural Networks*, volume 6354 of *Lecture Notes in Computer Science*, pages 92–101. Springer.

[Selvaraju et al. 2017] Selvaraju, R. R., Cogswell, M., Das, A., Vedantam, R., Parikh, D., and Batra, D. (2017). Grad-cam: Visual explanations from deep networks via gradient-based localization. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 618–626. IEEE.

[Smith 2017] Smith, L. N. (2017). Cyclical learning rates for training neural networks. *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 464–472.

[Srivastava et al. 2014] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958.

[Szegedy et al. 2015] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2015). Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, pages 1–9.

[Tajbakhsh et al. 2016] Tajbakhsh, N., Shin, J. Y., Gurudu, S. R., Hurst, R. T., Kendall, C. B., Gotway, M. B., and Liang, J. (2016). Convolutional neural networks for medical image analysis: Full training or fine tuning? *IEEE Transactions on Medical Imaging*, 35(5):1299–1312.

[Vogado et al. 2019] Vogado, L., Claro, M., Santos, J., and Veras, R. (2019). Deep learning em imagens: aplicando cnns com keras e tensorflow. In *Minicursos ERCAS-PI e ENUCOMPI 2019*, chapter 6, pages 65–96. Sociedade Brasileira de Computação, Porto Alegre.

[Wilson et al. 2017] Wilson, A. C., Roelofs, R., Stern, M., Srebro, N., and Recht, B. (2017). The marginal value of adaptive gradient methods in machine learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 4151–4161.

[Yosinski et al. 2014] Yosinski, J., Clune, J., Bengio, Y., and Lipson, H. (2014). How transferable are features in deep neural networks? In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 27.

Chapter 5

Automação de Tarefas com *APIs*: Integração entre *Google Calendar* e *WhatsApp* para Otimização de Processos

Francisnilto dos Santos Nascimento, Luis Guilherme Sampaio Fontenele, Ricardo Moura Sekeff Budaruiche e Vanessa Pereira Cunha

Abstract

This chapter discusses the development of a task automation solution by integrating the Google Calendar API with the WPPConnect Server, aiming to automate the delivery of personalized notifications via WhatsApp. The proposal employs an architecture based on RESTful APIs, secure authentication through OAuth 2.0, and modular configuration using Python. The configuration of APIs, daily retrieval of events and tasks, and automated message delivery processes are described. The approach highlights security best practices and the challenges related to using unofficial APIs. The proposed solution contributes to optimizing schedule management and disseminating automation processes in personal and corporate environments.

Resumo

Este capítulo aborda o desenvolvimento de uma solução para automação de tarefas, integrando a API (Application Programming Interface) do Google Calendar ao WPPConnect Server, com o objetivo de automatizar o envio de notificações personalizadas via WhatsApp. A proposta utiliza uma arquitetura baseada em APIs RESTful, com autenticação segura via OAuth 2.0 e configuração modular em Python. São descritas as etapas de configuração das APIs, recuperação de eventos e tarefas diárias e envio automatizado de mensagens. A abordagem destaca boas práticas de segurança e os desafios associados ao uso de APIs não oficiais. A solução proposta contribui para a otimização da gestão de compromissos e para a disseminação de processos de automação em ambientes pessoais e corporativos.

5.1. Introdução

Na era da informação e da hiperconectividade, a automação de tarefas consolidou-se como uma estratégia indispensável tanto no cotidiano quanto no ambiente corporativo,

permitindo otimizar rotinas, reduzir o esforço manual e aumentar significativamente a produtividade. A adoção de arquiteturas baseadas em serviços e interfaces bem definidas facilita a integração eficiente de diferentes sistemas. Essa tendência é impulsionada pela crescente utilização de APIs, que viabilizam a comunicação entre softwares de forma prática e escalável (SOMMERVILLE, 2019).

O conceito de *APIs* refere-se a um conjunto de regras e especificações que permitem que *softwares* distintos se comuniquem entre si. A adoção de *APIs* tem transformado a maneira como aplicações são desenvolvidas e integradas, especialmente com a difusão da arquitetura *REST* (*Representational State Transfer*), proposta por Roy Fielding em sua tese de doutorado em 2000, amplamente reconhecida como um marco na evolução dos sistemas distribuídos e da *Web* moderna. A *API* do *Google Calendar*, por exemplo, é uma *API RESTful*, o que significa que segue essa arquitetura, baseada em diretrizes que visam uma comunicação simples, escalável e eficiente entre sistemas distribuídos (GOOGLE DEVELOPERS, 2025).

A arquitetura *REST* é um estilo de arquitetura para sistemas distribuídos, projetado para garantir a comunicação eficiente entre sistemas. Ela utiliza o protocolo **HTTP** (*Hypertext Transfer Protocol*) como meio de comunicação, com operações padrão como consulta (*GET*), envio de dados (*POST*), atualização (*PUT*) e exclusão (*DELETE*) (FIELDING, 2000). A principal característica do *REST* é sua simplicidade e escalabilidade, permitindo que os sistemas sejam facilmente integrados e interoperáveis.

Segundo estudos (FIELDING, 2000), um sistema que segue a arquitetura *REST* deve ser *stateless* (sem estado), o que significa que cada requisição deve ser independente e conter todas as informações necessárias para ser processada, sem depender de sessões anteriores. Além disso, *REST* facilita a troca de dados entre sistemas distribuídos por meio de representações padronizadas, como *JSON* (*JavaScript Object Notation*) ou *XML* (*Extensible Markup Language*), garantindo uma comunicação eficiente e escalável.

Automatizar significa utilizar tecnologias capazes de executar ações sem a necessidade de intervenção humana contínua, otimizando processos e eliminando atividades repetitivas (BARRETO et al., 2024). Um exemplo prático é a integração entre o *Google Calendar* e o *WhatsApp*: imagine que, ao iniciar o dia, um usuário receba automaticamente, via WhatsApp, uma mensagem contendo seus compromissos agendados, proporcionando organização pessoal e ganhos de produtividade.

Ferramentas como o *Google Calendar* são fundamentais neste cenário, pois oferecem *APIs* bem documentadas e estáveis, facilitando integrações com outros serviços, como *Gmail*, *Google Meet* e *Google Tasks* (MELO MESSIAS et al., 2022). Da mesma forma, o *WhatsApp* também disponibiliza uma *API oficial*, a *WhatsApp Business API*, que permite a automação de interações e integrações com sistemas externos. Embora a *API* oficial do *WhatsApp* seja paga e voltada para empresas, ela oferece uma solução robusta e escalável para comunicação via *WhatsApp*.

No entanto, alternativas como o *WPPConnect*, uma *API* não oficial, também são amplamente utilizadas em projetos que buscam uma solução mais acessível e simples de implementar, embora com algumas limitações e riscos em termos de estabilidade e conformidade (WPPCONNECT, 2025). A arquitetura *REST*, utilizada tanto por essas *APIs* quanto por outras, é caracterizada pela escalabilidade, simplicidade na troca de informações (FIELDING, 2000).

Neste capítulo, exploraremos como a automação de tarefas pode ser realizada através da integração entre o *Google Calendar* e o *WhatsApp*, utilizando a arquitetura *REST* e *APIs* específicas para otimizar a gestão de compromissos pessoais e profissionais.

5.1.1. Conceitos e Vantagens da Automação de Tarefas.

Na atualidade, gerenciar compromissos e tarefas seja no ambiente empresarial ou na vida pessoal pode ser desafiador e repetitivo. Em um mundo dinâmico, onde a eficiência na organização é essencial, a automação de tarefas surge como uma solução indispensável, pois é capaz de otimizar processos e liberar tempo para atividades estratégicas (SANTOS, 2023).

A automação de tarefas proporciona a **redução de erros manuais** ao seguir padrões definidos, garantindo a **padronização e a execução** uniforme dos processos. Além disso, promove **economia de tempo** ao liberar o usuário de atividades repetitivas, permitindo maior foco em questões estratégicas. Como consequência, há um **aumento da produtividade**, com a diminuição do tempo gasto em tarefas operacionais e a melhoria na agilidade dos processos. A automação também representa um passo importante rumo à **transformação digital**, favorecendo a eficiência operacional e a inovação nos setores público e privado (BRANDAO, 2021).

5.1.2. Exemplos Práticos de Automação no Cotidiano Pessoal e Corporativo.

A automação está cada vez mais presente em diferentes aspectos da vida moderna, muitas vezes de forma quase imperceptível, simplificando tarefas rotineiras e aprimorando a experiência do usuário. No cotidiano, é possível observar a aplicação de soluções automatizadas como o envio automático de mensagens de aniversário ou lembretes via *WhatsApp*, a sincronização de agendas com alertas em tempo real e a geração automática de relatórios a partir de planilhas *online*. Esses recursos tornam-se aliados importantes na organização pessoal e na economia de tempo.

No ambiente corporativo, a automação também exerce um papel fundamental ao otimizar processos e melhorar a comunicação entre equipes. Exemplos comuns incluem o envio de alertas de reuniões com base em calendários compartilhados, o processamento automatizado de pagamentos e a emissão de faturas, além da integração entre sistemas de *CRM* (*Customer Relationship Management*) e plataformas de atendimento ao cliente. Tais aplicações, frequentemente viabilizadas por meio de *APIs*, demonstram como soluções tecnológicas simples e bem projetadas podem promover ganhos expressivos em produtividade, gestão de tempo e eficiência operacional.

5.1.3. Tipos de *APIs* para Automação.

A base da automação moderna está na integração entre sistemas, viabilizada principalmente pelo uso de *APIs*. Cada tipo de *API* possui uma finalidade específica, e compreendê-las é essencial para a escolha das soluções mais adequadas a diferentes demandas. **APIs de integração**, por exemplo, permitem a troca de dados entre plataformas distintas, como ocorre com serviços como *Google Calendar*, *Trello* e *GitHub*. Já as **APIs de comunicação** possibilitam o envio automatizado de mensagens, alertas e notificações, sendo representadas por ferramentas como *Twilio*, *Telegram* e *WhatsApp* (via *WPPConnect*).

Outro grupo importante são as *APIs* voltadas a assistentes virtuais, que utilizam inteligência artificial para interpretar comandos e executar tarefas, como *Alexa Skills* e

Google Assistant. O domínio dessas interfaces confere maior autonomia no desenvolvimento de soluções automatizadas sob medida, com potencial para promover ganhos significativos em agilidade, organização e eficiência. Seja em contextos pessoais ou profissionais, o uso estratégico de *APIs* contribui diretamente para a transformação digital e a otimização de processos.

5.2. *APIs Google Calendar* e *WPPConnect Server* - Automatizando Eventos e Comunicações.

A crescente demanda por soluções tecnológicas que otimizem a comunicação e a organização de atividades impulsiona o desenvolvimento de aplicações integradas a serviços em nuvem por meio de *APIs*. Em contextos empresariais, educacionais e sociais, a automação de tarefas e o gerenciamento eficiente de eventos tornam-se diferenciais estratégicos. Nesse cenário, destacam-se duas ferramentas amplamente utilizadas: a *API* do *Google Calendar* e o *WPPConnect Server*.

A *API* do *Google Calendar* permite o gerenciamento programático de agendas, oferecendo funcionalidades como criação, edição e exclusão de eventos, além de notificações personalizadas. Por meio do protocolo *OAuth* 2.0, garante-se o acesso seguro e autorizado aos dados do usuário, respeitando os princípios de privacidade e proteção da informação. Já o *WPPConnect Server* surge como uma alternativa acessível à *API* oficial do *WhatsApp*, permitindo a automação de mensagens e interações via *WhatsApp Web*, com flexibilidade e controle sobre sessões autenticadas.

A integração dessas duas tecnologias possibilita a criação de soluções robustas que não apenas organizam compromissos e atividades, mas também notificam automaticamente os usuários por meio de canais amplamente utilizados, como o *WhatsApp*. Este capítulo apresenta os conceitos, configurações e práticas de uso dessas *APIs*, visando apoiar o desenvolvimento de aplicações funcionais, escaláveis e centradas na experiência do usuário.

A Figura 5.1 ilustra a infraestrutura da solução desenvolvida, evidenciando a interação entre os principais componentes do sistema: as *APIs* do *Google Calendar* e *Google Tasks*, o servidor *WPPConnect* e o *script* de automação. O fluxo apresenta a sequência de operações desde a configuração inicial, passando pela consulta aos serviços de agenda, até o envio automatizado das mensagens via *WhatsApp*. Este diagrama visa proporcionar uma visão geral da arquitetura proposta, facilitando a compreensão das etapas descritas nas seções subsequentes.

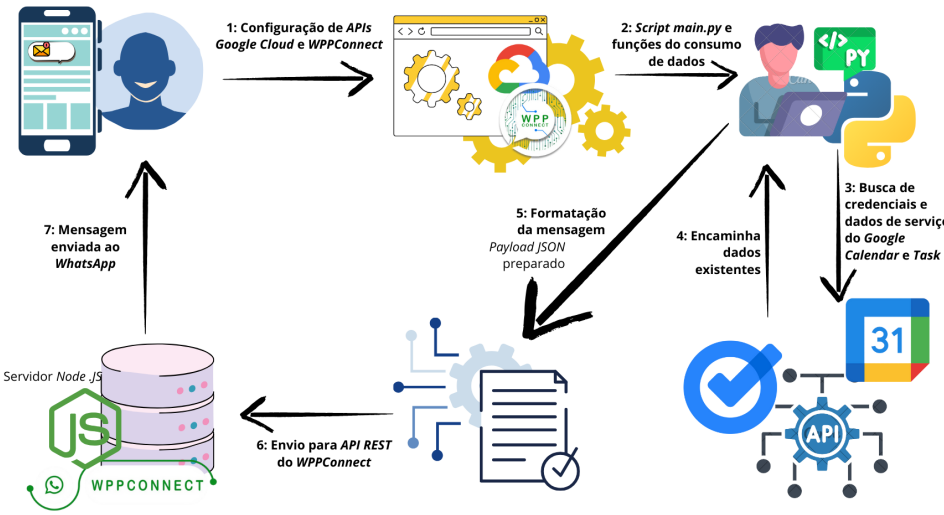


Figure 5.1: Arquitetura de integração para automatizar mensagens.

5.2.1. O Que é o *WPPConnect Server*?

O *WPPConnect Server* é uma solução de código aberto que viabiliza a integração com o *WhatsApp* por meio de uma interface *web*. Desenvolvido a partir da biblioteca *wppconnect*, esse servidor utiliza técnicas de **engenharia reversa** aplicadas ao protocolo *WebSocket* do *WhatsApp Web*, permitindo funcionalidades como envio, recebimento e gerenciamento de mensagens (WPPCONNECT, 2025).

Engenharia reversa é definida como o processo de análise de produtos ou protocolos existentes a partir de artefatos como binários, *memory dumps* ou tráfego de rede. Esse processo visa compreender a estrutura, lógica de operação e protocolos internos de sistemas, mesmo na ausência de documentação formal ou acesso ao código-fonte original (EILAM, 2011).

A utilização da engenharia reversa possibilita a construção de interfaces de comunicação com sistemas fechados, por meio de *APIs* não oficiais. Além disso, essa abordagem permite a recuperação de especificações de sistemas legados e a identificação de vulnerabilidades, contribuindo para ações de correção e auditoria.

No contexto do *WPPConnect Server*, observa-se o tráfego criptografado entre o cliente *web* do *WhatsApp* e seus servidores oficiais. Através da ferramenta *Puppeteer*, são reproduzidas as chamadas *WebSockets* e os *endpoints* utilizados pelo *WhatsApp Web*, os quais são expostos como uma *API REST*.

Contudo, por tratar-se de uma implementação não oficial da plataforma META, não há suporte oficial. Dessa forma, a cada modificação no *front-end* ou no protocolo do *WhatsApp*, torna-se necessário atualizar o processo de engenharia reversa para garantir a compatibilidade do sistema.

5.2.2. Criando um Projeto no *Google Cloud* para Uso da *API Google Calendar*.

Para utilizar a *API* do *Google Calendar* de maneira funcional e robusta, é necessário, primeiramente, criar um projeto na plataforma *Google Cloud Console*. Esse processo abrange desde a criação de um novo projeto até a geração de credenciais por meio de um arquivo *JSON* instalável.

Ao abrir a interface do *Google Cloud Console*, clique em "***Console***", localizado na parte superior da página. Posteriormente, será possível acessar o *Google Cloud Platform*,

que disponibiliza os meios de configurações e serviços de várias *APIs*, principalmente a *API* do *Google Calendar*. Acessando o *Google Cloud Platform* e depois logando uma conta google de usuário, crie um novo projeto ao clicar no botão "***My First Project***", logo, abrirá a janela de **seleção** ou **criação** de projeto.

Dessa forma, vamos criar um **novo projeto**, defina um nome de projeto, e, se necessário, selecione uma organização ou mantenha a opção padrão. Logo após a criação de seu projeto, certifique-se de que ele esteja selecionado no mesmo local que se encontrava a informação "***My First Project***", agora substituído pelo nome do seu projeto.

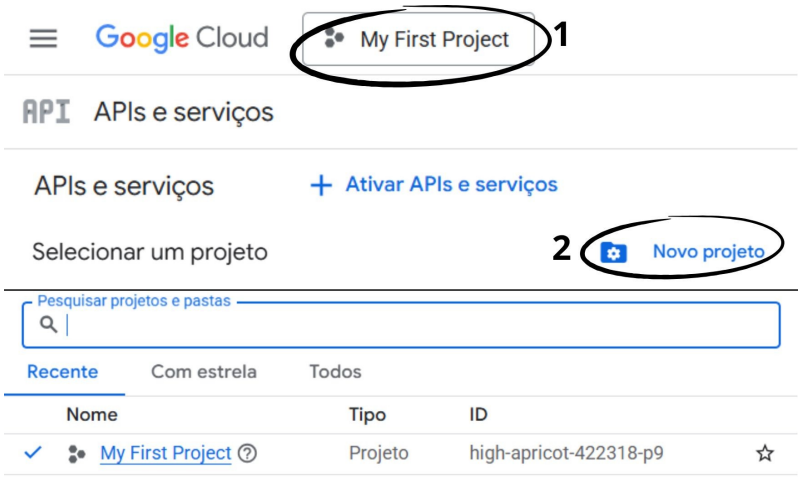


Figure 5.2: Criando um projeto no *Google Cloud*

Com o projeto selecionado, o próximo passo é acessar a seção responsável pela criação do **ID do Cliente *OAuth***. Clique nas **três barras** ao lado esquerdo superior da tela, procure por "***APIs* e Serviços**" e selecione por "**credenciais**".

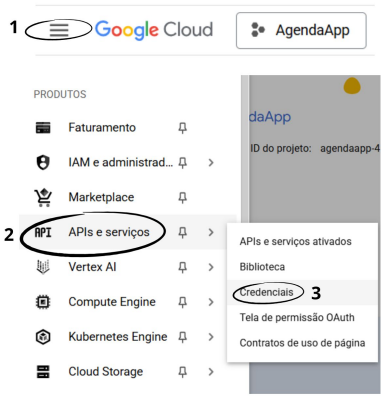


Figure 5.3: Passo a passo para acesso a tela de credenciais.

ID do Cliente é utilizado quando sua aplicação precisa acessar a *API* em nome de um usuário, permitindo a autenticação e autorização por meio da conta pessoal desse usuário. Esse tipo de credencial é essencial para aplicações que interagem diretamente com usuários e precisam de permissões específicas para acessar dados protegidos. Na parte superior da página de credenciais, clique em "**+ Criar credenciais**" e, em seguida, selecione a opção "**ID do Cliente *OAuth***".

Ao cadastrar uma conta ao *Google Cloud* pela primeira vez, a página de criação do ID do Cliente para a autenticação, antes de tudo, fará com que o usuário configure a **tela de consetimento**. Ao ser redirecionado para a página de configuração sobre permissões autenticadas, será necessário inserir dados como: **informações do *app*** - insira nome e *email* para suporte de usuário; **público** - escolha para público externo e **dados de contato** - recomenda-se inserir o mesmo *email* inserido em informações do *app*.

Figure 5.4: Passo a passo para acesso a tela de credenciais.

Após configurar a permissão de consetimento do usuário, a página será atualizada para a aba de "**visão geral de cliente *OAuth***". Logo, será possível criar e configurar um "**Cliente *OAuth***". Na mesma página de visão geral, em **métricas** clique em "**Criar um cliente *OAuth***".

Para criar um **Cliente *OAuth***, deve-se preencher os campos **tipo de aplicativo**, **nome do aplicativo** e ***URLs* de redirecionamento autorizados**. Para o "Tipo de aplicativo" deve ser **aplicação *web***, o "nome" fica ao critério coerente do usuário e para a "*URL* de redirecionamento autorizado" pode-se utilizar a chamada ***HTTP*** vinculada a um *localhost* da máquina, inicialmente apenas para testes de algoritmos trabalhados na sessão, por exemplo: `http://localhost:9090/`

Figure 5.5: Passo a passo para criar o Cliente *OAuth*.

Ressalta-se que, posteriomente, com toda a aplicação do sistema construída, a mesma passará para um deploy, subindo a aplicação para um servidor e com um domínio específico . Portanto, a *URL* de redirecionamento autorizado do *localhost* deve ser alterada para a nova *URL* do servidor, onde a aplicação estará rodando. Guarde esse arquivo de credenciais apra ser utilizado no código da aplicação.

Criando o cliente de autenticação do *Google*, será possível baixar o arquivo ***JSON*** com as respectivas credenciais do cliente. Veja ao anexo baixo a estrutura do arquivo *JSON*' gerado.

```
{"web":{
   "client_id": <"Seu ID de cliente">,
   "project_id":<"Seu ID de projeto">,
   "auth_uri":"https://accounts.google.com/auth", #Valor fictício
   "token_uri":"https://oauth.googleapis.com/token", #Valor fictício
   "auth_provider_x509_cert_url":"https://www.googleapis.com/oauth2/",
   "client_secret": <"Seu client_secret">,
   "redirect_uris":["http://localhost:9090/"]
   }
}
```

Listing 5.1: Dados de autenticação disponibilizados pelo *response body*

Ainda tratando-se de configurações para o Cliente *OAuth*, precisa-se realizar mais um passo: delimitar usuários testes que poderão usar as credenciais do ID de cliente criado para o projeto "AgendaAPP", pois o cliente não foi publicado. Logo, a autenticação vai ser bloqueada para as contas *emails* que não foram inseridas como conta teste para esse Cliente *OAuth* criado.

Para delimitar os usuários testes, procure por "***APIs* e Serviços**", selecione por "**Credenciais**", assim, é apresentada a janela que ficam registradas as "Chaves de *APIs*", "IDs do Cliente *OAuth* 2.0" e "Contas de Serviço". Em **IDs do Cliente *OAuth* 2.0**, escolha o Cliente ID criado e clique em editar - Ícone representado por um lápis na parte de ações.

Figure 5.6: Passo a passo para adicionar usuário testes.

Por fim, ative as *APIs* do *Google Calendar* e *Google Task*. Volte para a página inicial do *Google Cloud Console*, clique em "**Serviços e APIs**", logo em seguida, procure por **"Biblioteca"**. Ao acessar a interface de busca por serviços *Google*, insira na barra de pesquisa por ambas as *APIs*: "*Google Calendar*" e "**Google Task API**"

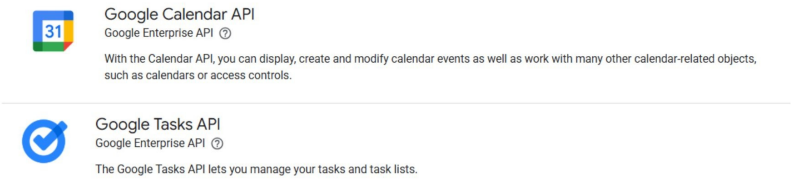


Figure 5.7: *APIs Google* necessárias para a automação de eventos e tarefas.

Como é possível ver na descrição da figura 5.7 para cada *API*, verifica-se que *Google Calendar API* é responsável por manipular os eventos agendados no *Google Calendar*; já *Google Tasks API* é um complemento funcional para o *Google Calendar*, que possibilita a manipulação de tarefas diárias agendadas para um determinado dia e horário.

5.2.3. Inicialização e Implantação Prática da *API*.

Antes de iniciar as implementações, o *WPPConnect Server* requer a instalação do Node.js, na versão 16.14 ou superior. Para verificar a versão instalada em sua máquina, utilize o comando `node -v` no *terminal*. Esse requisito se justifica pela necessidade de compatibilidade com funcionalidades modernas do *JavaScript/Node.js*, bem como com bibliotecas dependentes, além de proporcionar maior segurança, correção de *bugs*, e melhor desempenho no que se refere ao tempo de resposta de *APIs*, manipulação de grandes volumes de dados e processamento assíncrono.

Com o *Node.js* devidamente instalado, o próximo passo consiste em clonar o repositório oficial responsável pela estrutura da *API* e do servidor do *WPPConnect Server*. Após criar o diretório do projeto e acessá-lo por meio do *terminal*, *CMD* ou *PowerShell*, utilize o seguinte comando:

```
git clone https://github.com/wppconnect-team/wppconnect-server.git
```

Entre dentro da pasta *wppconnect-server* e instale todas as dependências da aplicação através do comando:

```
npm install
```

Atualmente, muitos projetos são desenvolvidos utilizando o ecossistema *JavaScript*, especialmente com o uso do *TypeScript*. O projeto *WPPConnect Server* adota essa abordagem. Nesse contexto, tanto a *API WPPConnect* quanto outros sistemas de *back-end* são construídos com recursos avançados do *Node.js*. Contudo, o *Node.js* não é capaz de interpretar código *TypeScript* de forma nativa, sendo necessário transpilar o código para *JavaScript*.

Para compilar a aplicação e inicializar o servidor, é necessário realizar o processo de *build* dentro do diretório *wppconnect-server*. Em seguida, o servidor pode ser executado por meio do comando de `start`, permitindo o acesso à interface do *WPPConnect Server* por meio do navegador.

npm run build

npm run start

Após a execução do comando `npm start`, torna-se possível acessar a interface *Swagger UI* da *API WPPConnect Server*. O ponto central para o desenvolvimento de uma automação básica de envio de mensagens via *WhatsApp* encontra-se nos *endpoints* do módulo de autenticação (*Auth*) da própria *API*.

A conexão entre o servidor e o *WhatsApp* é estabelecida por meio dos seguintes *endpoints* de autenticação:

```
POST/api/session/secretkey/generate-token
POST/api/session/start-session
```

```
GET/api/session/qrcode-session
```

Seguindo a ordem dos *endpoints* mencionados anteriormente, o primeiro a ser utilizado é o `POST/api/session/secretkey/generate-token`. Embora aspectos de segurança sejam discutidos mais detalhadamente em seção específica deste trabalho, é importante destacar que, para qualquer tentativa válida de conexão entre dados pessoais e o servidor, é necessário gerar um *token* de autenticação via ***JWT*** (*JSON Web Token*), utilizando uma *secretkey* - chave secreta vinculada à sessão a ser iniciada em um dispositivo.

Quando o servidor está configurado com autenticação baseada em *token*, esse *endpoint* deve ser invocado antes de qualquer outro, a fim de obter um *token* válido para autenticação nas requisições subsequentes.

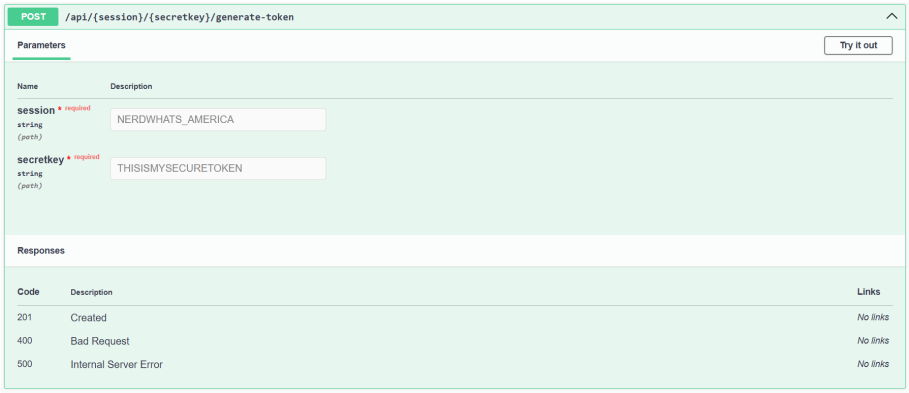


Figure 5.8: *Endpoint* para gerar *token* via *Swagger UI*.

É fundamental registrar corretamente o nome da sessão e a chave secreta, uma vez que essas informações serão utilizadas nas etapas posteriores de implementação do código. Esses dados permitem a comunicação direta com o servidor, possibilitando que requisições dos tipos *GET* e *POST* sejam devidamente autenticadas e executadas, assegurando o envio de mensagens e a execução de comandos por meio da *API*.

```
1 {
2    "status": "success",
3    "session": "NERDWHATS_AMERICA",
4    "token": <"Seu token">,
```

```
    "full": <"...">
}
```

Listing 5.2: Dados de autenticação disponibilizados pelo *response body*

Após habilitar os campos editáveis do `POST/api/{session}/{secretkey}/generate-token` por meio do botão ***Try it out***, e definir corretamente o nome da sessão e a chave secreta, ao clicar em ***execute***, o servidor retornará, no campo *response body*, o *token* de autenticação *JWT* correspondente à sessão que está prestes a ser iniciada.

Em seguida, ainda na interface do *Swagger*, deve-se clicar no botão ***Authorize*** para inserir o *token* gerado e autorizar a comunicação com o servidor por meio do *JWT*, garantindo o acesso autenticado às demais rotas da *API*. O procedimento detalhado pode ser visualizado na Figura 5.9.

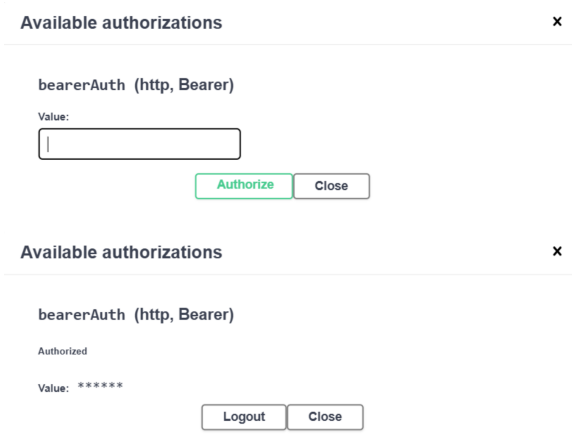


Figure 5.9: Janela de autorização do servidor via *JWT*.

Com o servidor devidamente autorizado por meio do *token JWT*, é possível iniciar uma sessão com maior confiabilidade e segurança em relação aos dados que serão instanciados ao longo da execução da aplicação.

Para iniciar uma sessão utilizando o *token* previamente autorizado, deve-se utilizar a requisição `POST/api/session/start-session`, disponível na interface do *Swagger*. Para esse processo, é necessário habilitar os campos editáveis por meio do botão ***Try it out***, assegurar que o nome da sessão coincida com o utilizado anteriormente, e, no campo ***request body***, atribuir o valor lógico `true` à propriedade ***`waitQrCode`***.

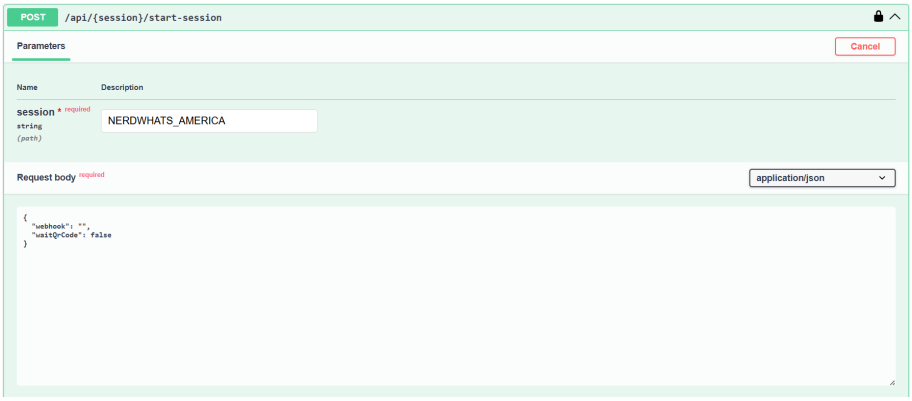


Figure 5.10: *Endpoint* para iniciar uma sessão.

Em seguida, ao executar a requisição, o próximo passo consiste na utilização do último *endpoint*, responsável por conectar o aplicativo *WhatsApp* ao servidor do *WPP-Connect Server*: `GET/api/session/qrcode-session`.

Não muito diferente dos demais *endpoints* trabalhados anteriormente, para este também é necessário clicar em ***Try it out***, definir o mesmo nome de sessão utilizado nos outros *endpoints* configurados previamente e, em seguida, acionar a opção ***execute***. Após essa execução, será gerado um ***QRCode***, o qual deve ser escaneado pelo aplicativo *WhatsApp* em um dispositivo móvel. Esse procedimento vincula a conta do usuário ao servidor do *WPPConnect*.

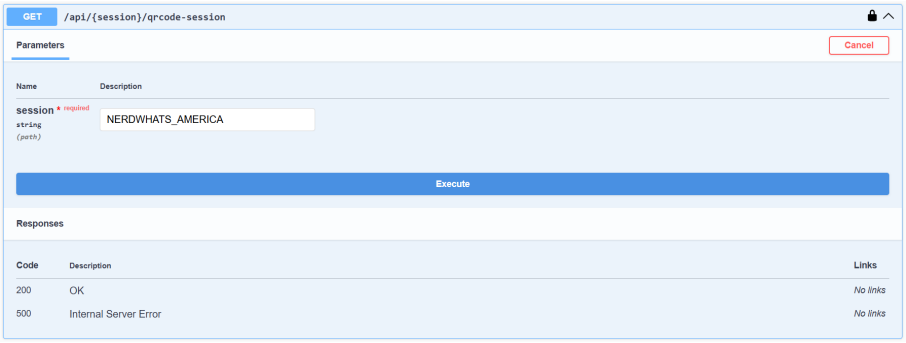


Figure 5.11: *Endpoint* para gerar o *QRCode* de vínculação.

5.3. Integração entre *Google Calendar* e *WhatsApp* para Recuperação de Eventos e Tarefas Diários.

5.3.1. Objetivo.

Neste tópico apresenta-se uma aplicação prática utilizando a *API* do *Google Calendar*, com o objetivo de recuperar, organizar e exibir os eventos agendados para o dia atual. Essa solução serve como base para sistemas de automação que visam facilitar a gestão de compromissos, notificações e produtividade pessoal ou corporativa.

5.3.2. Função de Carregar as Credenciais do *OAuth* 2.0

A aplicação é composta por um código em *Python* que utiliza bibliotecas amplamente conhecidas no ecossistema de desenvolvimento, disponibilizadas pelo arquivo ***requirements.txt***; faça o *download* e instalação dessas dependências de ambiente através do comando "`pip3 install -r requirements.txt`" dentro da pasta "*calendar-bot*".

Antes de tudo, é necessário importar bibliotecas essenciais para o funcionamento do processo. Utiliza-se módulos da biblioteca oficial do *Google*, como ***InstalledAppFlow***, que facilita o fluxo de autenticação para aplicações instaladas, e ***build***, para criar instâncias de serviços que acessam as *APIs*. A biblioteca ***pickle*** é utilizada para salvar e carregar o *token* de autenticação localmente, enquanto o pacote ***dotenv*** é responsável por carregar variáveis de ambiente definidas em um arquivo *.env*.

Listing 5.3: Importando módulos necessários para a função de carregar credenciais *Google*.

```
1 import os
2 import pickle
3 from google.auth.transport.requests import Request
```

```
from google_auth_oauthlib.flow import InstalledAppFlow
from googleapiclient.discovery import build
from dotenv import load_dotenv
load_dotenv()
]
```

Na sequência, é necessário definir os escopos de acesso, que determinam quais permissões a aplicação irá solicitar junto à conta *Google* do usuário. Os escopos funcionam como filtros de segurança, limitando rigorosamente quais recursos e tipos de dados a aplicação poderá acessar. Cada escopo está diretamente vinculado a uma funcionalidade específica, como acesso de leitura, configuração, edição ou fazer *download* de agenda.

Esses escopos são oficialmente documentados pelo *Google* e podem ser consultados no seguinte endereço https://developers.google.com/identity/protocols/oauth2/scopes?hl=pt-br. Nesse repositório oficial, encontra-se uma lista completa e constantemente atualizada dos escopos disponíveis para os diversos serviços da plataforma, como *Google Calendar*, *Tasks*, *Drive*, *Gmail*, entre outros.

Cada escopo possui uma descrição clara de sua função, permitindo que desenvolvedores escolham de forma precisa as permissões necessárias para suas aplicações. Isso garante não apenas o bom funcionamento do sistema, mas também reforça as práticas de segurança e privacidade, solicitando acesso apenas aos dados estritamente necessários para a finalidade proposta.

No contexto deste projeto, foram adotados escopos com permissões exclusivamente de leitura, tanto para o *Google Calendar* quanto para o *Google Tasks*. Essa escolha se justifica pelo fato de que a proposta da aplicação consiste apenas na consulta e exibição de informações, não havendo necessidade de criar, editar ou excluir eventos e tarefas.

Listing 5.4: Escopos necessários para permissionar a aplicação.

```
SCOPES = [
    "https://www.googleapis.com/auth/calendar.readonly",
    "https://www.googleapis.com/auth/tasks.readonly"
]
```

Em seguida, é implementada a função **carregarCredenciais()**, que tem como responsabilidade gerenciar todo o processo de autenticação e inicializar os serviços das *APIs*.

Listing 5.5: Função de carregar as credencias do *Google Calendar*.

```
def carregarCredenciais():
    creds = None
    token_path = 'token.pickle'
    client_secret_file = os.getenv("GOOGLE_CLIENT_SECRET", "
        credentials_oauth.json")
    ...
```

Listing 5.6: Verificação do *token*.

```
def carregarCredenciais():
    ...
    # Verifica se já existe um token salvo
```

```
    if os.path.exists(token_path):
        with open(token_path, 'rb') as token:
            creds = pickle.load(token)

    # Se não houver token válido, executa o fluxo de autenticação
    if not creds or not creds.valid:
        if creds and creds.expired and creds.refresh_token:
            creds.refresh(Request())
        else:
            flow = InstalledAppFlow.from_client_secrets_file(
                client_secret_file, SCOPES
            )
            creds = flow.run_local_server(port=9090,
               redirect_uri_trailing_slash=True)

        # Salva o token para reutilização futura
        with open(token_path, 'wb') as token:
            pickle.dump(creds, token)

    # Inicializa os serviços da API Calendar e Tasks
    service_calendar = build("calendar", "v3", credentials=creds)
    service_tasks = build("tasks", "v1", credentials=creds)
    return service_calendar, service_tasks
    ...
```

O processo descrito acima é essencial para qualquer integração com *APIs* do *Google*, garantindo que a aplicação acesse os dados de forma segura e autorizada. Além disso, o uso do arquivo **token.pickle** permite que o processo de autenticação não precise ser repetido toda vez que o *script* for executado, tornando-o mais eficiente.

5.3.3. Função Carregar Eventos.

O início do *script* traz as importações necessárias para o funcionamento da aplicação:

Listing 5.7: Importação de módulos necessários para carregar credenciais.

```
    from datetime import datetime, timedelta
    from dotenv import load_dotenv
    import os
    from utils import log
```

Cada uma dessas bibliotecas cumpre um papel específico. A ***datetime*** permite a manipulação de datas e horários, essencial para delimitar o intervalo de eventos a serem consultados. A biblioteca ***dotenv*** é usada para carregar variáveis de ambiente a partir de um arquivo externo *.env*, uma prática recomendada para manter dados sensíveis, como credenciais, fora do código-fonte. Já a biblioteca ***os*** permite acessar essas variáveis no ambiente de execução, enquanto o módulo ***log*** contém uma função auxiliar para registrar mensagens e erros.

Logo após as importações, o *script* carrega o ID do calendário utilizado na consulta à *API*. Esse ID é definido como uma variável de ambiente e carregado da seguinte forma:

Listing 5.8: Declarando variável de ID calendário.

```
    load_dotenv()
```

```
2     CALENDAR_ID = os.getenv("CALENDAR_ID")
```

A função principal do código chama-se carregarEventos e recebe como parâmetro o serviço autenticado da *API* do *Google Calendar*. Nela, é definido o intervalo de tempo correspondente ao dia atual, em formato *UTC* (*Coordinated Universal Time*), com base no horário do sistema. Isso garante que apenas os eventos do dia corrente sejam incluídos na busca:

Listing 5.9: Bloco de código que define intervalo de tempo ao dia atual.

```
1     def carregarEventos(service_calendar):
2     #Lista de mensagens
3     mensagens = []
4     try:
5         #Definindo intervalo de tempo com a biblioteca datetime
6         agora = datetime.utcnow()
7         inicio_do_dia = agora.replace(hour=0, minute=0, second=0,
              microsecond=0)
8         fim_do_dia = agora.replace(hour=23, minute=59, second=59,
              microsecond=999999)
9         ...
```

Com o intervalo definido, o próximo passo é fazer a requisição à *API*, informando o ***calendarId***, o intervalo de tempo e alguns parâmetros adicionais. A opção ***singleEvents=True*** garante que eventos recorrentes sejam tratados individualmente, e ***orderBy="startTime"*** organiza os resultados por horário:

Listing 5.10: Bloco de código

```
1     def carregarEventos(service_calendar):
2         ...
3         try:
4             ...
5             eventos_resultado = service_calendar.events().list(
6             calendarId=CALENDAR_ID,
7             timeMin=inicio_do_dia.isoformat() + 'Z',
8             timeMax=fim_do_dia.isoformat() + 'Z',
9             singleEvents=True,
10            orderBy="startTime"
11            ).execute()
12            ...
```

Os dados retornados pela *API* são armazenados em uma lista chamada eventos. Caso existam compromissos agendados, cada item é percorrido em um laço de repetição. A função extrai o horário de início e o título de cada evento e formata a mensagem em linguagem amigável, adequada para entrega ao usuário via interfaces como mensageiros ou assistentes virtuais.

Listing 5.11: Condicionais de existência para eventos

```
1     def carregarEventos(service_calendar):
2         ...
3         try:
4             ...
5             eventos = eventos_resultado.get('items', [])
```

```
            if eventos:
                mensagens.append("Eventos de hoje:")
                    for evento in eventos:
                        inicio = evento['start'].get('dateTime', evento
                            ['start'].get('date'))
                        titulo = evento.get('summary', 'Sem título')
                        mensagens.append(f"{inicio} - {titulo}")
            else:
                mensagens.append("Hoje você não tem eventos agendados."
                    )
            ...
```

Caso a agenda do dia esteja vazia, a função informa ao usuário com uma mensagem objetiva. Em situações de erro como falhas de autenticação, problemas de rede ou dados mal formatados o sistema captura a exceção, registra-a no terminal e retorna uma mensagem de falha controlada. Ao final da função, todas as mensagens são retornadas de forma unidas em um único texto, com quebras de linha:

Listing 5.12: Captura de exceção para controle de falhas

```
    def carregarEventos(service_calendar):
        ...
        try:
            ...
        except Exception as e:
            log(f"Erro ao carregar eventos: {e}")
            mensagens.append("Erro ao carregar eventos.")
        return "\n".join(mensagens)
```

5.3.4. Função Carregar Tarefas

A função apresentada a seguir tem como objetivo buscar e estruturar as tarefas presentes na conta do *Google Tasks*, organizando-as em uma mensagem formatada de forma clara e objetiva. Inicialmente, é realizada apenas a importação da função `log()`, responsável por exibir informações no *terminal*, especialmente em casos de erro durante a execução.

Listing 5.13: Importação da função log.

```
from utils import log
```

Em seguida, o código da função `carregarTarefas()` recebe como parâmetro o serviço da *API* do *Google Tasks* já autenticado. Dentro dela, inicia-se a criação de uma lista chamada `mensagens`, que armazenará os textos que compõem o retorno formatado da função.

Listing 5.14: Função de receber e criar a formatação das mensagens.

```
def carregarTarefas(service_tasks):
    mensagens = []
    tarefas_resultado = service_tasks.tasks().list(tasklist='@default')
        .execute()
    tarefas = tarefas_resultado.get('items', [])
```

A função também implementa um bloco de tratamento de exceções `try` e `except` para capturar e registrar possíveis erros na obtenção dos dados. Se ocorrer algum erro,

este é registrado no *terminal* através da função `log()` e uma mensagem de erro é retornada ao usuário.

Listing 5.15: Função de receber e criar a formatação das imagens.

```
def carregarTarefas(service_tasks):
    ...
    try:
        if tarefas:
            mensagens.append("\n*Tarefas de hoje:*")
            for tarefa in tarefas:
                titulo = tarefa.get('title', 'Sem título')
                mensagens.append(f"{titulo}")
        else:
            mensagens.append("Você não tem tarefas para hoje.")
    except Exception as e:
        log(f"Erro ao buscar tarefas: {e}")
        mensagens.append("Ocorreu um erro ao buscar as tarefas.")
    return "\n".join(mensagens)
```

Por fim, a lista `mensagens` é convertida em uma única *string*, separada por quebras de linha por meio do método `.join`.

5.3.5. Função de Enviar Mensagem

A seguir, o *script* apresentado tem como objetivo enviar mensagens via *WPPConnect*. Inicialmente, são importadas bibliotecas essenciais: ***requests*** para requisições *HTTP*, **os** para acesso a variáveis de ambiente e funções auxiliares importantes do sistema que são trabalhadas em modulos.

Listing 5.16: Importando módulos necessários para a função de enviar mensagem.

```
    import requests
    import os
    from utils import log

    #Funções auxilares importadas
    from credentials import carregar_credenciais
    from calendar_events import carregar_eventos
    from calendar_tasks import carregar_tarefas
```

Define-se variáveis que armazenam dados recuperados pela biblioteca **os**, sendo necessários para efetuar a chamada ao *endpoint*:

```
\textit{POST/api/{session}/send-message}
```

Listing 5.17: Definindo variáveis necessárias para a chamada de *endpoint*.

```
    # Nome da sessão do WPPConnect
    session = 'NERDWHATS_AMERICA'

    # Obter variáveis do ambiente
    CALENDAR_ID = os.getenv("CALENDAR_ID")
    WHATSAPP_NUMBER = os.getenv("WHATSAPP_NUMBER")
    WPP_API_SECRETKEY = os.getenv("WPP_API_SECRETKEY")
```

Após definir as variáveis necessários, criaremos uma função chamanda **enviarMensagem** que recebe um parâmetro chamado mensagem.

Listing 5.18: Bloco de código da requisição e postagem da mensagem.

```
    def enviarMensagem(mensagem):
        try:
            url = f'http://localhost:21465/api/{session}/send-message'
            headers = {
                'Authorization': f'Bearer {WPP_API_SECRETKEY}'
            }
            payload = {
                'phone': WHATSAPP_NUMBER,
                'message': mensagem
            }
            response = requests.post(url, json=payload, headers=headers
                )

    Restante do código nos próximos anexos...
```

No bloco ***try***, construímos a *URL* do *endpoint* utilizando o nome da sessão, declarada na variável *session* e o *endpoint* `/send-message`. Em seguida, é criado o cabeçalho de autenticação com o *token* pela linha `Authorization: bearer` e definido o corpo da requisição pelo *payload* com o número de telefone e a mensagem. A função *requests.post* é usada para enviar a requisão *HTTP* ao servidor *WPPConnect*.

Listing 5.19: Condicionais que verificam se a mensagem foi encaminhada.

```
    def enviarMensagem(mensagem):
        try:
            ...
            response = requests.post(url, json=payload, headers=headers
                )

            if response.status_code == 200:
                log("Mensagem enviada com sucesso!")
            else:
                log(f"Erro ao enviar mensagem: {response.text}")
        except Exception as e:
            log(f"Erro ao enviar mensagem: {e}")
```

Após o envio, a resposta é verificada. Caso o status seja 200 (sucesso), a mensagem de confirmação é registrada. Caso contrário, o erro retornado pelo servidor é exibido no *log*. Todavia, o bloco ***except*** captura exceções que possam ocorrer durante a execução, como falhas de conexão ou formatação incorreta da requisição, garantindo que erros sejam tratados de forma segura e registrada no sistema.

5.3.6. Função *Log*

O módulo auxiliar *utils.py* contém uma função fundamental para o diagnóstico e acompanhamento do sistema. A função ***log(msg)*** imprime mensagens no *terminal* precedidas pela data e hora, o que facilita a identificação de erros e o rastreamento do comportamento da aplicação ao longo do tempo. Seu funcionamento é simples, mas eficiente:

```
    from datetime import datetime

```

```
3     def log(msg):
4         print(f"[{datetime.now().strftime('%Y-%m-%d %H:%M:%S')}] {msg}"
              )
```

Esse recurso simples é útil na automação por permitir rastreamento de erros sem ferramentas externas, sendo leve, reutilizável e fácil de adaptar. A integração com o *Google Calendar* ilustra uma solução prática e eficiente, baseada em boas práticas como uso de variáveis de ambiente, tratamento de exceções e retorno claro ao usuário elementos que tornam o sistema automatizado confiável e aplicável em diversos contextos.

5.3.7. Função *Main*

O código a seguir representa o *script* principal responsável por executar uma tarefa agendada diariamente, integrando eventos de calendário e tarefas com o envio automatizado de mensagens via *WhatsApp*, utilizando o *WPPConnect Server*:

Antes de tudo, precisa-se importar bibliotecas como ***schedule*** e ***time*** responsáveis pelo agendamento e controle do tempo de execução. Além do mais, importa-se as funções modularizadas que, lidam com autenticação, coleta de dados e envio de mensagens, enquanto *log* exibe *logs* no *terminal*.

Listing 5.20: Importando módulos necessários para a formatação da mensagem.

```
1     import schedule
2     import time
3     from mensagem import enviar_mensagem, carregar_credenciais,
          carregar_eventos, carregar_tarefas
4     from utils import log
```

Posteriormente, trabalharemos apenas com uma função, **tarefaDiaria()**, que é responsável em realizar a ação principal do sistema: iniciar a autenticação com os serviços do *Google*, coleta os dados e formata a mensagem, que é exibida no *terminal* e enviada via *WhatsApp*.

Listing 5.21: Bloco de código para a função de formatação da mensagem.

```
1     def tarefaDiaria():
2         log("Executando tarefa diária...")
3         service_calendar, service_tasks = carregarCredenciais()
4         eventos = carregarEventos(service_calendar)
5         tarefas = carregarTarefas(service_tasks)
6         mensagem = eventos + "\n" + tarefas
7         log(mensagem)
8         enviarMensagem(mensagem)
```

Fora da função tarefaDiaria(), criamos uma estrutura condicional, afim de que garantir que o *script* seja executado diretamente. A tarefa processada pela função tarefaDiaria() é agendada para rodar todos os dias em um horário escolhido pelo usuário, parâmetros de tempo e tarefa passados para os métodos do módulo ***schedule***.

Listing 5.22: *Script* condicional que verifica a execução direta da mensagem.

```
1     if __name__ == "__main__":
2         schedule.every().day.at("10:33").do(tarefaDiaria)
3         log("Bot iniciado e aguardando o horário programado...")
4         while True:
```

```
5             schedule.run_pending()
6             time.sleep(30)
```

Em seguida, um *loop* infinito verifica se há tarefas pendentes para executar a cada 30 segundos, mantendo o *bot* ativo em segundo plano.

5.4. Segurança e Privacidade em *APIs*.

O uso de *APIs* em processos de automação e integração entre serviços impõe desafios importantes relacionados à segurança da informação e à privacidade dos dados. Esta seção aborda práticas recomendadas para o tratamento de dados sensíveis, autenticação segura, riscos relacionados ao uso de *APIs* não oficiais e ferramentas adequadas para testes.

Dados como nomes, senhas, CPF, localização e informações bancárias exigem atenção redobrada quando manipulados por sistemas automatizados. *Tokens* de autenticação, por exemplo, são recursos amplamente utilizados para acessar *APIs* de terceiros, como *Google Calendar* ou *WhatsApp*, e devem ser tratados como informações confidenciais. Seu vazamento pode resultar em acesso indevido, exposição de dados sensíveis ou comprometer a reputação da aplicação.

Para mitigar esses riscos, recomenda-se armazenar tokens e outras credenciais em arquivos *.env*, fora do controle de versão, utilizando bibliotecas como *dotenv* em *Python*. Além disso, é essencial garantir que esses arquivos estejam listados no *.gitignore*, prevenindo sua inclusão acidental em repositórios públicos. Um exemplo simples de carregamento seguro de variáveis de ambiente pode ser realizado com:

Listing 5.23: Instânciando módulo *dotenv*

```
1     from dotenv import load_dotenv
2     import os
3
4     load_dotenv()
5     API_TOKEN = os.getenv(<"Seu \textit{Token}">)
```

A **autenticação segura** é outro aspecto central. *APIs* mais robustas, como as do ecossistema *Google*, utilizam o protocolo *OAuth2*, que oferece uma arquitetura escalável e segura baseada em tokens de acesso temporários. O fluxo mais comum e seguro, conhecido como *Authorization Code Flow*, envolve a autorização explícita do usuário, emissão de um código de autorização e posterior troca por um *token* de acesso.

Tal abordagem elimina a necessidade de compartilhar senhas com o aplicativo e permite renovação de acesso por meio de *refresh tokens*. Ferramentas como o *Google OAuth 2.0 Playground* viabilizam o entendimento e testes desse processo de forma didática e segura.

Contudo, a **integração com APIs** nem sempre ocorre por vias oficiais. Em alguns projetos, como no caso abordado anteriormente, optou-se pela utilização da *WPPConnect Server* uma *API* não oficial para envio de mensagens via *WhatsApp*. Embora viabilize soluções automatizadas sem custos diretos, seu uso impõe riscos consideráveis. Dentre eles, destaca-se a possibilidade de violação dos termos de uso do *WhatsApp*, sujeitando o número utilizado a bloqueios.

Além disso, por serem mantidas por comunidades independentes, tais *APIs* não asseguram padrões consistentes de privacidade e segurança, e sua funcionalidade pode ser comprometida por alterações na estrutura do *WhatsApp Web*. Em razão disso, recomenda-

se que *APIs* não oficiais sejam utilizadas apenas em contextos experimentais, protótipos ou ambientes educativos.

Durante o **processo de desenvolvimento**, ferramentas como *Postman* e *Insomnia* são fortemente indicadas para testes seguros e organizados. Ambas permitem configurar variáveis de ambiente, simular autenticações com *Bearer Token*, inspecionar respostas e organizar coleções de chamadas *HTTP*.

Além disso, oferecem recursos de documentação e exportação de rotinas de teste, facilitando o trabalho colaborativo sem expor credenciais. Essas práticas tornam o processo de integração mais seguro, especialmente em equipes de desenvolvimento. Por fim, a Tabela 5.1 resume as recomendações fundamentais para segurança no uso de *APIs*:

Table 5.1: *Checklist* de Segurança em *APIs*

Recomendação	**Descrição**
Evitar versionamento de credenciais	Nunca incluir *tokens* ou senhas em repositórios.
Uso de `.env`	Armazenar chaves e variáveis fora do código-fonte.
HTTPS obrigatório	Garantir comunicação segura entre cliente e servidor.
Compreensão da autenticação	Entender se a *API* usa chave, *OAuth2* ou outro mecanismo.
Ferramentas de teste confiáveis	Preferir *Postman* e *Insomnia* para simulações.
APIs não oficiais apenas em protótipos	Usar com cautela, fora de ambientes de produção.

Adotar essas práticas é fundamental para garantir a integridade, a confidencialidade e a confiabilidade das aplicações que dependem de *APIs* externas. A segurança não deve ser tratada como etapa complementar, mas como parte integrante de todo o ciclo de desenvolvimento.

5.5. Considerações Finais

Este capítulo apresentou uma abordagem sistemática para a automação de tarefas a partir da integração entre a *API* do *Google Calendar* e o *WPPConnect Server*, demonstrando como a combinação dessas tecnologias viabiliza soluções eficientes para o gerenciamento de compromissos e a comunicação automatizada via *WhatsApp*. A aplicação proposta evidencia o potencial das arquiteturas baseadas em serviços de A *APIs* e o papel central das *APIs* na construção de sistemas distribuídos, escaláveis e interoperáveis.

A partir da explicação dos conceitos fundamentais de automação e das vantagens proporcionadas pela adoção de *APIs RESTful*, foi possível demonstrar o desenvolvimento de uma solução prática que automatiza a recuperação de eventos e tarefas agendados no *Google Calendar*, com envio automatizado de notificações via *WhatsApp*. Essa integração é particularmente relevante diante do cenário contemporâneo de hiperconectividade e necessidade crescente por soluções que otimizem processos e promovam a eficiência operacional.

O detalhamento das etapas de configuração do *Google Cloud*, da autenticação segura via *OAuth* 2.0 e da utilização do *WPPConnect Server* para o envio de mensagens permite compreender a complexidade técnica envolvida na construção de sistemas desse tipo. Ademais, evidencia a importância do domínio de boas práticas de desenvolvimento, como a modularização do código, o uso de variáveis de ambiente para proteção de credenciais e o tratamento adequado de exceções, fatores essenciais para garantir a confiabilidade e a manutenção da aplicação.

Ressalte-se que a escolha pelo *WPPConnect*, uma *API* não oficial para integração com o *WhatsApp*, impõe desafios adicionais relacionados à segurança, à privacidade e à conformidade com os termos de uso da plataforma. A utilização de *APIs* não oficiais deve ser criteriosamente avaliada, especialmente em ambientes de produção, em razão dos riscos inerentes à instabilidade, à ausência de suporte oficial e à possibilidade de bloqueio de contas. Nesse sentido, reforça-se a recomendação de seu uso em contextos acadêmicos, protótipos ou ambientes controlados.

Outro aspecto de destaque refere-se à segurança na manipulação de dados sensíveis, como *tokens* de autenticação e informações pessoais dos usuários. Foram discutidas práticas indispensáveis para a proteção desses dados, como o armazenamento seguro de variáveis de ambiente e a utilização de ferramentas confiáveis para testes de *APIs*, tais como *Postman* e *Insomnia*. Essas práticas são fundamentais para mitigar vulnerabilidades e assegurar a integridade e a confidencialidade das informações processadas pela aplicação.

A experiência descrita neste capítulo também aponta para oportunidades de evolução e aperfeiçoamento das soluções de automação. Entre as possibilidades futuras, destaca-se a integração com sistemas de inteligência artificial para o enriquecimento da análise e priorização de compromissos, bem como o desenvolvimento de assistentes virtuais capazes de interagir de forma proativa com os usuários. Tais avanços podem ampliar ainda mais os benefícios da automação, promovendo maior personalização e eficiência na gestão de tarefas e compromissos.

Vale destacar que todos os exemplos apresentados ao longo deste capítulo foram executados localmente (*localhost*), com o objetivo de facilitar o desenvolvimento e a depuração. Contudo, para que a aplicação seja executada de forma automática e sem interrupção, em especial para o envio diário de notificações via *WhatsApp*, torna-se essencial o seu uso em ambiente de nuvem como o *Google Cloud Run, AWS Lambda* ou outro provedor equivalente. Dessa forma, garante-se que o serviço esteja sempre ativo, sem depender de uma máquina local ligada todo o tempo, e possibilita escalabilidade, monitoramento e uma maior disponibilidade.

A integração entre o *Google Calendar* e o *WPPConnect Server* constitui um exemplo concreto do potencial transformador das *APIs* no contexto da automação de processos, demonstrando como a adoção de tecnologias bem estruturadas e a observância de boas práticas podem resultar em soluções funcionais e alinhadas às demandas atuais de produtividade e eficiência. A abordagem aqui apresentada contribui para a disseminação do conhecimento técnico sobre integrações de sistemas e reforça a importância de se considerar aspectos éticos, legais e de segurança na implementação de soluções baseadas em *APIs*.

Portanto, espera-se que os conceitos, metodologias e práticas descritos possam

orientar novos desenvolvimentos, pesquisas e aplicações no domínio da automação de tarefas, promovendo inovação e agregando valor em diferentes contextos, tanto pessoais quanto corporativos.

5.6. Referências Bibliográficas

BARRETO, Danilo Alves Paes et al. AUTOMAÇÃO E PROCESSOS ADMINISTRATIVOS: DESAFIOS E OPORTUNIDADES NA ERA DIGITAL. **Revista ft**, Revista ft Ltda, 2024.

BRANDÃO, Rodrigo. O futuro do trabalho: Entre a automação e a integração entre humanos e máquinas. **Revista Eletrônica Internacional de Economia Política da Informação da Comunicação e da Cultura**, v. 23, n. 3, p. 39–55, 2021.

EILAM, Eldad. **Reversing: Secrets of Reverse Engineering**. Indianapolis: Wiley, 2011.

FIELDING, Roy Thomas. **Architectural Styles and the Design of Network-based Software Architectures**. 2000. PhD thesis – University of California, Irvine. Disponível em: https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm. Acesso em: 23 maio 2025.

GOOGLE DEVELOPERS. **Google Calendar API Overview**. [S.l.: s.n.], 2025. https://developers.google.com/calendar/api/guides/overview. Acesso em: 23 maio 2025.

MELO MESSIAS, Márcio de et al. O uso de ferramentas Google para a produção automatizada de relatórios da aprendizagem em turmas de séries iniciais para as escolas públicas do município de Fortaleza-CE. **Research, Society and Development**, v. 11, n. 17, e122111739151–e122111739151, 2022.

SANTOS, Elizabete. Gestão de processos: desafios e benefícios da implantação de uma plataforma de automação para a otimização de processos. Universidade Federal de Mato Grosso do Sul, 2023.

SOMMERVILLE, Ian. **Engenharia de Software**. 10. ed. São Paulo: Pearson Universidades, 2019. ISBN 9788543024974. Available from: <https://archive.org/details/sommerville-engenharia-de-software-10e>.

WPPCONNECT. **WPPConnect Server - API for WhatsApp integration**. [S.l.: s.n.], 2025. https://github.com/wppconnect-team/wppconnect-server. Acesso em: 23 maio 2025.

Capítulo 6

Aplicação das técnicas de Rigging e Normal Map na Unity para criar animações 2D com profundidade e iluminação dinâmica a partir de uma única imagem

Héder Pereira Rodrigues Silva, Isabele Rodrigues de Souza, Iallen Gábio de Sousa Santos

Abstract

In this chapter, we will explore the creation of 2D animations through the application of two techniques: rigging, to structure and animate characters, and the use of normal maps, to simulate surface relief and enable dynamic lighting effects. These techniques will be implemented using the Unity engine. Additionally, we will use artificial intelligence tools as creative support throughout the process. The course will have a visual and conceptual focus, with no programming involved. The animations produced can be used in game development, marketing materials, animated video production, among other applications.

Resumo

Neste capítulo, exploraremos a construção de animações 2D através da aplicação de duas técnicas: o rigging, para estruturar e animar personagens, e o uso de normal maps, para simular relevos e permitir efeitos de iluminação dinâmica. As técnicas serão implementadas na engine Unity. Adicionalmente vamos utilizar recursos de inteligência artificial como apoio criativo durante o processo. O curso terá enfoque visual e conceitual, sem programação. As animações produzidas podem ser utilizadas para o desenvolvimento de jogos, peças de marketing, produção de vídeo animado, entre outras aplicações.

6.1. Introdução

As animações 2D desempenham um papel fundamental em diversas mídias, desde jogos e aplicativos interativos até conteúdos digitais [Patel and Desai 2022]. Ademais, em setores

como o da publicidade e mídia, as animações vêm sendo utilizadas como ferramenta para a criação de propaganda desde o início da televisão até os dias de hoje [Mencari 2007]. Marcas e criadores de conteúdo recorrem a essa linguagem visual para capturar a atenção do público e transmitir ideias de forma criativa e memorável.

Nesse contexto, a animação 2D se mantém como uma ferramenta no desenvolvimento de experiências digitais. Jogos como Cuphead [MDHR 2017], por exemplo, mostram como esse estilo permanece relevante. Lançado em 2017, o título encantou o público com seu visual inspirado nas animações da década de 1930, além de ter entrado para o Guinness World Records [Guinness World Records 2019] com mais de 50.000 quadros feitos à mão — um marco para a indústria. A dedicação artesanal da equipe de desenvolvimento pode ser vista na Figura 6.1. À esquerda, Maja Moldehauer finalizando um dos 120.000 quadros do jogo. À direita, canetas consumidas em apenas duas semanas de traba- lho intenso. Mesmo após anos de seu lançamento, Cuphead continua sendo amplamente jogado e reconhecido, o que reforça o poder expressivo e a longevidade da estética 2D.

Figura 6.1. Produção de animação do jogo Cuphead [MDHR 2017]

No mercado profissional, a animação 2D está presente tanto em grandes estúdios e agências quanto nas mãos de profissionais autônomos. Empresas de tecnologia, publicidade e entretenimento seguem valorizando artistas que dominam essa linguagem visual para enriquecer suas experiências digitais. No entanto, é especialmente entre freelancers, pequenos empreendedores, professores, estudantes e desenvolvedores independentes que a animação 2D se destaca como um produto criativo e economicamente viável. Com recursos digitais cada vez mais disponíveis, esses profissionais conseguem transformar ideias em projetos — e projetos em oportunidades reais de atuação no mercado.

Com o passar dos anos, essa linguagem visual passou por transformações e técnicas profundas. Nas primeiras animações tradicionais, como no próprio Cuphead, cada movimento exigia dezenas — às vezes centenas — de quadros desenhados à mão. Era um trabalho exaustivo, mas necessário para dar vida aos personagens. Hoje, com a multiplicação de canais, estilos e formatos, o mercado demanda produções mais rápidas e eficientes, capazes de atender a prazos curtos e grande volume de conteúdos. Para equipes pequenas ou artistas independentes, replicar a metodologia tradicional pode se tornar um grande desafio, já que exige uma carga de trabalho maior e, muitas vezes, investimentos que nem sempre são viáveis. Por isso, surgiram técnicas que tornam o processo mais prático e acessível, preservando a expressividade que caracteriza a animação 2D.

Entre essas técnicas estão o *Rigging* e o *Normal Mapping*. O *Rigging* consiste em construir uma estrutura articulada dentro do personagem, como se fosse um esqueleto invisível, como mostra a Figura 6.2. Isso permite manipular o personagem com muito mais facilidade, ajustando poses e expressões de maneira rápida, sem a necessidade de desenhar cada quadro manualmente. Dentro da engine *Unity*, uma das plataformas de desenvolvimento multiplataforma mais utilizadas no mundo para jogos e aplicações interativas [Unity 2025], o *Rigging* se integra diretamente ao fluxo de produção, oferecendo uma forma prática de construir animações mais dinâmicas e detalhadas. Essa abordagem não apenas reduz o tempo de produção de animações, como também possibilita que profissionais criem projetos complexos com alto nível técnico e visual, mesmo trabalhando de maneira independente.

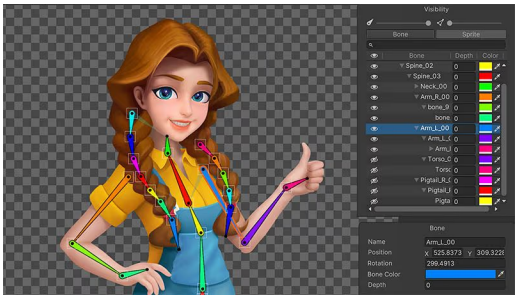


Figura 6.2. Aplicação de Rigging na Unity [Mango Animate 2023]

Outra técnica que contribui para ampliar as possibilidades visuais na animação 2D é o uso de *Normal mapping*. Embora tradicionalmente associados a gráficos 3D, os Normal Maps também podem ser aplicados em sprites 2D para criar efeitos de iluminação mais sofisticados. Eles funcionam armazenando informações sobre a orientação da superfície de um objeto em cada ponto da imagem, o que permite que a luz interaja de forma mais realista com os elementos visuais. Na prática, isso possibilita gerar sombras, brilhos e profundidade sem a necessidade de redesenhar manualmente cada variação de iluminação como visto na Figura 6.3. Para artistas independentes e pequenas equipes, o uso de Normal Maps pode representar uma grande economia de tempo e esforço, ao mesmo tempo em que eleva o nível de detalhamento e dinamismo das cenas.

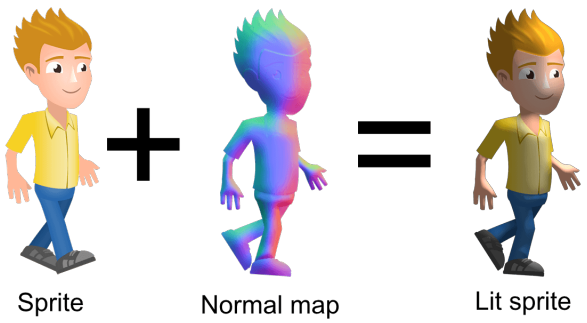


Figura 6.3. Aplicação de Normal Map no sprite

[GDQuest 2025]

Assim, ao longo deste minicurso, exploraremos as principais técnicas que tornam o desenvolvimento de animações 2D mais acessível e eficiente dentro do motor *Unity*. Através de conceitos como *Rigging* e *Normal Mapping*, será possível compreender como otimizar fluxos de trabalho, reduzir custos e, ainda assim, manter a expressividade e a qualidade estética nas produções. Nas próximas seções, vamos apresentar as bases fundamentais da animação digital, contextualizar a importância dessas práticas no mercado atual e, por fim, demonstrar na prática como implementar essas técnicas na criação de personagens e cenas interativas.

6.2. Rigging

A técnica de rigging (animação esquelética) tem raízes históricas na animação em recortes e fantoches. Lotte Reiniger, pioneira da animação de silhuetas, utilizou mecanismos de articulação física em seus famosos filmes de recortes em 1926, como Die Abenteuer des Prinzen Achmed, criando movimentos por meio de cavidades e rebites que funcionavam como juntas mecânicas [Moritz 1996].

Com a chegada da computação gráfica, esses princípios foram adaptados para o digital: a animação cutout evoluiu para o rigging virtual em softwares dedicados. Em 1988, Nadia Magnenat-Thalmann e colaboradores formalizaram o conceito de Joint-Dependent Local Deformations (JLD) para animar mãos e objetos articulados, introduzindo esqueletos hierárquicos que controlam a deformação de superfícies em tempo real [Magnenat-Thalmann et al. 1988]. Desde então, o rigging evoluiu com o surgimento de softwares especializados(como a própria Unity) que possibilitam pipelines digitais de animação 2D baseados em esqueletos.

6.2.1. Fundamentos

Na animação 2D, o rigging organiza um personagem em partes articuladas manipuláveis em um processo típico: primeiramente divide-se o desenho do personagem em camadas (corpo, membros, cabeça, olhos, boca etc.). Em seguida, define-se uma hierarquia de ossos interconectados (skeleton), onde cada osso corresponde a um segmento do personagem. Cada articulação (joint) estabelece um pivô de rotação – por exemplo, ombros e quadris são pontos de dobra naturais. A partir dos ossos, gera-se uma malha de vértices que cobre as partes do sprite; ferramentas modernas como Unity permitem autogerar essa geometria do sprite. Em seguida, atribuem-se pesos (weights) aos vértices: cada osso influencia certos vértices da malha, determinando como a imagem 2D se deforma durante o movimento. O resultado é um sistema de controle no qual o animador manipula "alavancas" em vez de redesenhar manualmente cada quadro. Isso agiliza a produção, permitindo movimentos fluídos e reutilização de animações complexas.

Exemplo de fluxo de rigging: (a ser ilustrado em figura) um personagem 2D é montado em partes; cria-se uma hierarquia de ossos e articulações; autorregenera-se a malha dos sprites; aplica-se skin weight para cada osso; então ajustam-se deformadores ou controladores inversos (IK) para animar o personagem.

6.2.2. Exemplos de Produções que utilizam Rigging

O rigging 2D é amplamente utilizado para agilizar animações em diversas produções. Por exemplo, estúdios de jogos independentes usam técnicas de cutout rig em Flash ou engines modernas: o próprio estúdio *Klei Entertainment* descreve que títulos como *Shank*, *Mark of the Ninja* e *Don't Starve Together* são animados com personagens montados a partir de partes em Flash [Kevin (Klei Entertainment) 2012]. Nesses pipelines, cada personagem é definido por cerca de uma dúzia de partes móveis (e.g. Mark of the Ninja) até dezenas (e.g. Shank), o que permite criar múltiplas vistas de cada membro e gerar sprites animados via scripts.

Na indústria de animação 2D, muitas séries de TV e curtas também se beneficiam de rigs digitais. Por exemplo, personagens de desenhos modernos (como certas séries animadas em *Toon Boom Harmony* ou *Spine*) são criados como marionetes digitais: as articulações predefinidas tornam a produção mais rápida e facilitam ajustes finos de poses. Outros casos notáveis incluem produções cinematográficas com animação híbrida (e.g. *America: The Motion Picture* usa rig de bonecos 2D) e franquias de televisão (por exemplo, *O incrível Mundo de Gumball* e *South Park*), que incorporam rigs para movimentação consistente dos personagens.

6.2.3. Softwares e Plataformas de Rigging 2D

Atualmente, existem diversas plataformas que suportam a criação de animações com Rigging, cada uma delas se especificando em algum gênero de animação em específico ou caso de uso; alguns exemplos são:

- Unity [Unity 2025]: o Unity provê um sistema completo de rigging 2D integrado. Nele, o animador pode criar cadeias de ossos no Sprite Editor, autogerar a geometria da sprite (mesh) e definir pesos automaticamente. Essa estrutura converte o sprite em um ator skinned, permitindo não apenas poses articuladas mas também aplicação de mapas normais para iluminação dinâmica nas sprites. Em versões recentes, o Unity adicionou suporte a Secondary Textures (mapas de normais e altura) em sprites animadas por ossos, o que viabiliza luzes 2D realistas e senso de profundidade em jogos 2D.

- Spine [Software 2025b]: software dedicado a animação esquelética 2D. Permite importar imagens, criar ossos hierárquicos, pintar pesos e exportar a animação para várias engines (incluindo Unity) via runtimes. É amplamente usado em jogos por ser leve e flexível.

- Toon Boom Harmony [Inc. 2025]: padrão da indústria de animação 2D para TV e filmes. A versão Premium conta com um poderoso sistema de rigging: art-redecut (montagem de personagens em partes), ferramentas de deformação (bones, curvas, morphs) e gestão de cenas complexas. Riggers profissionais usam Harmony para fazer personagens articuláveis, o que acelera produções seriadas.

- Moho [LLC 2025]: ferramenta de animação 2D focada em cutout rigging. Usa esqueletos vetoriais e ossos rígidos, com editor de Smart Bones que previne distor-

ções. É popular em animação independente e educacional, oferecendo um workflow rápido de montagem.

Cada uma dessas plataformas aplica o conceito de ossos hierárquicos e pesos em sprites, tornando o rigging uma etapa fundamental de pipelines de animação 2D. Essas ferramentas consolidadas coexistem com outras (como DragonBones, Synfig e Adobe Animate) que também oferecem sistemas de rig e skin, todos visando acelerar a produção e possibilitar efeitos avançados, como iluminação dinâmica ou deformações sutis.

6.3. Normal Map

A técnica de *Normal Mapping* surgiu no final da década de 1990 como resposta à crescente demanda por realismo gráfico em aplicações tridimensionais [Cignoni et al. 1998]. O objetivo era representar detalhes visuais de relevo — como rugosidades, saliências e ranhuras — sem a necessidade de aumentar a quantidade de polígonos dos modelos 3D. Isso reduzia significativamente o custo de renderização em tempo real. Um marco importante dessa abordagem foi seu uso no jogo *Doom 3* (2004), que popularizou o efeito ao aplicar o *Normal Mapping* para gerar superfícies detalhadas em ambientes com iluminação dinâmica [id Software 2004]. Desde então, a técnica foi adotada amplamente em engines de jogos e softwares de renderização.

Com a evolução das engines 2D, o uso de *normal maps* passou a ser explorado também em ambientes bidimensionais, adaptando seus princípios à estética e aos desafios técnicos do 2D. Em essência, um *normal map* é uma textura que armazena, em cada pixel, informações sobre a orientação da superfície naquele ponto — codificadas como vetores tridimensionais representados pelas cores RGB. Ao aplicar esse mapa a um sprite, o sistema de renderização interpreta como a luz deve se comportar sobre ele, criando áreas de sombra e brilho como se houvesse relevo [GDQuest 2020].

6.3.1. Fundamentos

A técnica de *normal mapping* em ambientes 2D se baseia na utilização de uma textura complementar, conhecida como *normal map*, para simular relevo e profundidade em imagens planas. Cada pixel dessa textura codifica a orientação da superfície em **coordenadas tridimensionais X, Y e Z**, representadas pelas **cores RGB**. O canal R representa a direção horizontal, o G a direção vertical e o B a inclinação da superfície em relação ao observador. Essas informações permitem que a engine calcule como a luz interage com cada ponto da imagem, gerando efeitos de sombra e brilho.

O fluxo básico da aplicação envolve, inicialmente, a criação de um sprite base (imagem principal do personagem ou objeto). Em seguida, produz-se um *normal map* correspondente, seja manualmente, com pintura direta dos vetores de iluminação, ou automaticamente, por meio de softwares especializados. Esse *normal map* é então associado ao sprite dentro da engine, onde será interpretado por um sistema de iluminação 2D. Durante a execução da cena, o motor gráfico combina a direção da luz com os vetores presentes no *normal map* para alterar, em tempo real, a aparência do sprite. Isso simula uma iluminação volumétrica sem alterar a geometria da imagem. O resultado é uma ilusão convincente de profundidade que mantém alto desempenho, pois não exige múltiplas versões sombreadas da imagem.

Exemplo de fluxo do *normal mapping*: (a ser ilustrado em figura) cria-se o sprite base; gera-se um *normal map* equivalente que corresponda à imagem; ambos são importados para a engine com um sistema de luz 2D; a engine processa os vetores do mapa em tempo real conforme a direção e intensidade da fonte de luz. A imagem plana reage com sombras e brilhos, criando efeitos de relevo visual.

6.3.2. Exemplos de Produções que Utilizam *Normal Mapping*

O uso de *normal mapping* em ambientes 2D tem ganhado espaço em produções que buscam enriquecer a iluminação sem comprometer a leveza gráfica. Estúdios independentes e animadores digitais utilizam essa técnica para simular relevo e profundidade em sprites, principalmente quando desejam aplicar efeitos de luz dinâmicos sem multiplicar os quadros de animação.

Um exemplo prático pode ser observado no jogo *Crawl*, desenvolvido pela Powerhoof, que utiliza *normal maps* para criar atmosferas dramáticas com lanternas oscilantes e fontes de luz em constante movimento [Powerhoof nd]. Outro caso relevante é o jogo *Dead Cells*, da Motion Twin, onde o *normal mapping* é empregado para acentuar detalhes dos cenários e personagens, como texturas de pedra, metal ou tecido, sem comprometer o desempenho [Vasseur 2020].

Na área de animação, produções experimentais e curtas independentes também vêm incorporando essa técnica. Um exemplo notável é o curta "Sprite DLight Demo Reel" (2016), desenvolvido com o software SpriteIlluminator, que demonstra diferentes formas de aplicar iluminação dinâmica em sprites 2D utilizando *normal maps* [CodeAndWeb 2023]. Além disso, ferramentas como Unity [Unity 2025] e Godot [Godot Engine 2025] vêm oferecendo suporte nativo para sistemas de luz 2D baseados em *normal mapping*, o que facilita a adoção da técnica por animadores e desenvolvedores.

Esses exemplos mostram como o *normal mapping* amplia as possibilidades expressivas da arte 2D, permitindo controlar efeitos de sombra, brilho e ambientação com mais flexibilidade e impacto visual, sem sacrificar o desempenho.

6.3.3. Softwares e Plataformas com Suporte a *Normal Mapping* 2D

Com o avanço dos motores gráficos e ferramentas de animação, diversas plataformas passaram a oferecer suporte nativo a *normal mapping* em ambientes 2D. Essa técnica permite simular relevo e iluminação dinâmica em sprites planos, proporcionando profundidade visual sem exigir modelagem tridimensional. A seguir, destacam-se algumas das principais soluções utilizadas por desenvolvedores e animadores:

- **Unity**: a partir da versão 2023.2, o Unity integra sistemas de luz 2D que suportam diretamente texturas de *normal map*. O componente *Sprite-Lit* permite que artistas associem texturas secundárias aos sprites, resultando em iluminação volumétrica com performance otimizada [GDQuest 2025].
- **Godot Engine**: a engine open-source Godot oferece suporte a *normal mapping* em seu sistema 2D. Através do recurso *CanvasItemMaterial*, é possível aplicar mapas normais, mapas de altura e efeitos de luz localizados em sprites e tiles [Godot Engine 2025].

- **SpriteIlluminator**: ferramenta especializada na criação de *normal maps* 2D. Permite gerar os mapas automaticamente ou pintá-los manualmente, com suporte de visualização da luz em tempo real. Exporta arquivos compatíveis com Unity, Godot, entre outros [CodeAndWeb 2023].

- **Materialize**: ferramenta gratuita desenvolvida pela Bounding Box Software. Com ela, é possível gerar mapas normais, de altura e especular a partir de uma imagem comum. Ideal para artistas que querem refinar detalhes manuais com controle total [Software 2025a].

- **NormalMap-Online**: aplicação web gratuita que permite gerar *normal maps* a partir de imagens de altura diretamente no navegador, sem necessidade de instalação [Cpetry 2025].

- **Laigter**: ferramenta gratuita focada na criação de *normal maps* e mapas especulares para sprites 2D, com suporte a visualização em tempo real e exportação compatível com diversas engines [Azagaya 2025].

- **Krita**: software de pintura digital que, através do *Tangent Normal Brush Engine*, permite a pintura manual de *normal maps*, oferecendo controle artístico detalhado sobre a orientação das superfícies [Foundation 2025].

- **Medibang Paint** e **Clip Studio Paint**: embora não possuam suporte nativo a *normal mapping*, esses softwares permitem que artistas pintem mapas manualmente utilizando cores RGB correspondentes às direções de luz (**R** = X, **G** = Y, **B** = Z). Esses mapas podem ser exportados como texturas normais e integrados posteriormente nas engines.

Essas ferramentas cobrem desde pipelines profissionais até abordagens manuais mais experimentais. Assim, o *normal mapping* em 2D torna-se acessível tanto para grandes estúdios quanto para artistas independentes.

6.4. Unity

O motor gráfico Unity, desenvolvido pela Unity Technologies [Unity 2025], consolidou-se ao longo dos anos como uma das plataformas mais populares para criação de jogos e experiências interativas. Lançado oficialmente em 2005 com foco na democratização do desenvolvimento de jogos, o Unity propôs uma abordagem acessível: permitir que artistas, desenvolvedores e pequenos estúdios criassem conteúdos multimídia complexos com um fluxo de trabalho integrado.

Originalmente concebido para projetos em 3D, a engine expandiu-se com o tempo para incluir suporte robusto à produção em 2D — especialmente a partir da versão 4.3 (2013), que introduziu um pipeline dedicado para sprites. Com isso, a Unity tornou-se uma ferramenta relevante não apenas para desenvolvedores de jogos tridimensionais, mas também para animadores, ilustradores e criadores de narrativas interativas em duas dimensões.

6.4.1. Animação 2D

No contexto da animação 2D, como já discutido nas Seções 6.2 e 6.3, a Unity oferece um conjunto de ferramentas versátil que cobre desde a importação de assets até a animação e renderização avançada. A engine suporta a manipulação de imagens vetoriais e rasterizadas, organização hierárquica de elementos, e animações baseadas em esqueletos — uma técnica detalhada na Seção 6.2. Além disso, permite a aplicação de efeitos visuais sofisticados, como sombras e iluminação dinâmica, que se beneficiam de mapas de normais, como abordado na Seção 6.3.

Embora outras ferramentas e engines também ofereçam suporte a esses recursos, ao longo do minicurso será dada uma atenção especial à Unity como ambiente principal de desenvolvimento. Ainda assim, é importante reconhecer que, no universo da computação gráfica, existem diversas abordagens e plataformas que implementam essas mesmas técnicas com diferentes níveis de complexidade e flexibilidade.

6.5. Tutorial: como criar um protótipo de animações na Unity

Uma vez apresentados os conceitos base relacionados às técnicas de **Rigging** e **normal map**, nesta seção será feita uma demonstração das técnicas apresentadas por meio de um tutorial para a criação de um protótipo básico de animação 2D na *Unity*. Mostraremos como fazer a textura de mapeamento normal no sprite e a sua preparação para que o rigging seja feito adequadamente, além de como importamos esse sprite para a Unity. Após isso, mostraremos como criar os ossos no sprite e as animações do personagem. Então, ensinaremos como criar ativadores e gatilhos para animações. Por fim, mostraremos como implementar a luz dinâmica na cena. Ao final, teremos um pequeno cenário interativo onde será possível testar movimentos articulados e efeitos de luz dinâmica, servindo de base para projetos mais complexos.

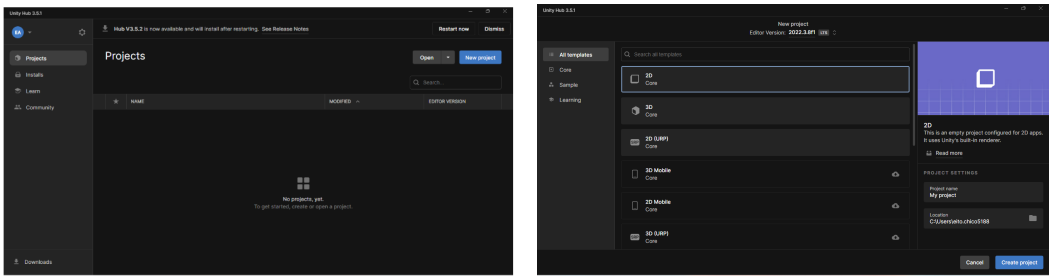
6.5.1. Preparação do Ambiente de Desenvolvimento

Antes de começarmos a desenvolver o sprite, devemos realizar alguns preparativos para a criação do protótipo. Inicialmente, é necessário fazer o download da plataforma de desenvolvimento Unity Hub, acessando o site oficial da plataforma [Unity 2025]. De acordo com o site oficial da Unity, os requisitos mínimos de sistema para utilização da plataforma são:

- OS: Windows 7 SP1+, 8, 10, 64-bit versions only; macOS 10.12+; Ubuntu 16.04, 18.04, and CentOS 7.
- GPU: Placa gráfica com recursos do DX10 (Shader Model 4.0).

Agora, criaremos o projeto do protótipo. Para isso, acesse o Unity Hub, localize o botão *New Project*, na aba de projetos, vista na Figura 6.4(a). Na aba New Project (Figura 6.4(b), selecione a opção *Universal 2D*, renomeie-o como preferir e clique no botão *Create Project*; assim, será criado seu projeto na Unity.

Ao fim deste processo de criação, aparecerá a interface de desenvolvimento básica da Unity, vista na Figura 6.5. Os elementos desta interface serão apresentados adiante.



(a) Aba Projects na Unity (b) Criação de um novo Projeto

Figura 6.4. Abas Project e New Project na Unity.

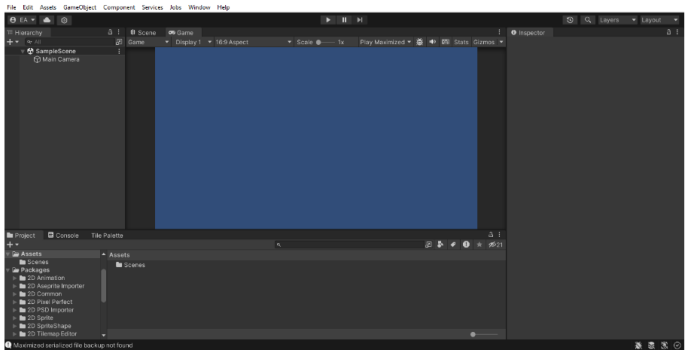


Figura 6.5. Interface Básica de Desenvolvimento

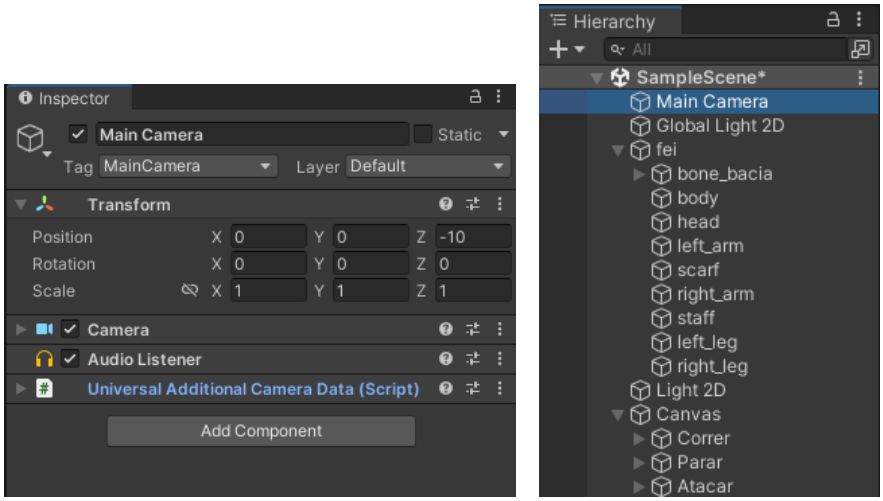
Na Unity, um projeto é representado por uma cena contendo objetos. Estes objetos contêm componentes que representam características do objeto, como: posicionamento, detector de colisões, corpo físico, elemento gráfico, elemento sonoro, etc. Além disso, é possível adicionar scripts a esses objetos para acessar as propriedades e manipulá-los.

6.5.1.1. Inspector

Localizada no lado direito do monitor, a janela inspector aborda as características e os atributos dos objetos. Nela são apresentadas as opções de configuração de um objeto dentro da Unity. Como pode-se ver na Figura 6.6(a), o objeto “Main Camera” possui o script "Universal Additional Camera Data"e os componentes “Transform”, “Camera” e “Audio Listener”.

6.5.1.2. Hierarchy

Nesta janela, localizada no lado esquerdo do visor, são organizados os objetos presentes na “cena”, como a câmera, o ponto de luz, os sprites do personagem e os seus ossos. Os objetos podem possuir sub-objetos, que são organizados e separados nesta janela, como é visto na Figura 6.6(b).

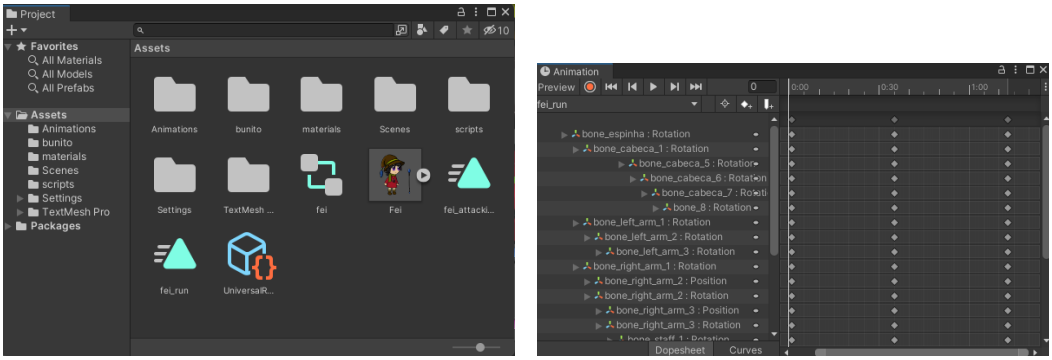


(a) Inspector (b) Hierarchy

Figura 6.6. Janelas Inspector e Hierarchy da Unity.

6.5.1.3. Project e Animation

Localizadas na região interior do editor estão as janelas Project e Animations. Na janela Project estão localizados os arquivos do protótipo, sendo possível acessar/escolher arquivos. Pode-se observar na Figura 6.7(a) que os arquivos ficam organizados em ficheiros. Na janela Animations (Figura 6.7(b) é onde as animações são desenvolvidas. Ela também possui uma aparência semelhante a um editor de vídeo, com botões de gravação e de reprodução e uma linha do tempo. Inicialmente, essa aba não está visível; para exibi-la deve-se utilizar o comando *Cntrl + 6* e, após isso, arrastar a janela para o lado da aba Project.



(a) Project (b) Console

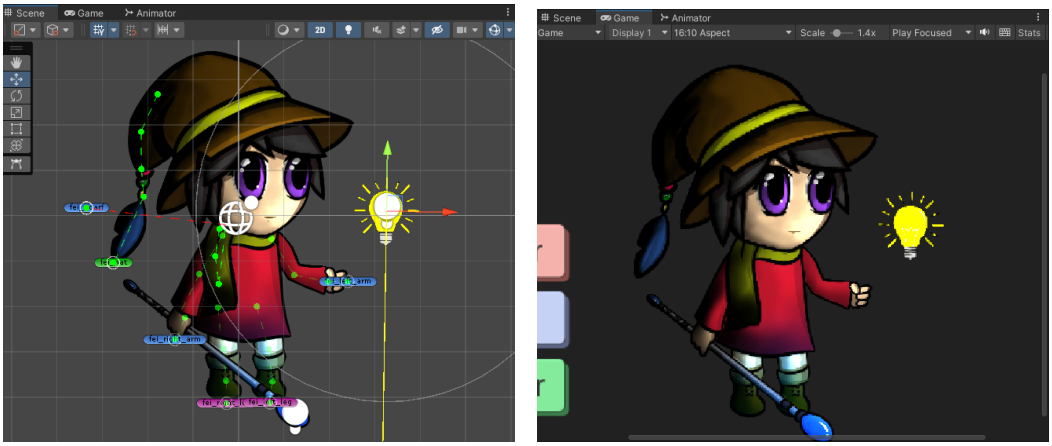
Figura 6.7. Janelas Project e Console da Unity.

6.5.1.4. Scene e Game

Em todo projeto, é necessário um local para organizar, posicionar e visualizar os objetos da cena. Na Unity, esta janela é chamada de Scene (Figura 6.8(a)). Nela, podemos alterar

características dos objetos, como suas posições, tamanho e rotação.

A janela Game possui a função de simular a visualização da animação quando ela é executada durante o processo de desenvolvimento(Figura 6.8(b)).



(a) Scene (b) Game

Figura 6.8. Janelas Scene e Game da Unity.

6.5.1.5. Animator

Essa janela é responsável por gerenciar a reprodução das animações na cena. Ela será descrita com mais detalhes posteriormente (Figura 6.9). Assim como a janela Animation, inicialmente ela não está visível, mas é exibida assim que um *Animation Controller* é criado.

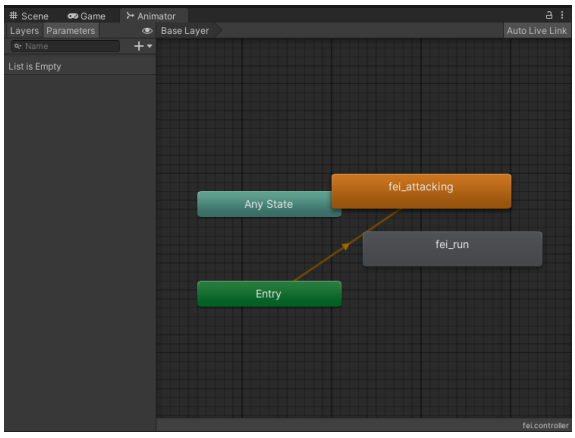


Figura 6.9. Janela Animator da Unity

6.5.2. Preparação do Sprite

Neste tutorial, vamos aplicar as animações e o mapeamento normal em um sprite obtido através da loja de *assets* da Unity, no entanto essas técnicas podem ser aplicadas em

qualquer desenho de personagem 2D. Para que a técnica de rigging seja aplicada com uma maior eficiência, é comum que seja feita a separação dos membros do personagem em camadas diferentes (Figura 6.10). Para a fabricação desse tipo de sprite, apenas é necessária a utilização de um software de desenho que possua um recurso de camadas. Neste tutorial, utilizaremos o software **Medibang Paint Pro** [MediBang Inc. 2025] para a separação do sprite e também, posteriormente, para o mapeamento normal. Após isso, cada camada deve ser exportada em um arquivo png próprio. No Medibang, isso é feito por meio do atalho *Cntrl* + *Shift* + *S* e selecionando o tipo de arquivo .png para salvar.

Figura 6.10. Separação do Sprite do personagem em camadas

Uma vez que a separação foi feita, deverá ser feita uma textura complementar de normal map para cada uma das camadas. As *coordenadas tridimencionais* - citadas na Seção 6.3 - são aplicadas utilizando uma *roda de cor normal* como base para a texturização. Exemplos de roda de cor de normal map podem ser observados na Figura 6.11(a), elas servem para mostrar quais cores indicam ângulos específicos de uma "esfera"com mapeamento normal, por exemplo: Se uma fonte de luz for colocada à direita de uma das esferas da Figura 6.11(a), as regiões com cores frias (azul, verde e roxo) serão mais iluminadas do que as regiões com cores quentes (rosa, vermelho e amarelo).

(a) Roda de Cor (b) Aplicação da Roda de Cor

Figura 6.11. Demonstração do efeito do mapeamento normal

Para criar a textura de um sprite, este raciocínio deve ser seguido: a forma como a luz deve se comportar no objeto deve ser similar à da roda de cor. Para uma melhor vi-

sualização, veja a Figura 6.11(b), perceba que as regiões pintadas com as cores que ficam mais à direita na roda de cor (regiões vermelhas e amareladas) estão mais iluminadas do que o resto da cabeça da personagem. Um pensamento simples que pode ser seguido no processo de texturização é: "Se eu colocar um ponto de luz em uma certa posição, quais partes da figura estarão mais iluminadas?". As regiões mais iluminadas serão aquelas mais próximas do ponto de luz na orientação da roda de cor, enquanto as mais escuras estarão do lado oposto à luz na roda de cor.

A texturização pode tanto ser feita em softwares especializados para criação de texturas, que dispõem de ferramentas para facilitar esse processo; porém, ela também pode ser feita em softwares de desenho com ferramentas de pintura e conta-gotas. As texturas complementares, assim como cada parte do sprite, devem ser salvas em formato .png transparente, e preferencialmente, em um diretório reservado junto com os arquivos do sprite principal. Uma boa prática para a criação dessas texturas é nomear o seu arquivo com o mesmo nome do sprite em que será aplicado juntamente com o sufixo "*_n*", por exemplo, se o arquivo do sprite da cabeça do personagem da Figura 6.11(b) for nomeado como "*head*", o nome do arquivo da textura deverá ser "*head_n*".

6.5.3. Criando os rigs de animação

Uma vez que o sprite esteja preparado, é preciso importar os arquivos para o projeto na Unity para iniciarmos o processo de **Rigging**. Para isso, no explorador de arquivos, arraste a pasta contendo os sprites e normal maps para dentro da janela *Project*, que está na interface do projeto Unity. Feito isso, todos os arquivos .png estarão dentro dos diretórios do projeto, classificados como "Sprites 2D".

Agora, para a criação dos rigs, devemos acessar um ambiente chamado *Skinning Editor*. Clique em um dos arquivos de sprite do personagem, assim a janela *Inspector* irá mostrar suas propriedades; nessa janela, clique no botão *Sprite Editor*, isso abrirá uma nova janela do sprite editor; no canto superior esquerdo, clique no menu dropdown e selecione *Skinning Editor*, isso levará a um ambiente semelhante ao da Figura 6.12.

Figura 6.12. Janela Skinning Editor Unity

Esse ambiente possui as seguintes abas que serão utilizadas no tutorial:

- **Pose** - Nessa aba você pode interagir com os ossos criados e ver a sua influência no sprite e também restaurar para o formato original.

- **Bones** - Aqui são criados os ossos por meio do *Create Bone*, quando o usuário clica nessa ferramenta ele pode gerar os ossos do sprite ao clicar e arrastar para definir tamanho e direção.
- **Geometry** - A geometria é responsável por delimitar qual serão as partes afetadas pelos ossos, criando arestas e vértices ao redor do Sprite.
- **Weights** - Os pesos definem quais arestas e vértices serão deformados pelos ossos e a intensidade da influência. As áreas de influência de cada osso são representadas pelas manchas coloridas

Para criar e configurar os ossos, deve-se primeiro clicar duas vezes no sprite para selecioná-lo, logo após ir para a aba de geometria e selecionar a ferramenta *Auto Geometry* e depois clicar no botão *Generate for Selected* - que aparecerá no canto inferior direito - e assim, automaticamente, a geometria do sprite será criada. Após isso, na aba *Bones*, selecione a ferramenta *Create Bone* e posicione os ossos conforme o formato do sprite, considerando a forma como aquele objeto deve se comportar. Por fim, na aba *Weights*, selecione a ferramenta *Auto Weights* e, de forma semelhante à geometria, clique no botão *Generate*. Com esses passos realizados, você vai ter um resultado semelhante à Figura 6.13, teste como os ossos estão afetando o sprite com o *Preview Pose*, se for necessário, faça ajustes utilizando *Edit Bone*, *Edit Geometry* e *Weight Slider*.

Visualizando a Figura 6.13, observa-se que o rigging na Unity é constituído de 3 principais elementos: os ossos, representados pelos ponteiros coloridos; a geometria, representada pelas linhas e pontos azuis que contornam o sprite e as linhas brancas que atravessam o sprite; e os pesos, representados pelas manchas coloridas, definindo a área de influência de cada osso.

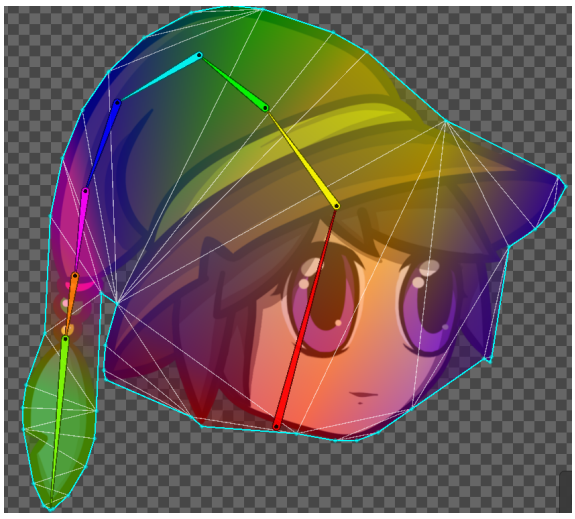
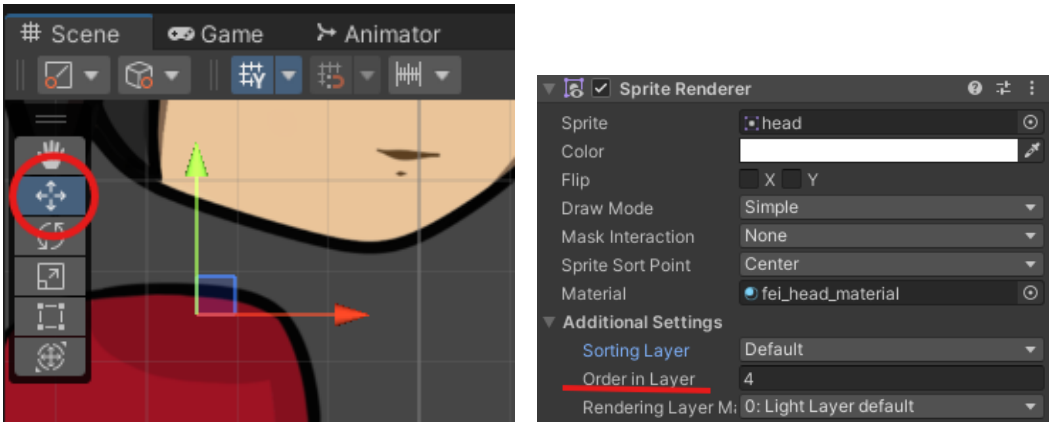


Figura 6.13. Rigging na Unity

6.5.4. Colocando o sprite no cenário e aplicando os ossos

Uma vez que o rigging for realizado em todas as partes do personagem, podemos montá-lo novamente dentro do cenário. Primeiro, vamos criar um objeto para o nosso player: na

janela *Hierarchy*, acesse o menu de contexto (pressionando o botão direito do mouse) e selecione a opção *Create Empty*, isso irá criar um objeto vazio onde iremos armazenar os ossos e os sprites do personagem, e então, renomeie o objeto para "Character". Logo após isso, crie mais dois objetos dentro de Character (faça o mesmo processo só que abrindo o menu de contexto clicando no objeto Character), e os renomeie para *Character_Sprites* e *Character_Bones*. Após isso, arraste e solte os sprites da janela *Project* para a cena, isso criará vários objetos com um componente chamado *Sprite Renderer* na hierarquia, arraste esses objetos para dentro de *Character_Sprite*. Feito isso, ajuste a posição dos sprites na cena utilizando a ferramenta de mover (Figura 6.14(a)) e também defina uma ordem de camada no componente *Sprite Renderer* - como na Figura 6.14(b) - para que as sobreposições estejam corretas.



(a) Ferramenta de Mover da cena (b) Order in Layer

Figura 6.14. Ferramenta de mover e Atributo de ordem de camada

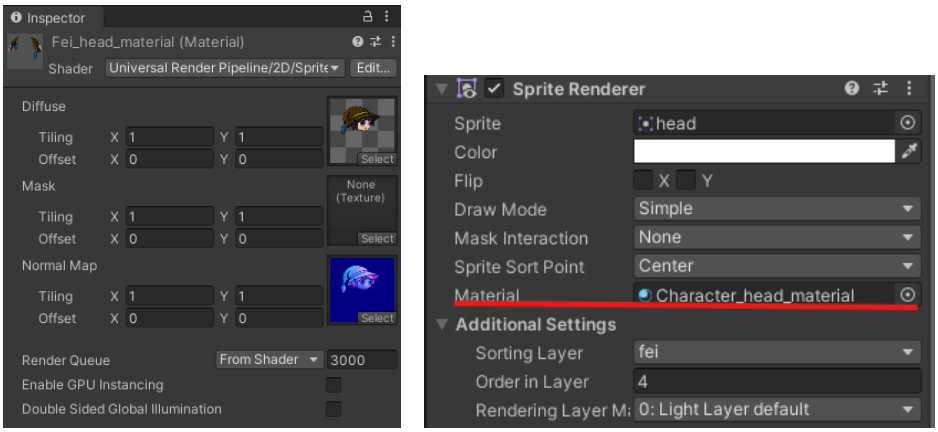
Para adicionarmos os rigs na cena, devemos adicionar um componente chamado *Sprite Skin* em todos os sprites da cena, isso é feito acionando o botão *Add Component* no inspetor do sprite e pesquisando pelo nome *Sprite Skin* na barra de pesquisa. Seguindo, pressione o botão *Create Bones* (Figura), isso criará os ossos como filhos do sprite selecionado na janela *Hierarchy*. Os ossos também têm uma hierarquia própria entre si, quando um osso pai é movimentado, todos os seus filhos são movimentados com ele, enquanto o contrário não ocorre, essa característica serve para criarmos ossos centrais, que funcionam como a espinha de todo o esqueleto do personagem. Por isso, mova os ossos para que estejam dentro do objeto *Character Bones* na hierarquia, e os organize para que componham um "esqueleto"unificado do seu personagem - esse passo é importante para facilitar a animação, e dar nomes identificativos para cada osso ajuda nesse processo. No final dessa etapa, a sua cena e hierarquia deverá estar parecida com a Figura 6.15.

6.5.5. Implementação do Normal Map no Sprite

O único passo restante antes de começarmos as animações é aplicarmos as texturas complementares no cenário. Primeiro, no diretório em que estão localizadas as texturas normais, selecione uma textura, e no *Inspector*, mude o parâmetro *Texture Type* para *normal map*, faça isso para todas as texturas. Então, crie uma pasta chamada "Materials"no diretório de assets do projeto, fazemos isso abrindo o menu de contexto na janela *Project*, se-

Figura 6.15. Rigging do Personagem na Cena

lecionando "*Create*"e então "*Folder*". De forma semelhante, crie **materiais** dentro desse diretório, seguindo os passos: Menu de contexto > Create > Material. Ao clicar em um material, o *Inspector* exibirá a interface mostrada na Figura 6.16(a). No espaço chamado *Diffuse*, clique em *Select* e pesquise por um dos sprites e selecione-o. No espaço *normal map*, adicione a textura complementar correspondente ao sprite selecionado. Tendo feito isso, só resta aplicarmos este material no *Sprite Renderer* do sprite na cena, como na Figura 6.16(b), e repetirmos esse processo para todos os membros do personagem.



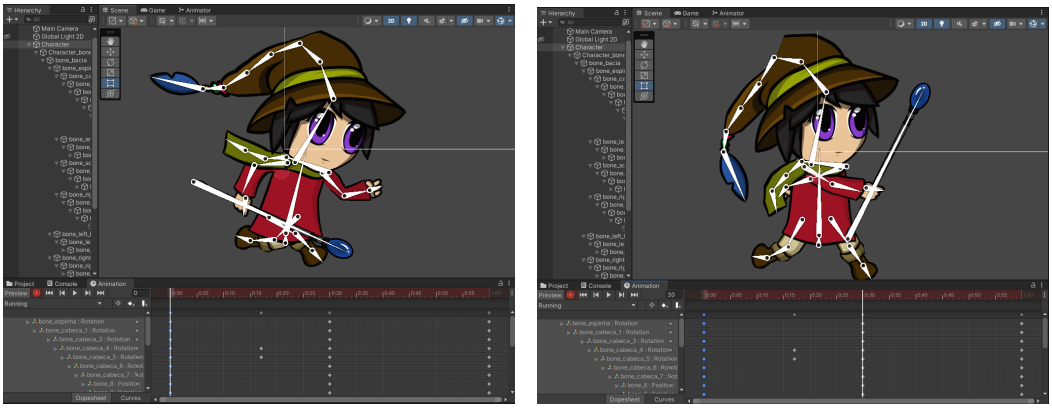
(a) Inspector de um Material (b) Material Adicionado no Sprite

Figura 6.16. Inspector de um Material

6.5.6. Montagem das animações

Com a preparação do personagem completa, crie uma nova pasta e a renomeie com o nome "*Animations*", nela armazenaremos os arquivos de animação e o controlador. Feito isso, por meio do menu de contexto, crie um arquivo do tipo *Animator Controller* e outro do tipo *Animation* e renomeie-os para "Character Controller"e "Running". Após isso, selecione o objeto "Character"na hierarquia e adicione o componente "*Animator*", então, coloque o "Character Controller"no campo *Controller* desse componente, de forma semelhante à aplicação do material no Sprite Renderer feita na seção passada.

Com esses passos, agora podemos gravar animações na janela *Animation* (Figura



(a) Keyframe 1 (b) KeyFrame 2

Figura 6.17. Gravação da Animação

6.17(a)). Vamos agora demonstrar como é feito um clipe de animação. Primeiramente, clique no botão de iniciar a gravação de keyframes - representado pelo círculo vermelho - isso fará com que o editor entre no modo de gravação - isso significa que todas as alterações feitas na cena existirão apenas como registros no arquivo Animation e serão aplicadas quando a animação for tocada. Logo, nesse estado, clique no objeto Character na hierarquia (isso tornará visíveis os ossos do personagem) e, clicando e girando os rigs, faça uma pose de corrida no personagem de forma semelhante à Figura 6.17(a). Com isso, você terá criado o primeiro *keyframe* da animação, representado pelo conjunto de pontos na linha do tempo; esses pontos são chamados de propriedades e eles são as alterações da cena feitas naquele frame, que no caso atual são apenas rotações feitas nos componentes *Transform* dos ossos movidos.

Em seguida, mova o indicador da linha do tempo para 0:30 (clique e arraste nos indicadores de milissegundos) e faça o segundo keyframe da animação, como na Figura 6.17(b). Clicando no botão de reprodução, veja que a Unity fará automaticamente a transição entre esses dois frames-chave, tirando a necessidade de cada frame ser construído como nos métodos de animação tradicional. Para terminarmos o loop de corrida, clique no primeiro keyframe (ponto mais acima, que representa todas as transformações), copie com o atalho *Cntrl + C*, mova o indicador para 1:00 e cole. Assim, clique novamente no botão de gravar keyframes para encerrar a gravação.

Com a animação feita, precisamos agora colocá-la no *Animator*, acessamos essa janela clicando duas vezes no *Animation Controller* na aba do projeto. Isso abrirá a interface chamada *Animator*, exibida na Figura 6.18. Nela, toda a animação é organizada em *States*, que representam clipes de animação reproduzidos conforme a máquina de estados avança; além dos states comuns (por exemplo a animação que criamos), existem três especiais: o *Entry*, que determina por qual animação a máquina inicia; o *Any State*, que permite disparar transições a partir de qualquer ponto; e o *Exit*, que encerra o fluxo de animação quando alcançado. Para adicionarmos a animação *Running* nesse ambiente, podemos arrastá-la da nossa janela project para dentro do controlador.

Entre esses estados, desenham-se transições, que conectam um estado a outro e podemos definir as regras de quando a máquina deve sair de um estado e entrar em outro.

Como a animação *Running* é a única que incluímos no ambiente, ela automaticamente virá com uma transição entre ela e o estado *Entry*, assim como na Figura 6.18. Feito isso, já é possível simular o protótipo (Botão de reprodução na parte superior do editor ou o atalho *Cntrl + C*), e vermos a animação que fizemos em ação na visão do usuário final.

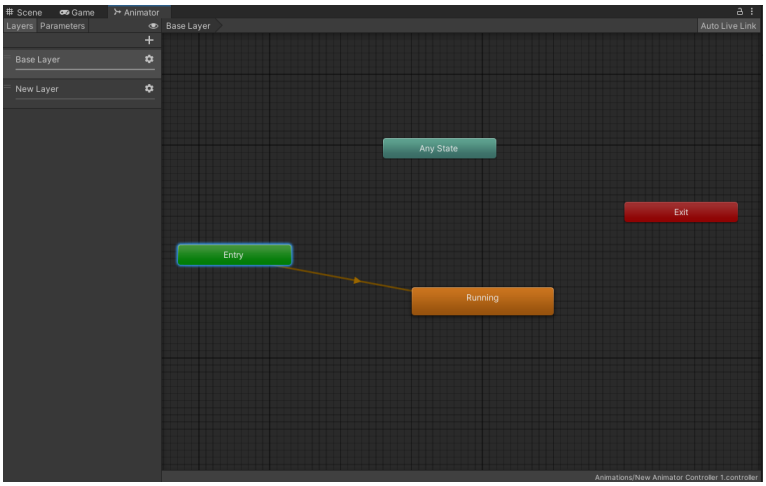


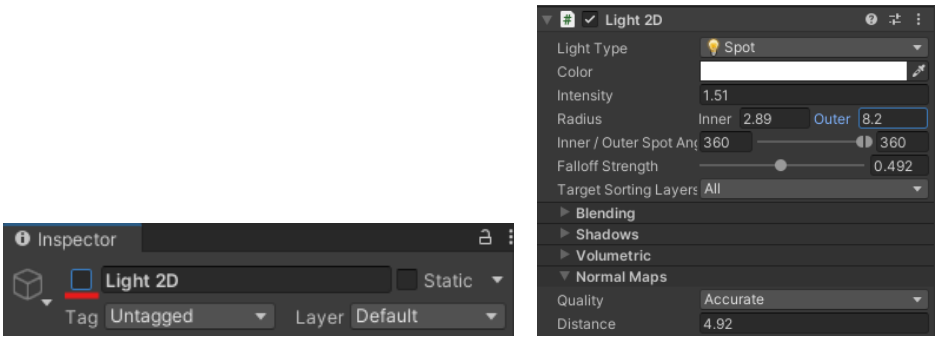
Figura 6.18. Caption

Se quiser adicionar outros estados, para criar uma nova transição, deve-se abrir o menu de contexto em um estado e selecionar *Make Transition*, e selecionar o novo estado para que a máquina irá se direcionar. Por padrão, a condição para a mudança de estados é o fim da reprodução do estado atual, se ele não for um loop, mas podemos adicionar regras para fazermos transições. Essas regras são avaliadas com base em **parâmetros** — variáveis do tipo float, int, bool ou trigger — que podem ser alteradas em tempo real pelo código ou pelo próprio Animator, guiando a evolução das animações conforme valores como velocidade ou condições lógicas. Para criar um novo parâmetro, clique no botão "+"(Figura 6.18) e selecione o tipo de parâmetro que definirá a transição; então, clique na transição, e no *Inspector* dela, adicione o parâmetro no segmento *Transitions*.

6.5.7. Criando a luz dinâmica na cena

Para podermos visualizar o efeito do mapeamento normal no personagem, devemos também utilizar um sistema de iluminação que explore este recurso. No momento, a cena está iluminada pelo objeto chamado *Global Light*, essa luz ilumina igualmente todos os objetos da cena, sem considerar o mapeamento normal. Então, para vermos o normal map em ação, devemos desabilitar a luz atual e adicionar uma luz que considere essa textura para aplicar a animação. Para isso, primeiro desative as propriedades da luz global na cena. Com esse objetivo, clique no objeto *Global Light 2D* na hierarquia - e no seu Inspector, desative o campo localizado ao lado do nome do objeto, como na Figura 6.19(a). Em seguida, na hierarquia, abra o menu de contexto, selecione a opção *Light* e depois *SpotLight 2D*, isso criará um novo objeto de luz, dessa vez uma luz limitada, que podemos mover pela cena, porém perceba que o mapeamento normal ainda não está em efeito. Para resolvermos esse problema, vá até o componente *Light 2D*, no campo *normal map*, mude o valor *Quality* para "**Accurate**", como na Figura 6.19(b). Por fim, faça ajustes nos valores *Inner* e *Outer* para deixar o raio em um tamanho favorável à visualização do

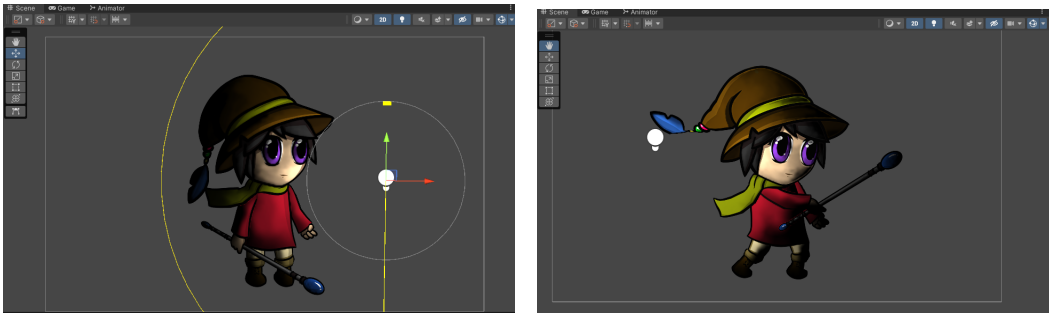
personagem. Assim, você terá uma luz que irá considerar a textura complementar dos sprites no momento de iluminá-los, dando a impressão de relevo.



(a) Desativando a Luz Global na cena (b) Configurando a nova luz Spot

Figura 6.19. Implementando a Nova Luz

Com todos esses passos concluídos, você terá um personagem totalmente articulado, com textura que reage dinamicamente à luz, permitindo a criação de diversos tipos de animações para diferentes gestos e ações. Além disso, um objeto de luz poderá ser movimentado ao redor do personagem, possibilitando a simulação de diferentes situações de iluminação, como ilustrado na Figura 6.20.



(a) Exemplo 1 (b) Exemplo 2

Figura 6.20. Resultado do Tutorial

6.6. Conclusão

Neste capítulo, apresentamos o rigging como a criação de um “esqueleto invisível” em cada sprite: uma estrutura hierárquica de ossos e articulações que permite deformar e posicionar personagens de forma fluida, sem a necessidade de desenhar todos os quadros manualmente. Mostramos que essa técnica integra conceitos tradicionais de animação e agiliza a produção ao oferecer controles diretos sobre poses e gestos. Posteriormente, introduzimos o normal mapping 2D, explicando sua origem no 3D e a adaptação para sprites: uma textura adicional cujos pixels codificam vetores de orientação de superfície em RGB, permitindo que sistemas de luz 2D gerem sombras e brilhos simulando relevo, sem aumentar a geometria ou comprometer o desempenho.

Em seguida, destacamos como Unity integra essas técnicas em um pipeline acessível e visual, sem necessidade de programação. Ao usar o 2D Animation Package, o artista pode trabalhar no Sprite Editor e no Skinning Editor para gerar automaticamente geometria, ossos e pesos, em conjunto com o sistema nativo de luzes 2D que suporta mapas normais e texturas secundárias. Esse fluxo oferece previews em tempo real — desde a montagem do rig até a aplicação de materiais com mapas normais e ajustes de iluminação.

Por fim, percorremos todo o processo prático, desenvolvendo um protótipo que aborda os conceitos principais das duas técnicas e que está pronto para servir de base a projetos mais complexos.

Portanto, espera-se que os conhecimentos adquiridos ao longo deste capítulo sejam não apenas valiosos para a compreensão do desenvolvimento de animações e texturas, mas também funcionem como uma porta de entrada para aqueles que desejam ingressar nessa área profissionalmente.

Referências

[Azagaya 2025] Azagaya (2025). Laigter - normal maps for 2d sprites. `https://azagaya.itch.io/laigter`. Acesso em: 12 maio 2025.

[Cignoni et al. 1998] Cignoni, P., Montani, C., and Scopigno, R. (1998). A comparison of mesh simplification algorithms. *Computers & Graphics*, 22(1):37–54.

[CodeAndWeb 2023] CodeAndWeb (2023). Spriteilluminator – normal map editor. `https://www.codeandweb.com/spriteilluminator`.

[Cpetry 2025] Cpetry (2025). Normalmap-online. `https://cpetry.github.io/NormalMap-Online/`. Acesso em: 12 maio 2025.

[Foundation 2025] Foundation, K. (2025). Tangent normal brush engine - krita manual. `https://docs.krita.org/en/reference_manual/brushes/brush_engines/tangen_normal_brush_engine.html`. Acesso em: 12 maio 2025.

[GDQuest 2020] GDQuest (2020). Lighting with 2d normal maps. `https://gdquest.com/tutorial/godot/2d/lighting-with-normal-maps/`.

[GDQuest 2025] GDQuest (2025). Lighting with 2d normal maps - gdquest. `https://gdquest.com/tutorial/godot/2d/lighting-with-normal-maps/`. Acesso em: 12 maio 2025.

[Godot Engine 2025] Godot Engine (2025). 2d lights and shadows. Acesso em: 14 maio 2025.

[Guinness World Records 2019] Guinness World Records (2019). Most hand-drawn frames in a video game. Acesso em 28 abr. 2025.

[id Software 2004] id Software (2004). Doom 3. Game, id Tech 4 Engine.

[Inc. 2025] Inc., T. B. A. (2025). Toon boom harmony. Disponível em: https://www.toonboom.com/products/harmony Acesso em: 2 maio 2025.

[Kevin (Klei Entertainment) 2012] Kevin (Klei Entertainment) (2012). What animation technique is used in 'Don2019t Starve'? https://gamedev.stackexchange.com/questions/44319/what-animation-technique-is-used-in-dont-starve. Answer on GameDev StackExchange.

[LLC 2025] LLC, L. M. (2025). Moho animation software. Disponível em: https://moho.lostmarble.com/ Acesso em: 2 maio 2025.

[Magnenat-Thalmann et al. 1988] Magnenat-Thalmann, N., Laperrière, R., and Thalmann, D. (1988). Joint-dependent local deformations for hand animation and object grasping. In *Proc. Graphics Interface '88*, pages 25–34.

[Mango Animate 2023] Mango Animate (2023). Animação e rigging de personagem 2d: Faça rigging 2d em 5 minutos. Disponível em: https://school.mangoanimate.com/pt/2d-character-rigging-and-animation-make-2d-rigging-in-5-minutes/. Acesso em: 31 de maio de 2024.

[MDHR 2017] MDHR, S. (2017). Cuphead. Jogo eletrônico.

[MediBang Inc. 2025] MediBang Inc. (2025). MediBang Paint Pro. https://medibangpaint.com/. Acesso em: 17 maio 2025.

[Mencari 2007] Mencari, M. A. (2007). O mundo animado nos comerciais: dos cobertores parahyba ao assolan. *Unknown Journal*.

[Moritz 1996] Moritz, W. (1996). Some critical perspectives on lotte reiniger. *Animation Journal*. Discusses early silhouette animation techniques and historical context.

[Patel and Desai 2022] Patel, R. and Desai, A. (2022). Animation techniques and trends in digital media. *Journal of Engineering Design and Analysis*, 4(2):15–22.

[Powerhoof nd] Powerhoof (n.d.). Crawl art process vid 1. https://www.powerhoof.com/crawl-art-process-vid-1/. Acesso em: 12 maio 2025.

[Software 2025a] Software, B. B. (2025a). Materialize - free texture map generator. https://www.boundingboxsoftware.com/materialize/. Acesso em: 12 maio 2025.

[Software 2025b] Software, E. (2025b). Spine - animação 2d para jogos. Disponível em: https://esotericsoftware.com/ Acesso em: 2 maio 2025.

[Unity 2025] Unity (2025). Plataforma de desenvolvimento em tempo real do unity. Disponível em: https://unity.com/pt. Acesso em: 04 de outubro 2023.

[Vasseur 2020] Vasseur, T. (2020). Art design deep dive: Using a 3d pipeline for 2d animation in dead cells. *Game Developer*. Acesso em: 12 maio 2025.