Grandes Modelos de Linguagem Multimodais (MLLMs): Da Teoria à Prática

Neemias da Silva, Júlio C. W. Scholz, John Harrison, Marina Borges, Paulo Ávila, Frances A Santos, Myriam Delgado, Rodrigo Minetto, Thiago H Silva

Abstract

Multimodal Large Language Models (MLLMs) combine the natural language understanding and generation capabilities of LLMs with perception skills in modalities such as image and audio, representing a key advancement in contemporary AI. This chapter presents the main fundamentals of MLLMs and emblematic models. Practical techniques for preprocessing, prompt engineering, and building multimodal pipelines with LangChain and LangGraph are also explored. For further practical study, supplementary material is publicly available online: `https://github.com/neemiasbsilva/MLLMs-Teoria-e-Pratica`*. Finally, the chapter discusses the challenges and highlights promising trends.*

Resumo

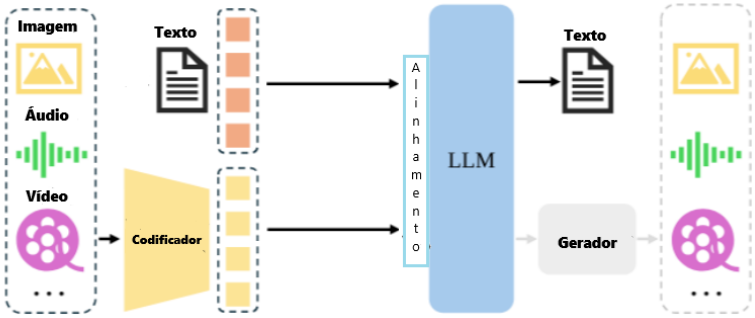
Os Grandes Modelos de Linguagem Multimodais (MLLMs do inglês Multimodal large language models) combinam a capacidade de compreensão e geração de linguagem natural dos LLMs com habilidades de percepção em modalidades como imagem e áudio, constituindo um avanço central na IA contemporânea. O capítulo apresenta os principais fundamentos dos MLLMs e modelos emblemáticos. Também são exploradas técnicas práticas de pré-processamento, engenharia de prompts e construção de pipelines multimodais com LangChain e LangGraph. Para aprofundamento prático, material complementar está disponível publicamente na web: `https://github.com/neemiasbsilva/MLLMs-Teoria-e-Pratica`*. Por fim, o capítulo discute desafios e aponta tendências promissoras.*

1.1. Introdução e Contextualização

Os Grandes Modelos de Linguagem Multimodais (MLLMs do inglês *Multimodal large language models*) representam um dos avanços mais significativos na interseção entre inteligência artificial, processamento de linguagem natural e visão computacional

[Caffagni et al. 2024, Wu et al. 2023, Yin et al. 2024]. Diferentemente de seus predecessores, os Grandes Modelos de Linguagem (LLMs do inglês *Large language models*) que operavam predominantemente com texto - os MLLMs possuem a capacidade notável de processar, interpretar e gerar conteúdo a partir de múltiplas modalidades, como texto, imagem, áudio e, em alguns casos, vídeo.

A Figura 1.1 traz uma visão geral da arquitetura de um MLLM. Os dados de entrada em múltiplas modalidades são primeiro codificados por modelos especializados e, assim como texto, são transformados em representações vetoriais chamadas *embeddings*. Um componente fundamental de um MLLM é o módulo de alinhamento, que adapta os *embeddings* ao formato compreensível pelo LLM (núcleo em azul formado em geral por um *decoder* de *Transformer* - o mesmo componente arquitetural que impulsiona LLMs de texto puro como o GPT-4 e o LLaMA). Este então processa os dados em conjunto, relacionando texto e outras modalidades. Assim, um MLLM consegue descrever, responder e raciocinar sobre diferentes tipos de informação. Na saída, há modelos que geram apenas texto e outros que, a partir de um mesmo processo de raciocínio interno, utilizam um gerador para converter *tokens* gerados pelo LLM em múltiplas modalidades.



Os MLLMs representam, portanto, um aprimoramento dos modelos de linguagem e visão - VLLMs do inglês *Vision-language large models* - que trabalham apenas com texto e imagem. Nesses modelos, tipicamente, um codificador de visão projeta as imagens em um espaço latente e um adaptador de *prompt* alinha essa representação com a entrada do LLM.

1.1.1. Relevância dos MLLMs na Web e Mídias Digitais

A *Web* moderna é um ecossistema essencialmente multimodal. Por exemplo, plataformas de *e-commerce* dependem da combinação de imagens de produtos com descrições textuais, redes sociais como *Instagram* e *TikTok* são construídas sobre a interação entre vídeo, áudio, imagem e texto, e até mesmo a busca na *web* está evoluindo para permitir consultas por imagem. Os MLLMs surgem como a ferramenta ideal para navegação, organização, moderação e criação de conteúdo para este ambiente complexo. Eles permitem diferentes recursos.

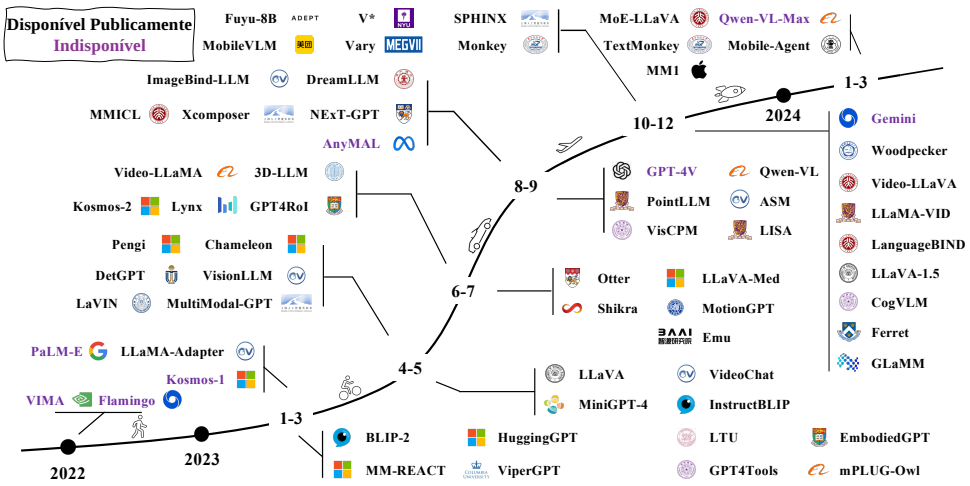
Busca e recuperação semântica: Encontrar produtos, notícias ou conteúdo multimodal com base em consultas complexas que envolvem tanto texto quanto imagem.

Acessibilidade: Gerar descrições textuais automáticas para imagens (*alt text*), tornando a web mais acessível para usuários com limitação visual.

Criação de conteúdo: Auxiliar na criação de conteúdos variados, como gerar legendas para *posts* em redes sociais, criar ilustrações a partir de rascunhos textuais ou mesmo produzir vídeos curtos baseados em um roteiro.

Moderação de conteúdo: Identificar conteúdo inadequado de forma mais contextual, analisando simultaneamente imagens e o texto associado, o que é mais eficaz do que analisar cada modalidade isoladamente.

A Figura 1.2 ilustra a rápida evolução e proliferação de MLLMs nos últimos anos, destacando a aceleração no desenvolvimento de modelos tanto proprietários quanto de código aberto, um testemunho do intenso interesse e investimento nesta área.



1.1.2. Desafios e Oportunidades no Processamento Conjunto das Multimodalidades

A motivação para o desenvolvimento e o estudo dos MLLMs reside na combinação de desafios complexos e oportunidades transformadoras decorrentes do processamento conjunto de diferentes modalidades. Os desafios são significativos [Liu et al. 2025, Kalai et al. 2025]. Diferentes modalidades existem em espaços de representação fundamentalmente distintos, por exemplo, *pixels* para imagens e *tokens* para texto. O principal obstáculo é, portanto, a criação de uma "ponte" semântica entre esses espaços, um processo conhecido como alinhamento de *embeddings*. Outros desafios críticos incluem a dessincronização de modalidades, onde o texto associado a uma imagem pode ser ambíguo, enganoso ou descrever apenas parcialmente o conteúdo visual; as alucinações multimodais, onde os MLLMs podem inventar detalhes visuais ou atributos textuais não presentes nas entradas, gerando informações incorretas com alta confiança; o custo com-

putacional elevado do treinamento e inferência com dados de alta dimensionalidade como imagens, levantando questões sobre eficiência energética e acessibilidade no desenvolvimento e no uso desses modelos; e as questões de viés e justiça, já que os MLLMs podem herdar e até amplificar vieses presentes nos dados de treinamento multimodais, levando a estereótipos prejudiciais em suas saídas.

No entanto, apesar dos desafios, os MLLMs atuais possibilitam várias aplicações em diversas áreas [Kuang et al. 2025, da Silva et al. 2025]. A capacidade de "raciocinar" sobre o mundo de forma integrada, tal como os humanos fazem naturalmente, abre um leque de aplicações anteriormente difíceis (ou não viáveis) de serem desenvolvidas. Por exemplo, MLLMs viabilizam o desenvolvimento de assistentes inteligentes avançados que podem verdadeiramente "ver" o mundo através da câmera de um *smartphone* e interagir com ele de forma contextual, ajudando usuários desde a identificação de um organismo vegetal até a solução de problemas em manuais de instrução. Na área de educação personalizada, permitem a criação de tutores interativos que utilizam diagramas, ilustrações e textos para explicar conceitos complexos de forma adaptativa. Para análise de dados complexos, possibilitam a interpretação automatizada de relatórios médicos que combinam imagens de raio-X com laudos textuais, ou análise de imagens de satélite acopladas a dados geolocalizados para monitoramento ambiental. Na automação robótica, fornecem aos robôs uma compreensão de alto nível de seu ambiente, permitindo que executem tarefas baseadas em instruções naturais, como "pegue a xícara vermelha sobre a mesa" (compreensão exemplificada por modelos como o *PaLM-E* [Driess et al. 2023]).

1.1.3. Visão Geral do Material

Este texto foi concebido para guiar os interessados em uma jornada compreensiva pelos Grandes Modelos de Linguagem Multimodais, abrangendo desde seus fundamentos teóricos até a implementação prática. A estrutura foi planejada para oferecer uma base sólida a iniciantes, ao mesmo tempo que apresenta tópicos avançados e o estado da arte para praticantes experientes. O conteúdo está organizado da seguinte forma:

- **Seção 1.2: Fundamentos de LLMs e Multimodalidade** - Esta seção estabelece as bases, revisitando a arquitetura *Transformer* e a evolução para os LLMs modernos baseados apenas em *decoders*. Em seguida, aborda-se uma parte chave da multimodalidade, explicando o conceito de alinhamento de *embeddings* e das principais arquiteturas de MLLMs, bem como uma análise de modelos emblemáticos como GPT-4V, Gemini e LLaVA.
- **Seção 1.3: Técnicas para Pipeline Multimodal** - Nesta seção são apresentadas, de forma prática, estratégias para o pré-processamento de dados multimodais e técnicas avançadas de engenharia de *prompt*. A seção inclui ainda o uso de arcabouços como LangChain e LangGraph para construir *pipelines* robustos e escaláveis que integram MLLMs a outras ferramentas.
- **Seção 1.4: Aplicações Práticas e Casos de Uso** - Esta seção apresenta exemplos concretos de aplicação de MLLMs, incluindo geração de descrições de imagens, perguntas e respostas multimodais (VQA do inglês *Visual Question Answering*) e uma análise detalhada de um estudo de caso real: o arcabouço MLLMsent para análise de sentimentos em imagens.

- **Seção 1.5: *Hands-on*: Tutorial Prático** - Nesta seção, utilizando modelos acessíveis, *notebooks* práticos mostram desde a classificação direta de imagens com MLLMs até a construção de um *pipeline* completo de classificação via descrições geradas.
- **Seção 1.6: Limitações e Oportunidades** - Esta seção discute, de forma crítica, as principais limitações atuais dos MLLMs, como alucinações, viés e custo computacional; também explora as tendências promissoras para pesquisas futuras, incluindo modelos menores e mais eficientes e o papel dos agentes multimodais.
- **Seção 1.7: Conclusões**.

Este material visa fornecer ao interessado não apenas uma compreensão teórica do funcionamento dos MLLMs, mas também as habilidades práticas necessárias para desenvolver e integrar essas tecnologias transformadoras em seus próprios projetos, contribuindo para a próxima geração de aplicações inteligentes na Web e nas mídias digitais.

1.2. Fundamentos de LLMs e Multimodalidade

Esta seção apresenta uma visão condensada das arquiteturas que sustentam LLMs e MLLMs. A Subseção 1.2.1 revisa a evolução dos *Transformers* aos LLMs modernos; na Subseção 1.2.2 são descritos os mecanismos de alinhamento de *embeddings* que viabilizam a multimodalidade; já a Subseção 1.2.3 detalha como diferentes arquiteturas de MLLMs incorporam múltiplas modalidades; e, por fim, a Subseção 1.2.4 apresenta modelos e arcabouços representativos.

1.2.1. Arquiteturas Base: Dos *Transformers* aos LLMs Modernos

A compreensão dos MLLMs requer uma fundamentação sobre a arquitetura que os sustenta. No centro dessa evolução está o mecanismo de atenção e a arquitetura *Transformer*.

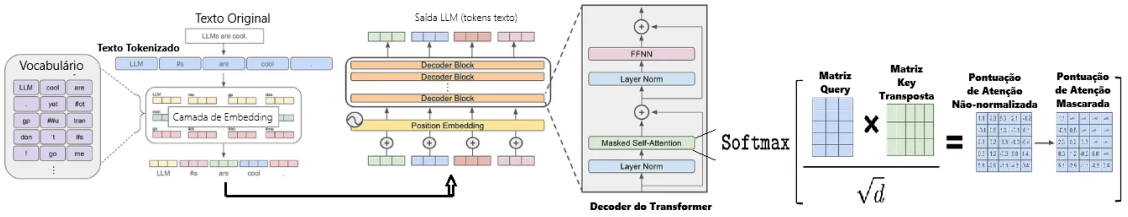
Introduzido pelo artigo seminal *"Attention is all you need"* [Vaswani et al. 2017], o mecanismo de auto-atenção (*self-attention*) permite que um modelo pondere a importância relativa de diferentes *tokens*[1] em uma sequência textual, considerando dependência de longo alcance. Esse mecanismo é a base para codificar contexto de maneira eficiente. O *Transformer* original é composto por uma pilha de *encoders*, que mapeiam uma sequência de entrada para uma representação latente, e uma pilha de *decoders*, que geram uma sequência de saída a partir dessa representação. Essa arquitetura inovadora abriu caminho para especializações em duas direções principais.

Modelos baseados em *encoder*, como o BERT [Devlin et al. 2019], que são otimizados para tarefas de **compreensão de linguagem**. Usam atenção bidirecional, acessando simultaneamente o contexto à esquerda e à direita de cada token, o que os torna adequados para análise de texto, classificação e resposta a perguntas condicionadas a um contexto dado. São considerados modelos não autorregressivos (preveem os *tokens* simultaneamente, sem dependência de *tokens* anteriores). Por outro lado, modelos baseados em *decoder*, como a família GPT [Radford et al. 2018, Radford et al. 2019], são otimizados

para a **geração de linguagem**. Nesse caso, a atenção é causal (ou mascarada), acessando apenas o contexto anterior na sequência para prever o próximo *token*. A geração ocorre de forma autorregressiva, *token* por *token*, o que garante fluidez na produção textual.

No uso contemporâneo, o termo LLMs refere-se predominantemente a *decoders* autorregressivos, como as famílias GPT e LLaMA [Touvron et al. 2023]. Com centenas de bilhões de parâmetros treinados em corpora massivos de texto, esses modelos demonstram capacidades emergentes de "raciocínio", execução de instruções e geração de linguagem natural [Wei et al. 2022a].

A Figura 1.3 ilustra o funcionamento geral de um LLM. Primeiro, a entrada textual é tokenizada, ou seja, dividida em unidades menores (*tokens*). Cada *token* é convertido em um vetor de representação numérica densa (*embedding*), sendo somado ao vetor de representação numérica densa da posição de cada *token* na sequência (*positional encoding embedding*). Esses vetores são enviados a uma pilha de *decoders* do *Transformer*, onde ocorre o processamento autorregressivo gerando um *token* de saída por vez. No *decoder*, a etapa central é a auto-atenção mascarada, onde cada *token* gera três vetores: **q** (*query*), **k** (*key*) e **v** (*value*); os quais, agregados para todos os *tokens* de entrada, formam as respectivas matrizes, Q, K e V. Calcula-se a similaridade entre Q e K, aplica-se *softmax* e multiplica-se pelos valores V, mas com uma máscara de elementos com valores infinitos que impede o modelo de ver *tokens* futuros. O resultado, somado a uma conexão residual, passa por uma camada de normalização, garantindo estabilidade e aprendizado eficiente. Em seguida, os vetores passam por uma rede neural de propagação direta (FFNN do inglês *feed-forward neural network*). Esse processo realizado em cada bloco do *decoder* se repete por todas as camadas do *Transformer* até a última, onde uma camada linear projeta o vetor final para o vocabulário, produzindo uma distribuição de probabilidade para o próximo *token*. Então, o modelo seleciona o referido *token* via amostragem e o realimenta no fluxo, gerando o conteúdo de maneira incremental.



A multimodalidade surge como uma extensão natural desses LLMs. Conforme mostrado na Figura 1.1, a inovação crucial reside na incorporação de *encoders* especializados que convertem entradas não-textuais (como *pixels* de imagem) em representações (*embeddings*) que podem ser projetadas no espaço latente dos LLMs. Através de técnicas de alinhamento multimodal, como ocorre no CLIP [Radford et al. 2021] (ver Figura 1.4), o modelo aprende a associar conceitos de diferentes modalidades, permitindo que este "entenda" uma imagem e converse sobre ela, ou que gere uma imagem a partir de uma descrição textual.

Modelos como o GPT-4V [OpenAI 2023b] e o Gemini [Team et al. 2023] partem da mesma base de *decoders* autorregressivos para textos, mas incorporam informações

adicionais, como visão ou áudio. A estratégia predominante consiste em projetar *embeddings* de todas as modalidades num espaço comum compreensível para o *decoder*. Assim, o núcleo autorregressivo é mantido, mas agora gera respostas para entradas multimodais. Essa herança arquitetural é o que permite aos MLLMs gerar linguagem de forma coerente sobre conteúdos além dos textuais, como visuais e auditivos.

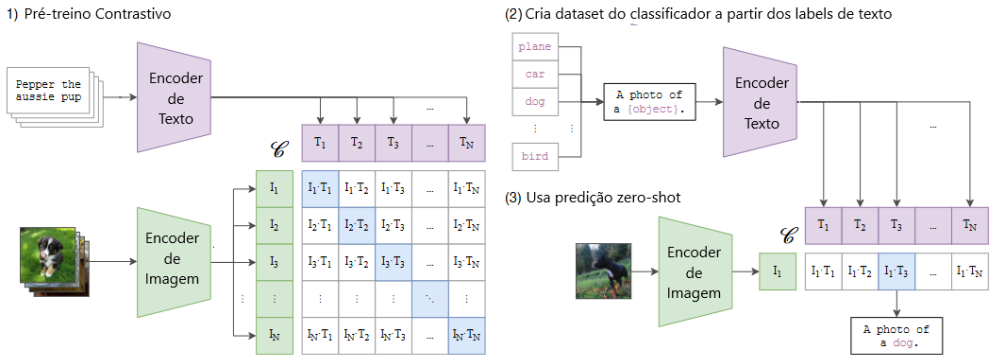
1.2.2. A Ponte para a Multimodalidade: Alinhamento de *Embeddings*

Um ponto-chave dos MLLMs é a capacidade de integrar informações de modalidades distintas, como texto, imagem e áudio, em um modelo coerente. O desafio é que cada modalidade possui uma representação numérica nativa (*tokens*, *pixels*, sinais de áudio, respectivamente), que habita espaços vetoriais distintos e não comparáveis diretamente. A solução usual está no alinhamento de *embeddings*.

Cada modalidade é inicialmente processada por uma arquitetura especializada. Modelos de linguagem baseados em *Transformer*, como o GPT, mapeiam sequências de *tokens* para um espaço vetorial $\mathscr{T}$. *Encoders* visuais, como *Vision Transformers* (ViTs) [Dosovitskiy et al. 2020], projetam *patches* de imagens em um espaço vetorial $\mathscr{V}$. Modelos de áudio, como *wav2vec 2.0* [Baevski et al. 2020], convertem sinais de áudio em um espaço $\mathscr{A}$. Sem um mecanismo de alinhamento, os espaços $\mathscr{T}$, $\mathscr{V}$ e $\mathscr{A}$ são como línguas diferentes: um modelo de linguagem é incapaz de processar nativamente representações vetoriais oriundas do domínio visual sem a devida mediação arquitetural. O *joint embedding* propõe justamente mapear essas modalidades em um espaço semântico latente compartilhado $\mathscr{C}$, ou seja, uma linguagem universal que o modelo entende. Nesse espaço, conceitos semelhantes, como a palavra "cachorro", a foto de um cachorro e o som de um latido, são projetados em regiões vizinhas [Baltrušaitis et al. 2018]. Essa projeção permite que o LLM, operando em $\mathscr{C}$, processe e gere conteúdo condicionado em múltiplas modalidades.

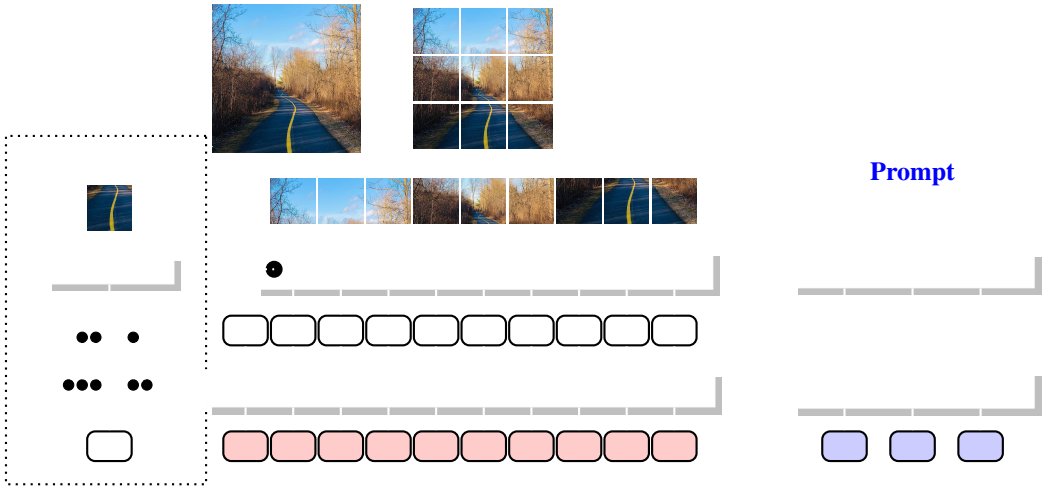
Uma técnica conhecida para produzir esse alinhamento é o aprendizado contrastivo, no qual o modelo aprende representações úteis comparando exemplos relacionados (alinhados) e não relacionados [Chen et al. 2020]. O CLIP (*Contrastive language–image pre-training*) [Radford et al. 2021] é um exemplo emblemático. Ele é treinado com milhões de pares (imagem, legenda) e, conforme ilustrado na Figura 1.4, utiliza dois *encoders*: um *encoder* de texto (ex.: *Transformer*) para mapeamento em $\mathscr{T}$ e um *encoder* de imagem (ex.: ViT) para mapeamento em $\mathscr{V}$. No CLIP, os *encoders* visual e textual são treinados conjuntamente para que ambos produzam *embeddings* em um espaço latente compartilhado $\mathscr{C}$, e o objetivo do aprendizado é maximizar a similaridade entre pares que são relacionados (diagonal principal) e minimizá-la para pares não relacionados (fora da diagonal principal). O resultado é um espaço multimodal compartilhado $\mathscr{C}$ que permite tarefas de recuperação *zero-shot* (inferência em um contexto novo, diferente do treino).

Nos MLLMs, o CLIP atua como uma ponte para a incorporação de imagens, fornecendo um espaço compartilhado $\mathscr{C}$ no qual representações visuais e textuais são alinhadas de forma simétrica [Radford et al. 2021]. Durante o treinamento do CLIP, um par de *encoders*, um visual e outro textual, é otimizado conjuntamente para ter seus *embeddings* de imagens e descrições projetados em um mesmo espaço latente, de modo que pares correspondentes fiquem próximos e pares não correspondentes se afastem. Em modelos multimodais baseados em LLMs, esses *embeddings* visuais produzidos pelo CLIP



podem ser posteriormente projetados para um espaço compreensível pelo modelo de linguagem [Liu et al. 2023], permitindo que o LLM trate a informação visual como uma forma de entrada textual compatível. Assim, o modelo não "traduz" a imagem diretamente para o texto, mas permite que o LLM interprete a informação visual como *tokens* compatíveis com seu espaço latente.

A Figura 1.5 ilustra o processo de geração dos *embeddings* de imagem e texto empregados como entrada em modelos de linguagem multimodais.



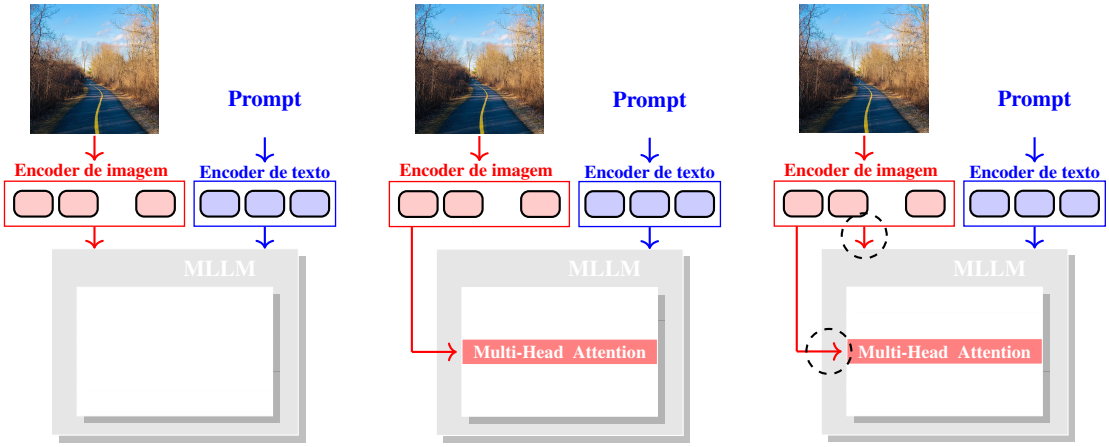
Inicialmente, a imagem fornecida pelo usuário com largura e altura de $W \times H$ *pixels* é dividida em pequenos blocos (ou *patches*) de $P \times P$ *pixels*, que são linearizados e empilhados sequencialmente. Cada *patch* é então projetado em um espaço vetorial de Q dimensões por meio de uma camada totalmente conectada, resultando em uma sequência de vetores que, juntamente com o *token* especial [CLS], formam a entrada para um *Vision Transformer* (ViT). O valor de Q é escolhido de tal forma a se encaixar na entrada de cada

token definido pelo ViT. Após o processamento pelo ViT, são obtidos os *image embeddings* correspondentes, que codificam a informação visual de maneira contextualizada. De forma análoga, a sequência textual, exemplificada pelo *prompt* "Descreva a imagem", é tokenizada, transformando cada palavra em uma unidade discreta. Esses são os *tokens* de texto, cujos *embeddings* são posteriormente projetados, assim como os *embeddings* visuais, em um mesmo espaço latente.

1.2.3. Arquiteturas de MLLMs: Como a Multimodalidade é Incorporada

Com os fundamentos estabelecidos, é possível compreender as principais estratégias de construção de MLLMs. A ideia central é utilizar o LLM como um "sistema cognitivo central", ao qual se conectam *encoders* especializados para processar outras modalidades. A questão é, portanto, de interface: como conectar essas representações ao modelo de linguagem de forma que ele possa "raciocinar" sobre informações multimodais.

Dentro desse panorama, é possível classificar as estratégias de integração multimodal em termos de fusão precoce (também chamada de *input-level fusion*) e fusão tardia (ou *model-level fusion*). Na fusão precoce, as diferentes modalidades são alinhadas e combinadas em um espaço comum logo no início do processamento, de modo que o modelo aprenda representações conjuntas desde as camadas iniciais. Já na fusão tardia, cada modalidade é processada separadamente e apenas em estágios posteriores as representações são combinadas, tipicamente dentro do LLM. A Figura 1.6 ilustra os dois tipos de integração, assim como um terceiro que agrega as duas estratégias, chamado de fusão híbrida. A escolha entre fusão precoce, tardia ou híbrida envolve importantes *trade-offs* práticos em termos de custo computacional, reuso de modelos e capacidade de generalização.



Fusão precoce oferece o maior grau de interação entre modalidades, pois o modelo aprende representações conjuntas desde as camadas iniciais. No entanto, essa abor-

dagem exige grandes volumes de dados multimodais anotados e treinamento ponta a ponta, o que aumenta o custo e o tempo de ajuste fino.

Fusão tardia é mais eficiente do ponto de vista de engenharia, pois permite reaproveitar *encoders* e LLMs já pré-treinados. Ela requer apenas um módulo adicional para combinar as representações - frequentemente por meio de blocos de *cross-attention* - mas pode gerar integrações mais superficiais, com menor sinergia entre modalidades.

Fusão híbrida busca equilibrar as limitações descritas anteriormente, combinando a integração inicial de *embeddings* com mecanismos de fusão em camadas intermediárias. Isso tende a melhorar a flexibilidade e a coerência multimodal, embora introduza complexidade arquitetural e desafios de otimização.

Um componente técnico central nas arquiteturas modernas de fusão tardia e híbrida é o mecanismo de atenção cruzada (*cross-attention*). Diferentemente da autoatenção, na qual cada token interage apenas com outros *tokens* da mesma sequência, a atenção cruzada permite que *tokens* textuais consultem representações visuais ou auditivas em cada etapa de decodificação.

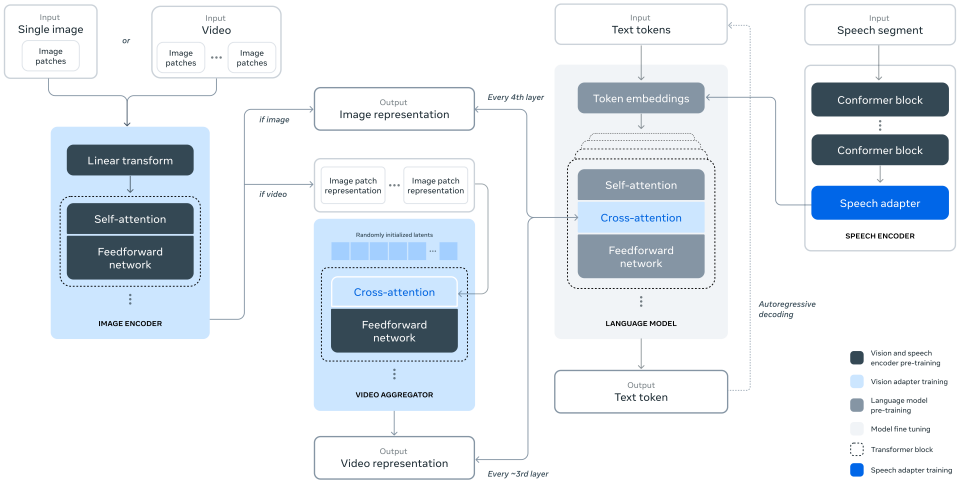
Em modelos como o *Flamingo* [Alayrac et al. 2022], blocos específicos de *cross-attention* são intercalados nas camadas do LLM, funcionando como portas de comunicação entre modalidades. Esses blocos aprendem a pesar dinamicamente quais partes da imagem são mais relevantes para o texto sendo gerado, viabilizando raciocínio multimodal contextual. Essa abordagem preserva o núcleo linguístico do LLM intacto, mas adiciona uma camada de alinhamento adaptativo, na qual o modelo aprende não apenas a compreender a imagem, mas também a consultá-la seletivamente durante a geração textual (um aspecto essencial para tarefas como descrição visual, interpretação de gráficos e raciocínio *visual-textual*).

1.2.4. Modelos Emblemáticos e Arcabouços

A trajetória recente dos MLLMs pode ser ilustrada por alguns modelos emblemáticos que marcaram diferentes fases de sua evolução. Entre eles, destaca-se o GPT-4V(*ision*) [OpenAI 2023b], que estende o GPT-4 para processar imagens. Embora detalhes arquiteturais não sejam públicos, acredita-se que siga o paradigma de encapsulamento de modalidades, integrando um *encoder* visual avançado ao núcleo de linguagem. Sua contribuição reside na demonstração de que modelos comerciais de larga escala podem realizar raciocínio visual sofisticado, incluindo tarefas de OCR e interpretação contextual de imagens do mundo real.

O Gemini [Team et al. 2023] representa outro marco, anunciado como a primeira família de modelos nativamente multimodais em sua concepção. Diferente de arquiteturas adaptadas, foi projetado desde o início para processar texto, imagem, áudio e vídeo de forma integrada. Sua arquitetura baseia-se em *Transformers* modificados e treinados ponta a ponta em dados multimodais, permitindo uma interação mais profunda entre modalidades. A família de modelos Gemini inclui variantes otimizadas para diferentes contextos de uso, desde dispositivos móveis até grandes servidores de processamento.

O PaLM-E [Driess et al. 2023], desenvolvido pelo Google DeepMind, é um modelo multimodal voltado para a robótica, que combina percepção visual e textual com



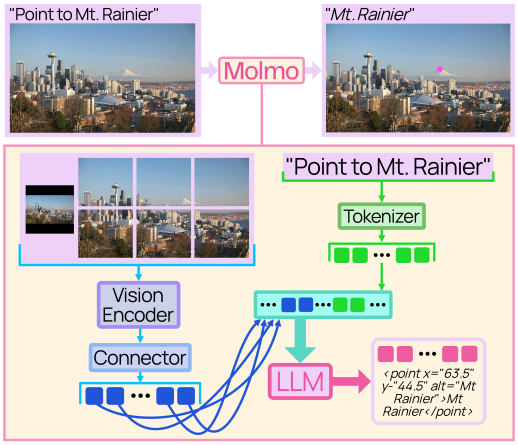
capacidade de planejamento e tomada de decisão em ambientes físicos. Seu diferencial é demonstrar como os MLLMs podem ser incorporados em agentes encarnados, aproximando a inteligência artificial de aplicações práticas no mundo real.

Outro modelo relevante é o Llama 3.2 [Grattafiori et al. 2024], disponibilizado pela empresa Meta. Este modelo é um exemplo ilustrativo da estratégia de integração multimodal por fusão tardia, como mostrado na Figura 1.7. Como se observa, além da modalidade textual, o modelo suporta também entradas visuais e sonoras.

Entre os modelos de código aberto, o LLaVA [Liu et al. 2023] tornou-se um dos mais influentes. Conectando o *encoder* visual do CLIP a um LLM derivado do LLaMA via projeção linear, demonstrou que a qualidade do alinhamento e o ajuste fino (*fine-tuning*) em instruções visuais são mais determinantes que a escala da arquitetura. O BLIP-2 [Li et al. 2023] segue uma linha semelhante, mas introduz um módulo intermediário de projeção treinado para alinhar imagens a modelos de linguagem pré-existentes de forma mais eficiente. Essa abordagem reduziu significativamente o custo de treinamento multimodal, impulsionando a adoção em contextos de pesquisa e aplicações abertas.

Outro modelo aberto é o Molmo [Deitke et al. 2025]. Conforme ilustrado na Figura 1.8, o codificador de imagem utiliza um *Transformer* de visão padrão, especificamente o CLIP. O termo *conector* na figura refere-se a um *projetor* que alinha os *embeddings* da imagem com o modelo de linguagem, implementando uma fusão precoce entre as modalidades. Desenvolvido pela AllenAI, o Molmo foi lançado com pesos e dados abertos, buscando oferecer uma referência transparente e reprodutível para pesquisas em MLLMs.

O modelo NVLM da NVIDIA [Dai et al. 2024] é particularmente interessante porque, em vez de focar em uma única abordagem, explora ambas as estratégias de integração multimodal. Na Figura 1.9, a parte inferior nomeada *Decoder-only* (NVLM-D)



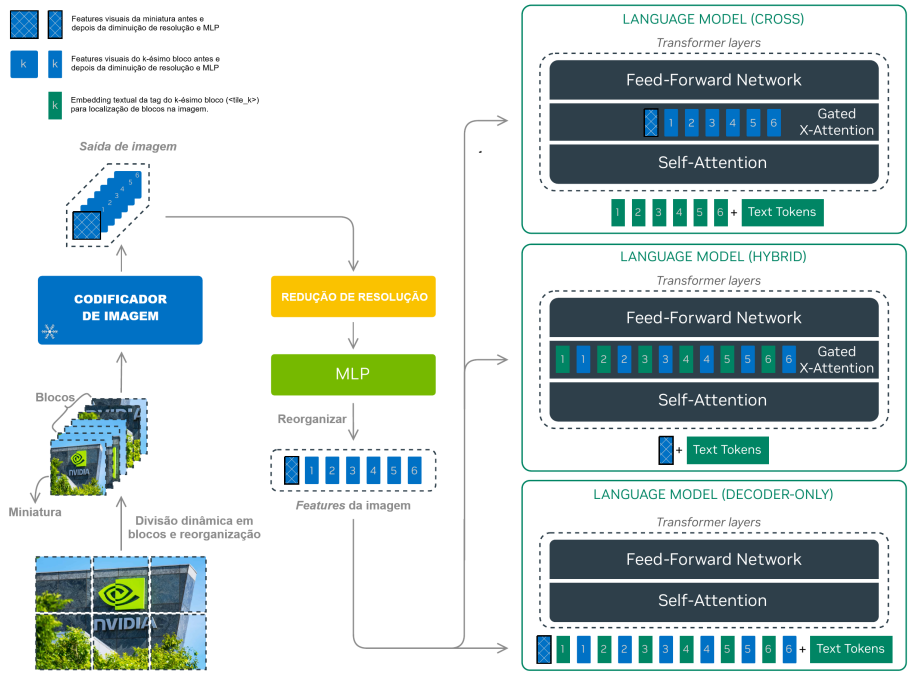
refere-se ao método de fusão precoce. A parte superior, nomeada *Cross* (NVLM-X), refere-se ao método de fusão tardia. Além disso, os autores desenvolvem uma abordagem híbrida, nomeada *Hybrid* (NVLM-H).

Os pesquisadores observam que a estratégia NVLM-X é mais eficiente, computacionalmente, para imagens de alta resolução. Já a estratégia NVLM-D apresenta melhor desempenho em tarefas relacionadas a OCR. Por fim, a estratégia NVLM-H combina os pontos fortes de ambos os métodos, NVLM-D e NVLM-X.

Por fim, ecossistemas como o Hugging Face [Wolf et al. 2020] têm desempenhado um papel crucial na disseminação dessas tecnologias. A biblioteca `transformers` fornece uma API unificada para carregar e utilizar modelos de linguagem e multimodais, além de *pipelines* prontos para tarefas como geração de legendas de imagens, VQA e tradução multimodal. Esse ecossistema facilita o compartilhamento de modelos e dados, acelera a reprodutibilidade e democratiza o acesso aos MLLMs de última geração, reduzindo a barreira de entrada para instituições acadêmicas, pequenas empresas e pesquisas independentes.

1.3. Técnicas para Pipeline Multimodal

Esta seção apresenta os componentes fundamentais para a construção de *pipelines* eficientes com modelos multimodais. O conteúdo é organizado de maneira progressiva, contemplando desde o tratamento inicial dos dados até ferramentas que permitem operacionalizar fluxos mais complexos. (i) Inicialmente, discute-se o pré-processamento multimodal, etapa essencial para garantir a qualidade das entradas e o alinhamento entre modalidades, permitindo que textos e imagens sejam representados em um espaço semântico compartilhado. (ii) Em seguida, são abordadas algumas técnicas de engenharia de *prompt*, incluindo estratégias como o *chain-of-thought* multimodal (por exemplo, descrever o raciocínio sobre uma imagem de forma incremental) e o *few-shot prompting* com pares imagem-texto para orientar o modelo em tarefas específicas. (iii) Por fim, a seção introduz arcabouços que facilitam a construção de *pipelines* avançados, tais como LangChain e LangGraph. Tais arcabouços fornecem abstrações, componentes reutilizáveis,



padronização de código e mecanismos de depuração e monitoramento, simplificando o desenvolvimento e a manutenção de sistemas multimodais.

1.3.1. Pré-processamento de dados

O pré-processamento é uma etapa crucial no desenvolvimento de pipelines multimodais, pois é responsável por converter os dados brutos (imagens - ver exemplo na figura 1.10, áudios, vídeos e textos) em representações vetoriais densas (*embeddings*) compreensíveis pelos MLLMs. Em outras palavras, é nessa fase que se realiza a "tradução" das modalidades para o espaço de linguagem do modelo.

Como discutido anteriormente, os MLLMs não operam diretamente sobre *pixels*, sinais de áudio ou *tokens* crus, mas sobre vetores que representam semanticamente cada conteúdo. A etapa de pré-processamento, portanto, consiste em (i) normalizar e limpar os dados de entrada, (ii) extrair suas representações (via *encoders* especializados), e (iii) projetar essas representações em um espaço semântico compatível com o LLM.

A implementação prática dessa etapa varia conforme a modalidade de entrada. Por exemplo, para mapear imagens e textos em um espaço multimodal compartilhado, há o modelo CLIP – uma das abordagens mais utilizadas [Radford et al. 2021]. O Código 1 mostra como carregar o CLIP e gerar *embeddings* da imagem apresentada na Figura 1.10 e um texto para posterior integração no pipeline.

```
1 from PIL import Image
2 import torch
3 import clip
```

```
from torchvision import transforms

# Carrega o modelo CLIP pré-treinado
device = "cuda" if torch.cuda.is_available() else "cpu"
model, preprocess = clip.load("ViT-B/32", device=device)

# Pré-processa uma imagem de entrada
image = preprocess(Image.open("bird.jpeg")).unsqueeze(0).to(device)

# Pré-processa o texto de entrada
text = clip.tokenize(["uma ave tesourinha em um galho de árvore"]).to(device)

# Extrai embeddings
with torch.no_grad():
    image_features = model.encode_image(image)
    text_features = model.encode_text(text)

# Normaliza e calcula similaridade
image_features /= image_features.norm(dim=-1, keepdim=True)
text_features  /= text_features.norm(dim=-1, keepdim=True)

similarity = (100.0 * image_features @ text_features.T).softmax(dim=-1)
print(f"Similaridade imagem-texto: {similarity.item():.4f}")
```

O BLIP-2 [Li et al. 2023] é uma evolução do CLIP, projetado especificamente para conectar representações visuais a LLMs, usando um módulo intermediário chamado *Q-Former*. Esse módulo aprende a projetar *embeddings* de imagem para o espaço de linguagem. O Código 2 apresenta um código escrito em Python com o BLIP-2, utilizando

a biblioteca *Transformers* da *Hugging Face*. Nota-se que neste exemplo, ao invés de calcular a similaridade, o modelo gera a descrição da imagem (*image captioning*).

```python
from transformers import Blip2Processor, Blip2ForConditionalGeneration
from PIL import Image
import torch

device = "cuda" if torch.cuda.is_available() else "cpu"

# Carrega o modelo BLIP-2
processor = Blip2Processor.from_pretrained("Salesforce/blip2-flan-t5-xl")
model = Blip2ForConditionalGeneration.from_pretrained("Salesforce/blip2-flan-t5-xl",
 ↪  torch_dtype=torch.float16).to(device)

# Pré-processa a imagem
image = Image.open("exemplo_imagem.jpg")
inputs = processor(images=image, return_tensors="pt").to(device, torch.float16)

# Extrai embeddings e gera texto condicional
generated_ids = model.generate(**inputs, max_new_\textit{tokens}=20)
generated_text = processor.batch_decode(generated_ids,
 ↪  skip_special_\textit{tokens}=True)[0]
print("Descrição gerada:", generated_text)
```

Para dados de áudio, o CLAP (*Contrastive Language–Audio Pretraining*) [Elizalde et al. 2023] desempenha um papel semelhante ao do CLIP, mapeando sons e descrições textuais para um espaço compartilhado, conforme mostrado no Código 3.

```python
from transformers import ClapProcessor, ClapModel
import torch
import torchaudio

# Carrega o modelo CLAP
processor = ClapProcessor.from_pretrained("laion/clap-htsat-unfused")
model = ClapModel.from_pretrained("laion/clap-htsat-unfused")

# Carrega um arquivo de áudio
waveform, sr = torchaudio.load("exemplo_audio.wav")

# Pré-processa o áudio e o texto
inputs = processor(audios=waveform, text=["som de pássaros cantando"],
 ↪  return_tensors="pt", padding=True)

# Extrai embeddings multimodais
with torch.no_grad():
    outputs = model(**inputs)
    audio_emb = outputs.audio_embeds
    text_emb = outputs.text_embeds
    similarity = outputs.logits_per_audio  # similaridade áudio-texto
```

```
21
22 print(f"Embedding de áudio: {audio_emb.shape}")
23 print(f"Embedding de texto: {text_emb.shape}")
24 print(f"Similaridade áudio-texto: {similarity.item():.4f}")
```

Além da geração de *embeddings* multimodais, o estágio de pré-processamento compreende procedimentos cruciais para o refinamento da qualidade e o incremento da interpretabilidade dos dados multimodais. Por exemplo, pode-se realizar a normalização, redimensionamento, segmentação de região de interesse (ROI do inglês *Region of Interest*) de imagens; remoção de ruído e normalização de amplitude em áudios; tokenização, remoção de *stopwords* e lematização em textos; geração de metadados contendo informações como formato, resolução, duração, ou anotações semânticas (e.g., entidades nomeadas), que podem auxiliar na recuperação e indexação de exemplos relevantes; e, ainda, o aumento de dados (*data augmentation*) multimodal, como variações visuais, ruído acústico ou reformulação de legendas, para aumentar a robustez do modelo. Tais práticas são especialmente importantes quando o objetivo é construir um conjunto de dados multimodais personalizados.

Desta maneira, o pré-processamento é, portanto, a ponte entre os dados brutos e o raciocínio em múltiplas modalidades do MLLM. Ele transforma *pixels*, sinais de áudio e *tokens* em vetores comparáveis, alinhando múltiplas modalidades em um mesmo espaço de linguagem, permitindo que o modelo "pense" sobre imagens, sons e textos de forma integrada.

1.3.2. Engenharia de *Prompt* para Tarefas Multimodais

Para melhorar os resultados dos MLLMs, um aspecto central é a elaboração de bons *prompts*, processo conhecido como engenharia de *prompt*. Mas o que caracteriza um "bom" *prompt*? De modo geral, é um conjunto de comandos com instruções claras, específicas e contextualizadas, nas quais a tarefa é descrita de maneira objetiva e, quando necessário, acompanhada de exemplos. Além disso, o *prompt* pode incluir instruções sobre o formato da saída (por exemplo, lista, tabela, JSON ou texto corrido) e orientações sobre estilo ou tom da resposta. A estruturação do *prompt* desempenha um papel significativo, especialmente quando ele se torna longo. Nestes casos, recomenda-se organizar o conteúdo em blocos lógicos, como listas numeradas, seções com títulos, ou palavras-chave destacadas. *Prompts* organizados facilitam o processamento interno do modelo e reduzem ambiguidades. Outra estratégia comumente utilizada é direcionar o modelo a assumir uma persona, isto é, uma configuração de comportamento ou papel atribuído ao modelo, influenciando seu estilo de resposta, tom, vocabulário e nível de detalhamento, como se estivesse "interpretando" uma identidade específica.

Com essas práticas, já é possível obter resultados relevantes por meio do *zero-shot prompting*, que consiste em solicitar que o modelo realize uma tarefa sem fornecer exemplos prévios, apenas com instruções diretas. No entanto, em tarefas mais complexas, o *zero-shot* pode não ser suficiente. Assim, utilizam-se técnicas mais avançadas, como o

few-shot prompting, *Chain-of-Thought* (CoT), *self-consistency*, ReAct (*Reason* + *Act*), e o *prompting* multimodal guiado.

O *few-shot prompting* [Brown et al. 2020] consiste em fornecer ao modelo alguns exemplos da tarefa desejada diretamente no *prompt*, antes de solicitar a resposta final. Ao observar esses exemplos, o modelo identifica o padrão de entrada e saída, reproduzindo o estilo, abordagem e nível de detalhamento esperado. Essa técnica é especialmente útil quando a tarefa é complexa, ambígua ou envolve convenções específicas, por exemplo, formato de resumo, estilo narrativo, ou critérios específicos de classificação. Em modelos multimodais, o *few-shot* pode incluir não apenas texto, mas também imagens ou combinações imagem-texto, demonstrando como relacionar as modalidades. É importante escolher exemplos curtos, representativos e consistentes, pois exemplos ruins ou variados demais podem levar o modelo a reproduzir padrões incorretos.

Já o CoT [Wei et al. 2022b] incentiva o modelo a explicitar o raciocínio passo a passo antes de apresentar a resposta final. Esse encadeamento de pensamento ajuda o modelo a organizar as informações de maneira sequencial e lógica, podendo reduzir erros e aumentar a precisão em tarefas que exigem raciocínio complexo, como problemas matemáticos, interpretação de gráficos, inferência causal e análise de imagens com múltiplos elementos. Em MLLMs, o CoT pode envolver a descrição progressiva da cena visual, destacando observações intermediárias antes de chegar à conclusão. A orientação explícita para "pensar em voz alta" tende a melhorar a qualidade da resposta, embora deva ser usada com cuidado em contextos sensíveis ou quando se deseja respostas curtas.

A técnica de *self-consistency* [Wang et al. 2022] parte da ideia de que o modelo pode gerar diferentes cadeias de raciocínio possíveis para uma mesma tarefa, especialmente quando o problema admite mais de um caminho lógico. Em vez de confiar em uma única resposta do CoT, o modelo é instruído a gerar múltiplas explicações ou raciocínios independentes, e posteriormente selecionar (ou combinar) o resultado mais recorrente ou coerente entre eles. Na prática, isso reduz o impacto de cadeias de raciocínio incorretas ou enviesadas que poderiam surgir em uma única tentativa. Apesar de aumentar a qualidade, ela implica maior custo computacional, já que requer múltiplas amostragens.

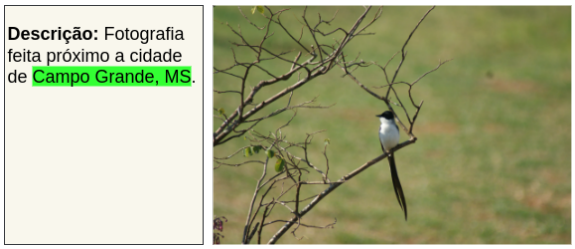
O ReAct [Yao et al. 2022] é uma técnica que combina raciocínio verbal e ações iterativas para resolver problemas de forma interativa. O modelo não apenas descreve o raciocínio, mas também executa ações intermediárias, como consultar documentos, examinar imagens novamente, fazer buscas externas ou solicitar informações adicionais. Esse ciclo "pensar - agir - pensar novamente" torna o modelo mais adequado para funções de agente autônomo, pipelines multimodais e cenários em que é necessário integrar fontes externas de conhecimento. Em MLLMs, isso pode significar interpretar uma imagem, fazer hipóteses, verificar detalhes adicionais na própria imagem ou consultar metadados, e então ajustar a conclusão final. O ReAct reduz erros relacionados a suposições precipitadas, mas depende de um ambiente que permita operações de consulta ou APIs externas.

O *prompting* multimodal guiado [Liu et al. 2023] refere-se a estratégia de orientar explicitamente como o modelo deve integrar e relacionar informações de diferentes modalidades. Por exemplo, pode-se instruir o modelo da seguinte forma: "descreva o que está na imagem e explique como isso se relaciona com o texto fornecido". Em MLLMs, simplesmente fornecer texto e imagem não garante que o modelo interpretará suas rela-

ções de forma alinhada ao objetivo da tarefa. Por isso, o *prompt* pode orientar etapas, como observar a imagem primeiro, identificar elementos-chave, relacionar esses elementos ao texto fornecido, e apenas então formular a resposta final. Essa técnica melhora a coerência entre modalidades e reduz interpretações equivocadas ou superficialmente descritivas. Ela é particularmente útil em tarefas de análise de documentos visuais, gráficos científicos, *captioning*, e explicação de conteúdo visual.

1.3.3. Arcabouços para facilitar a construção de *pipelines* avançados

A depender da complexidade da tarefa, os MLLMs, por si sós, podem não ser suficientes, o que torna necessária a criação de um *pipeline* que combine esses modelos com outras ferramentas para alcançar o objetivo desejado de forma satisfatória. Por exemplo, considerando a plataforma WikiAves[2], uma conhecida comunidade *online* de observadores de aves que mantém uma extensa base de dados sobre aves do Brasil. Supondo que essa plataforma deseje adicionar uma funcionalidade de busca por imagens, permitindo que os usuários façam *upload* de uma foto acompanhada de um breve texto com informações adicionais, como a localidade onde o registro foi feito, conforme ilustrado na Figura 1.11. Dessa forma, mesmo usuários iniciantes poderiam aprender mais sobre as aves, ainda que não conheçam seus nomes, que atualmente constituem o principal critério de busca na plataforma.

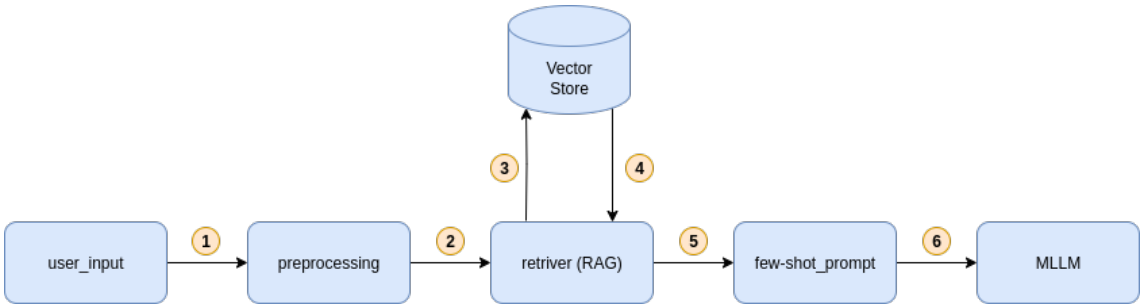


Para isso, a WikiAves pretende utilizar um MLLM capaz de processar as informações fornecidas pelo usuário (isto é, a foto e o breve texto) para identificar a espécie da ave e, em seguida, buscar em sua base de dados o registro correspondente, redirecionando o usuário para a página que contém todas as informações sobre ela. Considerando o exemplo apresentado na Figura 1.11, realizou-se um teste utilizando o modelo GPT-4o-mini-2024-07-18, da OpenAI (ou apenas, GPT-4o-mini, por simplicidade), por meio de uma requisição à API[3] do modelo, representada no Código 4.

```
{
  "model": "gpt-4o-mini",
  "messages": [
    {
      "role": "system",
```

```
      "content": "Você é um assistente especializado em análise visual. Você recebe
      ↪  como entrada uma imagem de uma ave e um breve texto com informações sobre a
      ↪  foto, e deve identificar qual ave se trata e responder o seu nome popular e
      ↪  científico. Responda em português."
    },
    {
      "role": "user",
      "content": [
        {
          "type": "text",
          "text": "Fotografia feita próximo a cidade de Campo Grande, MS."
        },
        {
          "type": "image_url",
          "image_url": {
            "url": "<URL>"
          }
        }
      ]
    }
  ],
  "max_\textit{tokens}": 500
}
```

Obteve-se a seguinte resposta do modelo: "A ave na fotografia é o **Bem-te-vi**. Seu nome científico é **Pitangus sulphuratus**", informações estas claramente erradas. Os motivos que levaram o modelo a cometer esse erro podem ser diversos, incluindo a possível falta de conhecimento sobre a fauna brasileira durante seu pré-treinamento. Para superar essa limitação, a plataforma WikiAves precisaria implementar um *pipeline* mais sofisticado para aprimorar o desempenho do modelo. A Figura 1.12 apresenta um possível *pipeline* para mitigar esse problema.



Como pode-se observar, após obter a entrada fornecida pelo usuário, o primeiro passo do *pipeline* é o pré-processamento, que, conforme discutido na Seção 1.3.1, além de gerar os *embeddings*, pode também ser utilizado para melhorar a qualidade da imagem, identificar e segmentar a região de interesse (isto é, a porção da imagem em que

se encontra a ave) e extrair as entidades nomeadas presentes no texto, neste caso, o local onde a fotografia foi capturada (a cidade de Campo Grande, MS). Com essas informações em mãos, o próximo passo do *pipeline* consiste em recuperar, a partir da base de dados da WikiAves, previamente vetorizada, ou seja, a *vector store*, registros de aves semelhantes que podem ocorrer na região informada, utilizando algum método de RAG (*Retrieve Augmented Generation*) [Lewis et al. 2020]. A partir dos resultados obtidos nessa etapa, constrói-se o *few-shot prompting*, como descrito na Seção 1.3.2, no qual são inseridos no *prompt* os exemplos de aves semelhantes recuperados via RAG. Dessa forma, o modelo pode aprender em tempo de execução com os exemplos fornecidos no contexto, para elaborar uma resposta mais assertiva.

Pode parecer um processo bem mais complexo do que simplesmente realizar uma solicitação à API do modelo, mas atualmente existem diversos arcabouços que facilitam a construção de *pipelines* como esse, tais como LangChain e LangGraph. Esses arcabouços fornecem várias ferramentas já implementadas, o que torna seu uso bastante simples, além de contribuírem para a padronização do código e oferecerem recursos para depuração de eventos.

No restante desta seção, serão fornecidos detalhes da utilização do LangChain e do LangGraph, arcabouços amplamente adotados pela indústria e pela academia, para a construção de *pipelines* com MLLMs.

1.3.3.1. LangChain

Com o LangChain [LangChain Inc. 2025a], é possível modelar um *pipeline* multimodal como uma sequência de componentes encadeados (*chains*), em que cada etapa realiza uma tarefa específica e passa os resultados para a próxima.

Cada etapa pode ser implementada como uma função encapsulada em um *RunnableLambda* ou *LLMChain*, compondo uma cadeia sequencial com o *SequentialChain*. O Apêndice A.1 mostra um exemplo em alto nível, em Python, de como implementar o *pipeline* utilizando o LangChain.

Durante a execução, o estado do *pipeline* é atualizado a cada etapa, e seus campos (como roi, rag_docs e *prompt*) são propagados entre as funções. Tal arquitetura confere transparência e modularidade ao fluxo de execução, viabilizando o desacoplamento de componentes e a substituição granular de etapas sem comprometer a integridade do *pipeline*. Essa abordagem também permite incluir ramos de decisão e verificação de erros, integrar modelos e ferramentas externas (como *embedders* multimodais, APIs e bancos vetoriais). Além disso, o LangChain possui integrações nativas com múltiplos tipos de modelos LLMs, MLLMs, *embeddings*, *retrievers*, ferramentas de visão computacional, e mais, o que simplifica o desenvolvimento de pipelines híbridos [LangChain Inc. 2025a].

A combinação de LangChain com ferramentas como LangSmith [LangChain Inc. 2025c] facilita a instrumentação, rastreamento e depuração de cada etapa do *pipeline*, sendo possível visualizar o fluxo completo de execução, incluindo as entradas e saídas de cada *chain*; medir latência, custos de chamadas e métricas de desempenho; identificar gargalos, falhas e respostas inesperadas em tempo real.

Modelar o *pipeline* com LangChain traz vantagens práticas e conceituais, tais como a **modularidade**, sendo que cada componente é independente, facilitando manutenção e testes; o **reuso**, onde as *chains* podem ser combinadas ou reutilizadas em outros fluxos; a escalabilidade, já que permite adicionar novas etapas à *chain*; e, por fim, a **integração**, tendo interoperabilidade direta com LangSmith, LangServe e LangStudio. Contudo, o LangChain apresenta algumas limitações em cenários que exigem controle mais sofisticado de fluxo e estado.

Por exemplo, cada *chain* é essencialmente uma função isolada, e o estado global do *pipeline* precisa ser propagado manualmente entre as etapas, o que aumenta a complexidade e o risco de inconsistências, especialmente em *pipelines* multimodais nos quais *embeddings* de diferentes tipos de dados (como imagem, texto e áudio) precisam ser sincronizados. Além disso, fluxos condicionais e ramificações lógicas não são tratadas de forma nativa. Implementações que envolvem verificações ou caminhos alternativos, como o caso em que nenhuma ave é detectada na imagem, acabam exigindo código imperativo fora da estrutura de *chains*, reduzindo a clareza e modularidade do projeto. Outra limitação é a ausência de uma representação gráfica explícita do *pipeline*, o que dificulta a visualização e a depuração em fluxos complexos.

1.3.3.2. LangGraph

Em contrapartida às limitações do LangChain, o LangGraph [LangChain Inc. 2025b] permite modelar o *pipeline* como um grafo de estados, no qual cada nó representa uma etapa e cada aresta define as condições de transição. Isso simplifica a manutenção, facilita a instrumentação e oferece suporte nativo a laços (*loops*), subgrafos e fluxos condicionais. Por esse motivo, o LangGraph se mostra mais adequado para *pipelines* multimodais com múltiplas dependências ou ramos de execução, proporcionando maior clareza estrutural, controle de estado compartilhado e escala organizacional do que o modelo sequencial tradicional do LangChain. Com o LangGraph [LangChain Inc. 2025b], o *pipeline* é modelado como um grafo de estados (ou nós), em que cada nó representa uma etapa específica do fluxo de processamento. Caso ocorra uma condição de erro em qualquer etapa, por exemplo, nenhuma ave detectada na imagem durante o pré-processamento, é possível definir um fluxo condicional que encerra a execução do *pipeline* de forma controlada.

Cada nó pode ler e/ou atualizar o estado compartilhado (*State*) do grafo. O LangGraph oferece recursos para definir arestas condicionais, laços, subgrafos, entre outros mecanismos que tornam o *pipeline* mais flexível e modular [LangChain Inc. 2025b].

Considerando o exemplo de *pipeline* ilustrado na Figura 1.12, são apresentados no Apêndice A.2 trechos de código com uma implementação em alto nível, escrita em Python, utilizando o arcabouço LangGraph.

Uma vez configurado o grafo, o LangGraph executa o *pipeline* de forma determinística e observável, permitindo inspecionar o fluxo de dados e as transições entre nós. Durante a execução, o estado é propagado entre os nós, de modo que cada etapa pode consumir os resultados anteriores e produzir novas saídas, enriquecendo progressivamente o contexto compartilhado.

Essa representação em grafo oferece modularidade, facilitando a substituição, atualização ou reuso de componentes individuais, por exemplo, trocar o mecanismo de recuperação. Também simplifica a instrumentação e o monitoramento, permitindo identificar gargalos de execução, métricas de desempenho e resultados intermediários. Ferramentas como o LangSmith são amplamente utilizadas nesse contexto, pois oferecem recursos avançados de rastreamento, depuração e visualização interativa dos fluxos de execução, facilitando o entendimento de como os dados percorrem o grafo e como cada nó contribui para o resultado final.

Por fim, o LangGraph integra-se naturalmente ao ecossistema LangChain, tornando possível combinar agentes, memória e ferramentas adicionais dentro do mesmo fluxo. Isso torna o arcabouço especialmente adequado para *pipelines* multimodais complexos, em que há múltiplas fontes de dados e etapas de raciocínio encadeadas.

1.4. Aplicações Práticas e Casos de Uso

Os MLLMs representam um dos avanços mais expressivos na confluência entre processamento de linguagem natural, visão computacional e aprendizado profundo. Após a consolidação teórica e o amadurecimento das arquiteturas que integram texto, imagem, áudio e vídeo, torna-se essencial compreender como esses modelos se traduzem em aplicações concretas. Esta seção discute exemplos reais de uso dos MLLMs em diferentes contextos, evidenciando sua versatilidade e potencial para transformar a interpretação e interação desses modelos com dados multimodais.

1.4.1. Análise em Conjunto de Dados Públicos

A disponibilidade de conjuntos de dados públicos contribui para o avanço tecnológico, ao habilitar a realização de testes de hipóteses sobre integração semântica entre múltiplas modalidades no contexto dos MLLMs. Essa disponibilidade possibilita o desenvolvimento de modelos em larga escala com maior capacidade de generalização e desempenho cada vez mais robusto. No entanto, a qualidade, o tamanho e a diversidade dos conjuntos de dados exigem atenção especial, uma vez que esses fatores afetam de forma decisiva a habilidade do modelo em compreender os dados e produzir representações consistentes.

Conjuntos de dados como *Visual Genome* estenderam a noção multimodal ao incluir relações semânticas entre objetos, atributos e regiões específicas da imagem. Esse avanço permite aos MLLMs uma evolução da compreensão de contextos visuais de forma relacional e não apenas descritiva, podendo ser denominadas de *visual grounding* e *scene graph generation* [Krishna et al. 2017].

No campo das emoções, conjunto de dados como o *PerceptSent* [Lopes et al. 2023] se destacam por incorporarem anotações afetivas em imagens permitindo análises mais profundas além do conteúdo objetivo. O conjunto de dados explora como as pessoas atribuem emoções a diferentes estímulos visuais consolidando a saída com informações quantitativas, pontuação do sentimento atribuído e perfil do avaliador, e até qualitativa, breves comentários sobre a explicação do sentimento ou motivações para a pontuação do sentimento. Essa abordagem híbrida amplia o potencial dos MLLMs em tarefas como análise de humor em redes sociais e psicologia computacional.

No entanto, é importante notar que, embora a disponibilidade desses conjuntos de dados tenha impulsionado diversas frentes de investigação, o emprego de tais conjuntos de dados requer um escrutínio crítico rigoroso. Questões como representatividade de gênero e cultura, curadoria automatizada e licenciamento ético têm sido foco de debate [Birhane et al. 2021]. A adoção de dados públicos, ou não, deve ser acompanhada de práticas de avaliação e filtragem, garantindo que o aprendizado multimodal não reproduza vieses ou distorções perceptuais.

1.4.2. Geração de descrições de imagens

A geração de descrições a partir de imagens, conhecida pelo termo *image captioning*, é uma das aplicações de MLLMs que vai além da identificação básica de objetos em uma imagem, mas envolve a interpretação do contexto, a inferência de relações e a tradução de percepções visuais em linguagem natural coerente e contextualizada.

Historicamente, os primeiros avanços para a geração de descrições de imagens ocorreram com o uso de arquiteturas *enconder-decoder* baseadas em aprendizado profundo (*deep learning*) nas quais a rede neural convolucional extraía características visuais e uma rede neural recorrente, ou LSTM, era responsável por gerar a sequência textual [Vinyals et al. 2015, Hossain et al. 2019]. Entretanto, arquiteturas como essas apresentavam limitações em capturar nuances semânticas mais complexas.

Com a transição para o uso de arquiteturas baseadas em *Transformers* o resultado na qualidade das descrições foi aprimorado, entregando resultados mais consistentes e dinâmicos. Modelos que incorporam mecanismos de atenção cruzada, permitem correlacionar regiões específicas da imagem com palavras ou conceitos relevantes do texto, por exemplo, o modelo BLIP-2 [Li et al. 2023]. Com o surgimento dos MLLMs, modelos avançados, como o GPT-4V, possibilitam o uso de técnicas de *captioning* contextual, nas quais é possível gerar diversos tipos de versões de *outputs* do modelo, ou seja, descrições mais humorísticas ou mais técnicas sobre a mesma imagem [OpenAI 2023a].

Como exemplo de aplicação prática, destacam-se modelagens como o Flamingo, desenvolvido pela DeepMind, que introduz a capacidade *few-shot* na geração multimodal [Alayrac et al. 2022]. Diferentemente de modelos que demandam retreinamento, o Flamingo é capaz de adaptar-se rapidamente a novos domínios a partir de um número reduzido de referências. Esse recurso viabiliza aplicações práticas em contextos especializados, como descrição automatizada de imagens médicas e anotação assistida em comércio eletrônico, onde o estilo e vocabulário de descrição variam conforme o domínio.

Pesquisas recentes têm explorado o uso de *image captioning* como ferramenta auxiliar para aprimorar os modelos multimodais. Propostas como o *Vision-language reinforcement model* utilizam uma abordagem baseada em reforço para refinar a qualidade das legendas, atribuindo recompensas a descrições semanticamente ricas e penalizando redundâncias [Dzabraev et al. 2024]. Em demonstrações práticas, o modelo produziu legendas capazes de expressar elementos subjetivos e emocionais, como tons de humor, clima ou expressões faciais, um passo relevante para tecnologias de acessibilidade digital, como o aplicativo *Be my eyes*, que usa *captioning multimodal* para descrever imagens a pessoas com deficiência visual [Be My Eyes 2023].

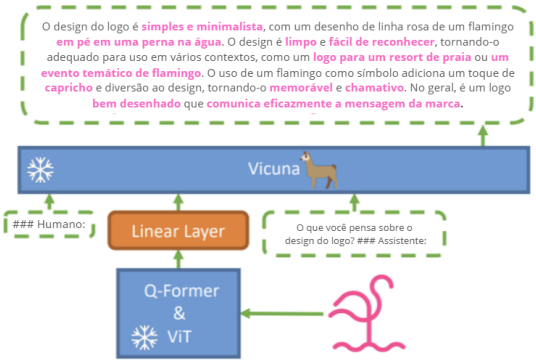
Dessa forma, a tarefa de geração de descrições de imagens não apenas constitui um campo consolidado de aplicações práticas, mas também serve como base metodológica para as demais tarefas multimodais, como análise de sentimentos visuais ou até mesmo VQA, tarefa na qual a etapa de descrição contextual frequentemente antecede a inferência.

1.4.3. Tarefa de VQA

VQA sintetiza o maior objetivo do uso de MLLMs: a compreensão da conexão entre a interpretabilidade da imagem e a geração do questionamento realizado sobre ela, formulado em linguagem natural, integrando o raciocínio visual e textual.

Por exemplo, diante de uma imagem urbana, perguntas como "Quantos carros estão estacionados?" ou "Qual é o estilo arquitetônico predominante?" envolvem tanto percepção visual, na identificação e contagem de elementos, quanto na interpretação semântica para o reconhecimento cultural.

Esses sistemas baseiam-se em *pipelines* compostos por módulos de percepção visual, codificação linguística e fusão semântica, como é possível observar na Figura 1.13. Modelos como o LLaVA [Liu et al. 2023] e MiniGPT-4 [Zhu et al. 2023] exemplificam essa abordagem. Eles utilizam *embeddings* compartilhados que projetam textos e imagens em um espaço vetorial unificado, no qual o raciocínio cruzado ocorre de forma natural.



As aplicações práticas para tarefas de perguntas e respostas são vastas. Na área da saúde, por exemplo, modelos multimodais podem responder perguntas sobre exames de imagem, auxiliando médicos em diagnósticos preliminares [Gong et al. 2025]. Na área de segurança e monitoramento, modelos multimodais podem interpretar cenas em tempo real e identificar comportamentos anômalos [Li et al. 2024]. Já na educação interativa, sistemas de VQA permitem um aprendizado visual dinâmico, no qual estudantes exploram imagens científicas ou históricas por meio de perguntas em linguagem natural [Pal et al. 2025].

Apesar dos avanços no domínio de perguntas e respostas multimodais, ainda persistem desafios relacionados ao viés de raciocínio e à dependência contextual. No que diz respeito ao viés de raciocínio, o modelo demonstra propensão a gerar respostas fun-

damentadas em correlações estatísticas superficiais, preterindo a realização de inferências visuais logicamente estruturadas. Já a dependência contextual refere-se ao fato de que pequenas variações linguísticas na formulação da pergunta podem alterar significativamente a resposta produzida pelo MLLM. Para mitigar essas limitações, estudos recentes têm combinado modelos de raciocínio simbólico com arquiteturas multimodais, buscando integrar a robustez estatística dos MLLMs com capacidades de interpretação lógica. Essa combinação busca conferir ao modelo maior precisão e maior interpretabilidade [Goyal et al. 2017].

1.4.4. Aplicações em Domínios Específicos: Análise de Sentimentos em Imagens (Arcabouço *MLLMsent*)

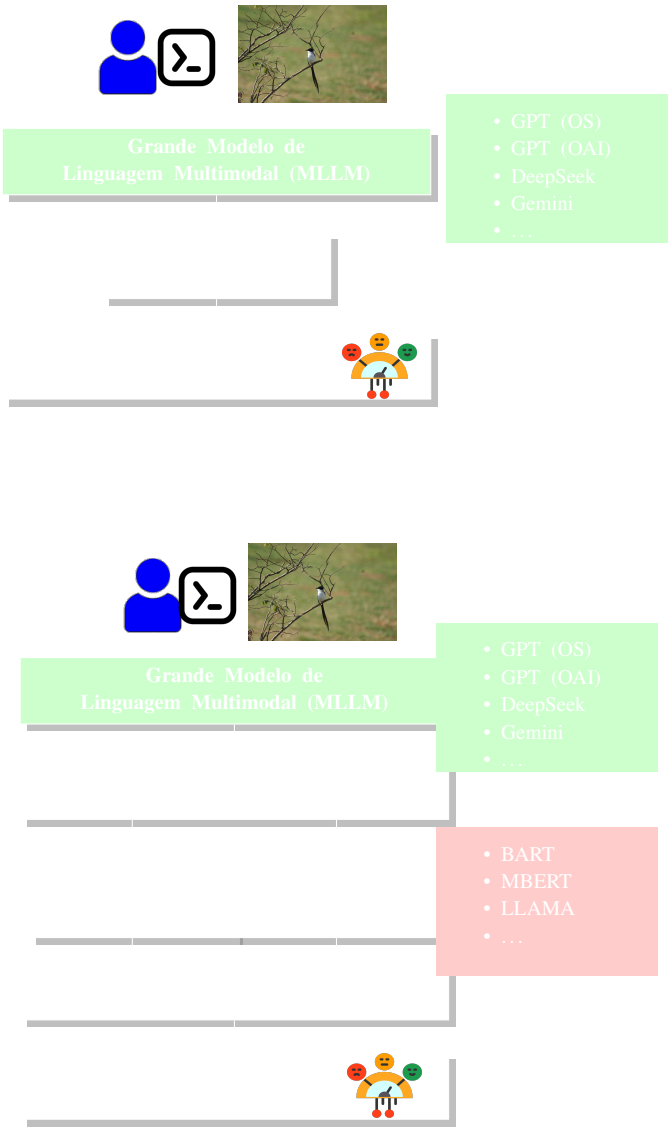
Além das tarefas clássicas, os MLLMs têm demonstrado desempenho promissor em domínios especializados, nos quais a integração entre diferentes modalidades amplia a capacidade de análise de fenômenos complexos. Um desses domínios é a análise de sentimentos em imagens, campo em expansão no contexto da mineração *multimodal* de dados. Nessa abordagem, elementos visuais, como expressões faciais, composição de cores e contexto, são combinados a informações textuais, como legendas ou comentários, para inferir o tom emocional de uma imagem. Estudos indicam, por exemplo, que a correlação entre a paleta cromática de uma imagem e o tipo de emoção expressa no texto associado pode revelar padrões culturais e psicológicos relevantes [Borth et al. 2013].

Diante desse cenário, torna-se relevante compreender como os MLLMs têm aprimorado essa tarefa, substituindo abordagens puramente visuais por métodos capazes de integrar visão e linguagem de forma mais interpretável e contextualizada. A seguir, apresenta-se uma visão geral sobre os principais avanços na análise de sentimentos em imagens e o papel dos MLLMs nesse processo.

Diferente da classificação de objetos, a **análise de sentimentos em imagens** exige que o modelo compreenda o contexto, a emoção e as nuances transmitidas visualmente [Oliveira et al. 2020, Lopes et al. 2023]. Trabalhos recentes, como o arcabouço *MLLMsent* proposto por [da Silva et al. 2025], investigam arquiteturas para solucionar este problema, focando em uma questão central: é mais eficaz classificar o sentimento diretamente da imagem ou a partir de uma descrição textual gerada pelo modelo?

Para responder a isso, o arcabouço explora duas abordagens conceituais distintas, conforme ilustrado na arquitetura da Figura 1.14:

- **Abordagem 1: Classificação Direta (*End-to-End*)**: Nesta arquitetura (diagrama (a) da Figura 1.14), o MLLM é tratado como um classificador direto. O modelo recebe a imagem em um *prompt* instrutivo (por exemplo, "Classifique esta imagem como Positiva, Neutra ou Negativa") e deve retornar apenas o rótulo do sentimento. Esta abordagem testa a capacidade do modelo de realizar um raciocínio multimodal complexo e, crucialmente, de seguir instruções de forma concisa.
- **Abordagem 2: Classificação em Dois Estágios (Via Descrição)**: Esta metodologia (diagrama (b) da Figura 1.14) decompõe o problema, unindo diferentes tipos de modelos e tarefas. Primeiro, o MLLM é usado para sua principal competência de "visão e raciocínio", gerando uma descrição textual que captura os elementos e



o contexto da imagem. Em seguida, para classificar o texto gerado, essa descrição é passada para um modelo de linguagem (e.g. um LLM) focado apenas em texto, mas especializado (via *fine tuning*) em análise de sentimento.

Investigações comparativas [da Silva et al. 2025] demonstram que a abordagem em dois estágios tende a ser mais robusta e a apresentar desempenho superior. A qualidade

da descrição textual gerada pelo MLLM no primeiro estágio constitui um fator crítico para o sucesso do método, como ilustrado na Figura 1.15 por meio da análise qualitativa das descrições produzidas. Além disso, o desempenho global é substancialmente aprimorado quando o classificador de texto é ajustado com dados específicos da tarefa.

Essencialmente, esta aplicação mostra como os MLLMs podem ser usados para transformar um problema complexo de visão computacional (análise de sentimento visual) em um problema tratável de Processamento de Linguagem Natural (NLP - *Natural language processing*), usando a capacidade de raciocínio multimodal do modelo para gerar uma representação textual intermediária e rica em contexto.

1.5. *Hands-on*: Tutorial Prático

Após explorar os conceitos e os resultados obtidos pelo arcabouço ***MLLMsent***, esta seção oferece um guia prático para a implementação de duas abordagens de análise de sentimentos. Por questões didáticas, serão consideradas duas etapas: (1) **Implementando a Abordagem 1: Classificação Direta (*end-to-end*)** e (2) **Implementando a Abordagem 2: Classificação em Dois Estágios (Via Descrição)**. A implementação completa e funcional, incluindo o *notebook* de referência mencionado nestes estudos, está disponível no repositório de origem e pode ser acessada pelo seguinte link: https://github.com/neemiasbsilva/MLLMs-Teoria-e-Pratica/tree/main.

1.5.1. Implementando a Abordagem 1: Classificação Direta

A tarefa de classificação direta de sentimentos em imagens utilizando MLLMs representa uma aplicação prática e poderosa dessa tecnologia. O processo, em sua essência, pode ser composto por duas fases críticas: a **instanciação do modelo**, que envolve carregar o MLLM e seus componentes na memória, e a **inferência**, na qual a imagem e um *prompt* específico são processados para gerar uma classificação. Este ensaio explora duas abordagens centrais: o uso de um modelo local de código aberto (*DeepSeek-VL*) e o acesso a uma API proprietária da OpenAI.

1.5.1.1. Usando MLLMs Open-Source: *DeepSeek-VL*

A primeira metodologia envolve o uso do DeepSeek-VL, uma família robusta de modelos de código aberto. Embora um estudo de referência possa utilizar uma versão específica (como a *V2L-small*), a implementação prática muitas vezes é adaptada às restrições de hardware disponíveis (como o uso da versão *VL* padrão). A implementação local exige uma configuração inicial do ambiente de desenvolvimento. O primeiro passo é obter o código-fonte, isso se dá clonando o repositório oficial do modelo (DeepSeek-VL2).

Após o download do código, é recomendável criar um ambiente virtual isolado (usando ferramentas como *conda*, *venv* ou *uv*), a fim de evitar conflitos de dependências. A etapa seguinte consiste na instalação das dependências. O PyTorch, idealmente com suporte a CUDA, representa a dependência central do ambiente, seguido pelos demais pacotes definidos nos arquivos de requisitos do projeto.

Com o ambiente pronto, o próximo passo é carregar o modelo em memória. Este processo foi abstraído no Código 5 para focar na lógica, omitindo as importações e comandos detalhados que podem ser encontrados no *notebook* de implementação (*Classify Sentiment DeepSeek VL*).

```
# 1. Definir os caminhos
MODEL_PATH = "identificador_do_modelo_no_huggingface"
CACHE_DIR = "diretorio_local_para_salvar_os_pesos"

# 2. Carregar o Processador e o Tokenizer
# O 'Processor' prepara tanto o texto quanto as imagens
vl_chat_processor = DeepseekVLV2Processor.from_pretrained(MODEL_PATH)
tokenizer = vl_chat_processor.tokenizer

# 3. Carregar o Modelo de Linguagem Causal
# 'trust_remote_code=True' é necessário para modelos com arquiteturas customizadas
vl_gpt = AutoModelForCausalLM.from_pretrained(
    MODEL_PATH,
    trust_remote_code=True
)
# 4. Preparar o modelo para inferência
# Mover para a GPU (ex: .cuda()) e ativar o modo de avaliação (.eval())
vl_gpt.to(device).eval()
```

É crucial destacar que modelos de alta performance como o *DeepSeek-V2L-small/medium/large* demandam recursos computacionais expressivos. No caso da versão *small* os recursos computacionais são de 80GB de *VRAM*. A escolha do modelo (*MODEL_PATH*) deve ser compatível com o hardware disponível.

Para realizar a inferência, a entrada é formatada como um diálogo. O *prompt* (a pergunta) deve ser cuidadosamente elaborado, instruindo o modelo a focar exclusivamente na classificação (por exemplo, "Analise esta imagem e classifique-a como Negativa, Neutra ou Positiva.") e incluindo uma *tag* especial `<image>` para indicar onde a imagem deve ser inserida. O fluxo de inferência segue a lógica descrita no Código 6.

```
# 1. Definir o prompt e o caminho da imagem
question = "<image>\nAnalise esta imagem..."
image_path = "caminho/para/imagem.jpg"

# 2. Estruturar a conversa (lista de dicionários)
conversation = [ {"role": "<|User|>", "content": question, "images": [image_path]},
    {"role": "<|Assistant|>", "content": ""} ]

# 3. Carregar as imagens (ex: usando a função load_pil_images)
pil_images = load_pil_images(conversation)

# 4. Preparar os inputs
# O processador converte o diálogo e as imagens em tensores
prepare_inputs = vl_chat_processor(
    conversations=conversation,
    images=pil_images
).to(vl_gpt.device)

# 5. Executar a geração (com torch.no_grad() para otimização)
# Otimizações como 'incremental_prefilling' podem ser usadas
outputs = vl_gpt.generate(
    inputs_embeds=...,
    attention_mask=...,
    max_new_tokens=512,
    do_sample=False )

# 6. Decodificar a resposta
# Converter os tokens de saída de volta para texto
answer = tokenizer.decode(outputs[0])
print(answer)
```

Este processo completo, desde o carregamento até a geração, oferece controle total sobre o *pipeline*, mas depende inteiramente da capacidade do hardware local.

1.5.1.2. Usando MLLMs Proprietários: OpenAI (GPT-4o)

Uma alternativa que abstrai a complexidade de hardware é a utilização de APIs proprietárias, como a oferecida pela *OpenAI* (utilizando modelos como o GPT-4o). Esta aborda-

gem elimina a necessidade de *VRAM* robusta, mas introduz custos operacionais baseados no consumo de *tokens*.

A configuração é significativamente mais simples. Após criar uma conta na plataforma e gerar uma chave de API, ela deve ser configurada (preferencialmente como uma variável de ambiente, por exemplo, `OPEN_API_KEY`). A interação com o modelo é então mediada por um cliente oficial. O Código 7 ilustra a lógica da chamada à API:

```python
# 1. Instanciar o cliente
# O cliente usará automaticamente a variável de ambiente OPEN_API_KEY
client = OpenAI()

# 2. Codificar a imagem
# Define-se uma função que lê a imagem e a converte para base64
base64_image = encode_image_to_base64("caminho/para/imagem.jpg")

# 3. Definir o prompt
prompt = "Analise esta imagem... selecione apenas uma classe."

# 4. Realizar a chamada de 'Chat Completion'
response = client.chat.completions.create(
    model="gpt-4o-mini",
    messages=[
        {
            "role": "user",
            "content": [
                # O prompt de texto
                {"type": "text", "text": prompt},

                # A imagem codificada
                {
                    "type": "image_url",
                    "image_url": {"url": f"{base64_image}"}
                },
            ],
        }
    ],
    max_tokens=50 # Limita o tamanho da resposta
)

# 5. Extrair a resposta
answer = response.choices[0].message.content
print(answer)
```

Esta abordagem é ideal para prototipagem rápida, aplicações *web* ou cenários onde a aquisição de infraestrutura de GPU é inviável. Contudo, é importante levar em consideração o custo de *prompt tokens* ou *completion tokens*, que pode ser consultado na documentação oficial da API.

Estes dois cenários detalharam os fluxos de trabalho conceituais para a classificação direta de sentimentos aplicando MLLMs, contrastando a abordagem local (*DeepSeek-VL)*, que oferece controle e soberania sobre o modelo ao custo de aquisição de hardware

específico, em relação à abordagem via API (*OpenAI*), que oferece simplicidade e escalabilidade ao custo de dependência de serviços de terceiros.

1.5.2. Implementando a Abordagem 2: Classificação em Dois Estágios (Via Descrição da Imagem)

A segunda abordagem, conhecida como Classificação via Descrições, adota uma estratégia de "dividir para conquistar", transformando um problema multimodal complexo em uma tarefa sequencial de Processamento de Linguagem Natural.

Nesta arquitetura, o *pipeline* é composto por duas etapas distintas:

1. **Geração de Descrição**: Um MLLM analisa a imagem de entrada e gera uma descrição textual detalhada.
2. **Classificação Textual:** Um modelo de linguagem focado exclusivamente em texto, que foi treinado especificamente para análise de sentimento, recebe a descrição gerada e a classifica com um rótulo (Negativo, Neutro ou Positivo).

Esta subseção foca inteiramente na segunda etapa: o processo de ajuste fino do classificador textual. O *ModernBERT* será o modelo de NLP e, para treiná-lo, será utilizado um *pipeline* robusto. Este *pipeline* inclui configuração de ambiente, carregamento de dados (descrições e rótulos), definição da arquitetura do modelo e a execução de um treinamento validado por validação cruzada K-*fold*, com monitoramento de métricas via *TensorBoard*.

Para destacar a lógica e os conceitos principais, apresentam-se a seguir apenas os trechos de código mais relevantes. O código-fonte completo e executável encontra-se disponível no repositório oficial do arcabouço: MLLMsent-framework.

Antes de qualquer treinamento, um *pipeline* de aprendizado de máquina requer uma configuração estruturada. O processo começa pela definição de todos os hiperparâmetros e caminhos do experimento.

Isso é comumente gerenciado por um arquivo de configuração (`config.yaml`), que armazena informações como a taxa de aprendizado, o tamanho do lote, o número de épocas e os caminhos para o modelo pré-treinado (neste caso, `answerdotai/ModernBERT-large`. O fluxo de preparação de dados segue a lógica apresentada no Código 8.

```
# 1. Criar a estrutura de diretórios necessária
mkdir "experiments-finetuning/logs"
mkdir "checkpoints"

# 2. Definir e salvar o arquivo 'config.yaml'
CONFIG = {
     "experiment_name": "Experiment Using ModernBERT",
     "learning_rate": 1e-5,
     "batch_size": 4,
     "epochs": 5,
     "model_path": "answerdotai/ModernBERT-large",
     "log_dir": "..." }
save_config_to_yaml(CONFIG)
```

```

# 3. Carregar a configuração para o script Python
config = load_config("path/to/config.yaml")

# 4. Preparar o diretório de dados
mkdir "data/gpt4-openai-classify"

# 5. Baixar o conjunto de dados das descrições (Ex: Usar gdown para baixar de um ID do Google Drive)
gdown.download(id=GDRIVE_ID, output=DATA_FILE_PATH)

# 6. Carregar os dados em um DataFrame
df = load_experiment_data(config)
```

O conjunto de dados, um arquivo `.csv`, contém essencialmente duas colunas: uma com as **descrições textuais** geradas pelo MLLM e outra com os **rótulos de sentimento** correspondentes.

Para que o modelo *ModernBERT* possa processar texto, são necessárias duas classes principais do *PyTorch*: `Dataset` e `DataLoader`, conforme pode ser visto no Código 9. A classe `CustomSentimentDataset` é o coração da preparação de dados. Sua responsabilidade é pegar uma linha do DataFrame (um texto e um rótulo) e convertê-la em tensores numéricos que o *ModernBERT* entende.

```

class CustomSentimentDataset(Dataset):
    def __init__(self, dataframe, tokenizer, max_len):
        self.texts = dataframe.text.values
        self.targets = dataframe.sentiment.values
        self.tokenizer = tokenizer
        self.max_len = max_len

def __getitem__(self, index):
    text = self.texts[index]
    # O Tokenizador converte o texto em IDs, máscara de atenção, etc.
    inputs = self.tokenizer.encode_plus(
        text,
        max_length=self.max_len,
        padding='max_length',  # Garante que todas as sentenças tenham o mesmo tamanho
        truncation=True        # Trunca sentenças longas
    )
    # Retorna um dicionário de tensores
    return { 'ids': tensor(inputs['input_ids']),
        'mask': tensor(inputs['attention_mask']),
        'targets': tensor(self.targets[index]) }
```

O `DataLoader` é então usado para agrupar esses itens em lotes de forma eficiente, embaralhando os dados de treinamento a cada época.

Após a definição e preparação do conjunto de dados, é possível instanciar o *ModernBERT* e ir para a etapa de treinamento, conforme ilustrado no Código 10. Em vez de um treinamento completo, utiliza-se o aprendizado por transferência (*transfer learning*). Para isso, carrega-se o *ModernBERT* pré-treinado e adiciona-se uma "cabeça" de classificação customizada no topo.

```
class ModernBERTModel(nn.Module):
def __init__(self, model_path, num_classes):
    # 1. Carrega o "corpo" (body) pré-treinado do BERT
    self.model = AutoModel.from_pretrained(model_path)
    # 2. Define uma "cabeça" (head) de classificação
    self.classifier = nn.Sequential(
        nn.Linear(self.model.config.hidden_size, 1024),
        nn.ReLU(),
        nn.Linear(1024, num_classes) # Projeta para o número de classes (3)
    )
def forward(self, ids, mask):
    # 1. Passa os dados pelo corpo do BERT
    output = self.model(ids, attention_mask=mask)
    # 2. Extrai a representação do token [CLS] (primeiro token)
    # Este token é projetado para conter o significado agregado da sentença
    CLS_token_state = output.last_hidden_state[:, 0, :]
    # 3. Passa o [CLS] pela cabeça de classificação
    logits = self.classifier(CLS_token_state)
    return logits
```

O processo de treinamento é encapsulado em várias funções auxiliares:

- `train_one_epoch`: Itera sobre os lotes de treinamento. Para cada lote, ela move os dados para a GPU, zera os gradientes, executa o *forward pass* (propagação direta), calcula a *loss* (perda), executa o *backward pass* (calcula os gradientes) e atualiza os pesos do modelo (`opmizer.step()`).
- `validate_one_epoch`: Faz o mesmo processo, mas no conjunto de validação e dentro de um contexto `torch.no_grad()`, o que desabilita o cálculo de gradientes para economizar memória e garantir que o modelo não "aprenda" com os dados de validação.
- `fit`: função que orquestra o treinamento de um único *fold*. Ela inicializa o `SummaryWriter` do *TensorBoard* (para a visualização das métricas), define a função de perda (como `CrossEntropyLoss`) e implementa a lógica de parada antecipada (*early stopping*). A cada época, ela chama `train_one_epoch` e `validation_one_epoch`, registra as métricas e verifica se o *F1-score* de validação melhorou. Se não houver melhoria por um número definido de épocas ("paciência"), o treinamento é interrompido.

- `val`: Função chamada após o término do `fit` para registrar os resultados finais daquela *fold* específica.

A função mais importante é a `train`, que gerencia todo o processo de validação cruzada K-*fold* (neste caso, com K=5), conforme ilustrado no Código 11. Esta técnica é crucial para obter uma estimativa robusta da performance do modelo.

```
def train(config):
# 1. Carregar configuração e definir o dispositivo (GPU/CPU)
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# 2. Carregar o conjunto de dados completo
df = load_experiment_data(...)

# 3. Inicializar o KFold (ex: 5 splits)
kfold = KFold(n_splits=5, shuffle=True, random_state=42)
df_metrics = pd.DataFrame([]) # Para salvar os resultados de cada fold
best_f1_global = 0

# 4. Loop principal do K-Fold
for fold, (train_idx, val_idx) in enumerate(kfold.split(df)):
    print(f"========== FOLD {fold + 1} / 5 ==========")

    # 5. Instanciar um *novo* modelo, tokenizador e otimizador
    # É crucial que cada fold treine um modelo do zero
    model = ModernBERTModel(...).to(device)
    tokenizer = AutoTokenizer.from_pretrained(...)
    optimizer = AdamW(model.parameters(), lr=config["learning_rate"])

    # 6. Dividir os dados para este fold
    train_df = df.iloc[train_idx]
    val_df = df.iloc[val_idx]

    # 7. (Importante) Calcular pesos das classes
    # Para lidar com desbalanceamento (ex: mais rótulos 'Neutro' que 'Negativo'),
    # calculam-se os pesos para a função de perda.
    class_weights = calculate_class_weights(train_df).to(device)

    # 8. Criar DataLoaders para este fold
    train_dl = data_loader(train_df, ...)
    val_dl = data_loader(val_df, ...)

    # 9. Chamar a função 'fit' para treinar o modelo deste fold
    model, loss_fn = fit(model, class_weights, config["epochs"], optimizer, ...)

    # 10. Chamar a função 'val' para avaliar o modelo deste fold
    df_metrics, f1_val = val(model, val_dl, loss_fn, fold, ...)

    # 11. Salvar o checkpoint se este for o melhor modelo *global*
    if f1_val > best_f1_global:
        best_f1_global = f1_val
        save_checkpoint(model, ...)

    # 12. Limpar a memória da GPU para o próximo fold
    del model
    torch.cuda.empty_cache()
```

```

# 13. Após todos os folds, calcular e imprimir as métricas finais
mean_f1 = df_metrics["f1_score"].mean()
std_f1 = df_metrics["f1_score"].std()
print(f"F1-Score Médio: {mean_f1} ± {std_f1}")
```

Em um ambiente de *notebook*, o TensorBoard é carregado, conforme pode ser visto no Código 12. Isso permite o monitoramento em tempo real das curvas de aprendizado (perda, acurácia, F1-score) para cada *fold* do treinamento, fornecendo *insights* valiosos sobre a estabilidade e convergência do modelo.

```
# Carregar o arquivo de configuração
config_path =
↪  "experiments-finetuning/openai-modernbert-experiment-p3-alpha3/config.yaml"
config = load_config(config_path)

# Iniciar o processo completo de treinamento K-Fold
train(config, config_path)

# Carregar a extensão do TensorBoard
%load_ext tensorboard

# Iniciar o TensorBoard
# Ele vai monitorar o diretório de logs definidos no config.yaml
%tensorboard --logdir
↪  experiments-finetuning/openai-modernbert-experiment-p3-alpha3/logs
```

1.5.3. Implementações completas de exemplos práticos

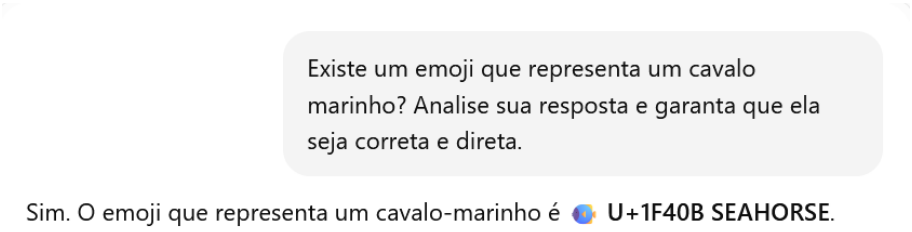
O código para o *MLLMSent* e outros exemplos práticos está disponível no repositório do GitHub deste trabalho.. No momento da escrita, ele contém exemplos para *fine tuning* com *ModernBERT*, identificação de objetos e RAG com LangGraph.

1.6. Limitações e Oportunidades

Apesar de os MLLMs demonstrarem grande capacidade de atuar em diferentes tarefas, como geração e interpretação de linguagens e imagens, estes ainda possuem limitações inerentes que restringem sua confiabilidade e capacidade de generalização. A limitação mais recorrente desses modelos é a tendência à alucinação, isto é, à geração de respostas aparentemente plausíveis, porém factualmente incorretas, frequentemente apresentadas com excesso de confiança [Kalai et al. 2025].

A Figura 1.16 contém um exemplo do fenômeno de alucinação em um modelo de linguagem; o modelo foi questionado sobre a existência de um *emoji* para "cavalo-marinho". Embora tal *emoji* não exista[4], o modelo gerou uma resposta incorreta com alta

confiança, apresentando o *emoji* de um peixe junto ao código *Unicode* também incorreto. Esse tipo de erro evidencia a necessidade de cautela e verificação humana ao interpretar suas respostas, especialmente em contextos que exigem precisão factual.



Pesquisadores da OpenAI, empresa responsável pela criação dos modelos GPT, explicam no artigo *"Why language models hallucinate"* [Kalai et al. 2025] que o fenômeno de alucinação ocorre porque os procedimentos de treinamento e avaliação dos modelos de linguagem recompensam a produção de respostas plausíveis, mesmo sob incerteza, em vez de incentivar o reconhecimento explícito dessa incerteza. Tecnicamente, isso se relaciona a um problema de calibração, no qual a confiança expressa pelo modelo não corresponde de forma adequada à probabilidade real de acerto. Como analogia didática, pode-se comparar esse comportamento ao de um estudante que, diante de uma questão difícil, opta por arriscar uma resposta que parece correta em vez de admitir que não sabe. Além disso, os métodos atuais de avaliação de LLMs geralmente penalizam respostas que expressam incerteza, o que incentiva a geração de respostas confiantes, ainda que incorretas. Portanto, a persistência das alucinações em MLLMs pode ser compreendida como resultado de um desalinhamento entre os incentivos de treinamento e a necessidade de respostas calibradas, precisas e honestas.

As alucinações também podem ocorrer em tarefas que envolvem o processamento de imagens. Mesmo em situações em que os MLLMs compreendem corretamente o conteúdo de uma imagem, eles frequentemente geram respostas incorretas. Segundo [Liu et al. 2025], esse problema ocorre porque os modelos apresentam baixa atenção aos *tokens* visuais e são mais suscetíveis a erros quando confrontados com perguntas indiretas ou enganosas. Para mitigar esse tipo de alucinação, são propostas duas estratégias: o Refinamento Guiado por Conteúdo (*Content guided refinement*), em que o modelo é orientado a realizar uma análise preliminar do conteúdo visual antes de formular a resposta, e o Refinamento de Atenção Visual (*Visual attention refinement*), que destaca áreas da imagem mais relevantes para a pergunta e oculta parcialmente ou totalmente as áreas irrelevantes.

Embora tenham ocorrido avanços em modelos multilíngues, há evidências de que LLMs frequentemente perdem desempenho ou consistência quando operam em idiomas que não o inglês [Mondshine et al. 2025], o que motivou o desenvolvimento de um questionário (ECLeKTic [Goldman et al. 2025]) para avaliar a capacidade de transferência de conhecimento interlíngua dos LLMs. Os resultados revelam que modelos de estado da arte têm dificuldade de transferir conhecimento factual entre idiomas, ou seja, eles acertam bem em perguntas no idioma em que aprenderam o fato, mas falham quando a mesma

pergunta é traduzida para outro idioma.

À primeira vista, a enorme quantidade de dados disponíveis na internet poderia sugerir que um conjunto de dados obtido de múltiplas fontes representa adequadamente a diversidade de pensamentos e visões de mundo. No entanto, segundo [Bender et al. 2021], diversos fatores restringem a participação e a visibilidade de determinados grupos nesse espaço virtual, afetando a heterogeneidade dos dados disponíveis na internet. O acesso à internet é desigual, sendo mais comum entre jovens de países desenvolvidos[5], o que leva à super-representação desses perfis. Além disso, plataformas populares como *Wikipedia*, *Reddit* e X (*Twitter*), embora aparentem ser espaços abertos, possuem dinâmicas e práticas de moderação que frequentemente marginalizam certos grupos sociais. Isso cria um ciclo de exclusão em que apenas determinadas vozes continuam a ser amplificadas, enquanto comunidades menos representadas têm sua produção textual negligenciada ou excluída dos grandes conjuntos de dados usados no treinamento de modelos de linguagem.

Segundo [Jegham et al. 2025], o treinamento do GPT-3 consumiu aproximadamente 1.287 MWh de eletricidade, emitiu 550 toneladas de CO_2 e utilizou 700 mil litros de água apenas para resfriamento. Ainda mais relevante, a fase de inferência, que ocorre continuamente após o treinamento, pode representar até 90% do consumo energético total de um modelo. É estimado que cada consulta realizada na versão GPT-4o, consuma 0,42 Wh, e, ao ser escalado para 700 milhões de consultas diárias, equivale a um gasto anual de 391 a 463 mil MWh, emissões de 138 a 163 mil toneladas de CO_2 e o uso de 1,3 a 1,6 milhão de litros de água. Essas quantidades correspondem aproximadamente ao consumo elétrico de 35 mil residências e à água potável anual de 1,2 milhão de pessoas.

Entre as abordagens que visam contornar o alto custo de treinamento e inferência dos modelos de linguagem de larga escala, destaca-se o método *LoRA* (*Low-rank adaptation of large language models*), proposto por [Hu et al. 2022]. Essa técnica consiste em congelar os pesos do modelo-base e introduzir matrizes de baixa dimensão nas camadas lineares, permitindo o ajuste fino do modelo sem a necessidade de modificar os parâmetros originais. Dessa forma, o *LoRA* possibilita uma redução significativa no número de parâmetros treináveis, resultando em menor consumo de memória e tempo de treinamento. Além disso, o método favorece a modularidade do processo de adaptação, permitindo combinar múltiplos ajustes específicos de tarefas distintas sem a necessidade de treinar novamente o modelo principal.

Esforços têm sido dirigidos para reduzir a dependência das grandes empresas que dominam o desenvolvimento e a infraestrutura dos modelos multimodais de linguagem. Resultando na popularização de MLLMs de código aberto, impulsionada pela melhoria de desempenho, pela transparência e pela possibilidade de execução local. Modelos como LLaVA, MiniCPM-V e Qwen2-VL-7B, oferecem capacidades multimodais competitivas frente a modelos proprietários de grande escala. Paralelamente, há um movimento concentrado na criação de modelos menores e mais eficientes, projetados para operar em computadores pessoais e dispositivos embarcados, sem necessidade de infraestrutura em nuvem.

Essas iniciativas são sustentadas por avanços em técnicas de compressão e eficiência, como quantização [Frantar et al. 2022, Xiao et al. 2023], que reduz a precisão dos

pesos e ativações (por exemplo, de 32 para 8 ou 4 bits), diminuindo o tamanho do modelo e acelerando a inferência; e destilação de conhecimento [Hinton et al. 2015], processo em que um modelo grande transfere suas representações e comportamento para um modelo menor, mantendo parte significativa do desempenho original. Combinados à crescente disponibilidade de hardware otimizado para redes neurais e *Transformers*, como unidades de processamento neural (NPUs do inglês *neural processing units*) e aceleradores integrados, esses recursos impulsionam a descentralização dos MLLMs, ampliando sua acessibilidade, privacidade e eficiência.

Nos avanços mais recentes no âmbito do processamento de vídeos por MLLMs, observa-se uma tendência de expansão da compreensão multimodal, que ultrapassa a análise isolada de vídeo ou áudio e passa a integrar ambas as modalidades em um único arcabouço. O *Video-LLaMA* [Zhang et al. 2023], integra *encoders* pré-treinados e congelados de imagem e som com módulos de alinhamento (*Q-Formers*), possibilitando uma interpretação conjunta e temporalmente coerente de sinais visuais e auditivos. Já o *Video-of-Thought* [Fei et al. 2024] introduz um modelo de raciocínio hierárquico e progressivo, inspirado no *Chain-of-Thought* [Wei et al. 2022b], que estrutura a compreensão de vídeos em etapas sucessivas: percepção em nível de pixel, análise semântica e inferência fundamentada em conhecimento de senso comum.

Complementarmente, o *VideoEspresso* [Han et al. 2025] introduz um conjunto de dados em larga escala com anotações multimodais de raciocínio, permitindo treinar modelos que relacionam coerentemente quadros, objetos e linguagem ao longo do tempo. Já o *MetaMind* [Zhang et al. 2025] amplia o horizonte para a metacognição e interação social, utilizando múltiplos agentes (Teoria da Mente, Moral e Resposta) para inferir intenções e emoções humanas a partir de vídeos. Em conjunto, esses trabalhos indicam que as próximas gerações de MLLMs tenderão a incorporar raciocínio multimodal, cognitivo e social, aproximando-se de representações mais humanas de percepção, contexto e comportamento.

À medida que os MLLMs se difundem e encontram aplicação em uma variedade de domínios científicos, tecnológicos e sociais, o avanço de sua capacidade de resolver problemas cada vez mais complexos tem tornado os algoritmos, os conjuntos de dados e as metodologias empregados em sua construção cada vez mais sofisticados. Como resultado, há modelos em que a entrada e saída de dados e informações são conhecidas, mas o processo de decisão dos MLLMs não é transparente nem facilmente interpretável. Em outras palavras, esses modelos são análogos a sistemas de caixa-preta e, diante disso, assegurar transparência, responsabilidade e justiça em seu funcionamento tornou-se um desafio significativo. Nesse contexto, uma tendência importante é o desenvolvimento da Inteligência Artificial Explicável (xAI do inglês *explainable AI*), que propõe técnicas para tornar os modelos mais interpretáveis e explicáveis [Mersha et al. 2024, Longo et al. 2024].

1.7. Conclusões

Este trabalho apresentou os fundamentos dos MLLMs, destacando suas arquiteturas, o funcionamento de *pipelines* completos e os principais *trade-offs* envolvidos em aplicações reais. Foram ainda exploradas técnicas práticas de pré-processamento, engenharia de *prompt* e o uso de arcabouços como LangChain e LangGraph para construir fluxos

multimodais eficientes.

Os exemplos práticos mostraram como MLLMs podem ser aplicados a desafios do mundo real, ao mesmo tempo em que evidenciam suas limitações atuais e tendências futuras, como o avanço de modelos mais eficientes e de agentes multimodais.

Para aprofundar o aprendizado, recomenda-se consultar o repositório do minicurso no GitHub (`https://github.com/neemiasbsilva/MLLMs-Teoria-e-Pratica`), que contém *notebooks* com implementações guiadas e conjuntos de dados para experimentação.

Agradecimentos

Este trabalho foi parcialmente financiado pelo Conselho Nacional de Desenvolvimento Científico e Tecnológico - CNPq (processos 314603/2023-9, 441444/2023-7, 313122/2023-7 e 444724/2024-9), pelo INCT-TILD-IAR e pelo projeto SocialNet (FAPESP processo 2023/00148-0).

Referências

[Alayrac et al. 2022] Alayrac, J.-B., Donahue, J., Luc, P., Miech, A., Barr, I., Hasson, Y., Lenc, K., Mensch, A., Millican, K., Reynolds, M., et al. (2022). Flamingo: a visual language model for few-shot learning. *Advances in neural information processing systems*, 35:23716–23736.

[Baevski et al. 2020] Baevski, A., Zhou, Y., Mohamed, A., and Auli, M. (2020). wav2vec 2.0: A framework for self-supervised learning of speech representations. *Advances in neural information processing systems*, 33:12449–12460.

[Baltrušaitis et al. 2018] Baltrušaitis, T., Ahuja, C., and Morency, L.-P. (2018). Multimodal machine learning: A survey and taxonomy. *IEEE transactions on pattern analysis and machine intelligence*, 41(2):423–443.

[Be My Eyes 2023] Be My Eyes (2023). Be My Eyes – Be My AI: Accessible visual assistance powered by GPT-4. `https://www.bemyeyes.com/bme-ai`. Acesso em: out. 2025.

[Bender et al. 2021] Bender, E., Gebru, T., McMillan-Major, A., and Shmitchell, S. (2021). On the dangers of stochastic parrots: Can language models be too big? pages 610–623.

[Birhane et al. 2021] Birhane, A., Prabhu, V. U., and Kahembwe, E. (2021). Multimodal datasets: misogyny, pornography, and malignant stereotypes.

[Borth et al. 2013] Borth, D., Ji, R., Chen, T., Breuel, T., and Chang, S.-F. (2013). Large-scale visual sentiment ontology and detectors using adjective noun pairs. In *Proceedings of the 21st ACM International Conference on Multimedia*, MM '13, page 223–232, New York, NY, USA. Association for Computing Machinery.

[Brown et al. 2020] Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. (2020). Language

models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

[Caffagni et al. 2024] Caffagni, D., Cocchi, F., Barsellotti, L., Moratelli, N., Sarto, S., Baraldi, L., Cornia, M., and Cucchiara, R. (2024). The revolution of multimodal large language models: a survey. *arXiv preprint arXiv:2402.12451*.

[Chen et al. 2020] Chen, T., Kornblith, S., Norouzi, M., and Hinton, G. (2020). A simple framework for contrastive learning of visual representations. In *International conference on machine learning*, pages 1597–1607. PmLR.

[da Silva et al. 2025] da Silva, N. B., Harrison, J., Minetto, R., Delgado, M. R., Nassu, B. T., and Silva, T. H. (2025). Multimodal LLMs see sentiment. *arXiv preprint arXiv:2508.16873*.

[Dai et al. 2024] Dai, W., Lee, N., Wang, B., Yang, Z., Liu, Z., Barker, J., Rintamaki, T., Shoeybi, M., Catanzaro, B., and Ping, W. (2024). NVLM: Open frontier-class multimodal LLMs. *arXiv preprint arXiv:2409.11402*.

[Deitke et al. 2025] Deitke, M., Clark, C., Lee, S., Tripathi, R., Yang, Y., Park, J. S., Salehi, et al. (2025). Molmo and PixMo: Open weights and open data for state-of-the-art vision-language models. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 91–104.

[Devlin et al. 2019] Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*, pages 4171–4186.

[Dosovitskiy et al. 2020] Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., et al. (2020). An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*.

[Driess et al. 2023] Driess, D., Xia, F., Sajjadi, M. S. M., Lynch, C., Chowdhery, A., Ichter, B., Wahid, A., Tompson, J., Vuong, Q., Yu, T., Huang, W., Chebotar, Y., Sermanet, P., Duckworth, D., Levine, S., Vanhoucke, V., Hausman, K., Toussaint, M., Greff, K., Zeng, A., Mordatch, I., and Florence, P. (2023). PaLM-E: An embodied multimodal language model.

[Dzabraev et al. 2024] Dzabraev, M., Kunitsyn, A., and Ivaniuta, A. (2024). VLRM: Vision-language models act as reward models for image captioning. *arXiv preprint arXiv:2404.01911*.

[Elizalde et al. 2023] Elizalde, B., Deshmukh, S., Al Ismail, M., and Wang, H. (2023). CLAP learning audio concepts from natural language supervision. In *ICASSP 2023-2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1–5. IEEE.

[Fei et al. 2024] Fei, H., Wu, S., Ji, W., Zhang, H., Zhang, M., Lee, M.-L., and Hsu, W. (2024). Video-of-thought: Step-by-step video reasoning from perception to cognition. *arXiv preprint arXiv:2501.03230*.

[Frantar et al. 2022] Frantar, E., Ashkboos, S., Hoefler, T., and Alistarh, D. (2022). GPTQ: Accurate post-training quantization for generative pre-trained transformers. *arXiv preprint arXiv:2210.17323*.

[Goldman et al. 2025] Goldman, O., Shaham, U., Malkin, D., Eiger, S., Hassidim, A., Matias, Y., Maynez, J., Gilady, A. M., Riesa, J., Rijhwani, S., et al. (2025). ECLeKTic: a novel challenge set for evaluation of cross-lingual knowledge transfer. *arXiv preprint arXiv:2502.21228*.

[Gong et al. 2025] Gong, L., Yang, J., Han, S., and Ji, Y. (2025). MedBLIP: A multimodal method of medical question-answering based on fine-tuning large language model. *Computers in Medical Imaging and Graphics*, 124:102581.

[Goyal et al. 2017] Goyal, Y., Khot, T., Summers-Stay, D., Batra, D., and Parikh, D. (2017). Making the V in VQA matter: Elevating the role of image understanding in visual question answering. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*.

[Grattafiori et al. 2024] Grattafiori, A., Dubey, A., Jauhri, A., Pandey, A., Kadian, A., Al-Dahle, A., Letman, A., Mathur, A., Schelten, A., Vaughan, A., et al. (2024). The LLaMA 3 herd of models. *arXiv preprint arXiv:2407.21783*.

[Han et al. 2025] Han, S., Huang, W., Shi, H., Zhuo, L., Su, X., Zhang, S., Zhou, X., Qi, X., Liao, Y., and Liu, S. (2025). Videoespresso: A large-scale chain-of-thought dataset for fine-grained video reasoning via core frame selection. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 26181–26191.

[Hinton et al. 2015] Hinton, G., Vinyals, O., and Dean, J. (2015). Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*.

[Hossain et al. 2019] Hossain, M. D., Sohel, F., Shiratuddin, M. F., and Laga, H. (2019). A comprehensive survey of deep learning for image captioning. *ACM Computing Surveys (CSUR)*, 51(6):1–36.

[Hu et al. 2022] Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., Chen, W., et al. (2022). Lora: Low-rank adaptation of large language models. *ICLR*, 1(2):3.

[Jegham et al. 2025] Jegham, N., Abdelatti, M., Elmoubarki, L., and Hendawi, A. (2025). How hungry is AI? benchmarking energy, water, and carbon footprint of LLM inference. *arXiv preprint arXiv:2505.09598*.

[Kalai et al. 2025] Kalai, A. T., Nachum, O., Vempala, S. S., and Zhang, E. (2025). Why language models hallucinate. *arXiv preprint arXiv:2509.04664*.

[Krishna et al. 2017] Krishna, R., Zhu, Y., Groth, O., Johnson, J., Hata, K., Kravitz, J., Chen, S., Kalantidis, Y., Li, L.-J., Shamma, D. A., Bernstein, M. S., and Fei-Fei, L. (2017). Visual genome: Connecting language and vision using crowdsourced dense image annotations. *International Journal of Computer Vision (IJCV)*, 123(1):32–73.

[Kuang et al. 2025] Kuang, J., Shen, Y., Xie, J., Luo, H., Xu, Z., Li, R., Li, Y., Cheng, X., Lin, X., and Han, Y. (2025). Natural language understanding and inference with MLLM in visual question answering: A survey. *ACM Computing Surveys*, 57(8):1–36.

[LangChain Inc. 2025a] LangChain Inc. (2025a). LangChain. `https://www.langchain.com/`. Acesso em: 16 out. 2025.

[LangChain Inc. 2025b] LangChain Inc. (2025b). LangGraph. `https://www.langchain.com/langgraph`. Acesso em: 16 out. 2025.

[LangChain Inc. 2025c] LangChain Inc. (2025c). LangSmith. `https://www.langchain.com/langsmith/observability`. Acesso em: 16 out. 2025.

[Lewis et al. 2020] Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W.-t., Rocktäschel, T., et al. (2020). Retrieval-augmented generation for knowledge-intensive NLP tasks. *Advances in neural information processing systems*, 33:9459–9474.

[Li et al. 2024] Li, J., Bercea, C. I., Müller, P., Felsner, L., Kim, S. H., Rueckert, D., Wiestler, B., and Schnabel, J. A. (2024). Multi-image visual question answering for unsupervised anomaly detection. In *CoRR*, volume abs/3sJvMF5m5D. Preprint on OpenReview.

[Li et al. 2023] Li, J., Li, D., Savarese, S., and Hoi, S. (2023). BLIP-2: Bootstrapping language-image pre-training with frozen image encoders and large language models. In *International conference on machine learning*, pages 19730–19742. PMLR.

[Liu et al. 2023] Liu, H., Li, C., Wu, Q., and Lee, Y. J. (2023). Visual instruction tuning. *Advances in neural information processing systems*, 36:34892–34916.

[Liu et al. 2025] Liu, Y., Liang, Z., Wang, Y., Wu, X., Tang, F., He, M., Li, J., Liu, Z., Yang, H., Lim, S., et al. (2025). Unveiling the ignorance of mLLMs: Seeing clearly, answering incorrectly. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 9087–9097.

[Longo et al. 2024] Longo, L., Brcic, M., Cabitza, F., Choi, J., Confalonieri, R., Ser, J. D., Guidotti, R., Hayashi, Y., Herrera, F., Holzinger, A., Jiang, R., Khosravi, H., Lecue, F., Malgieri, G., Páez, A., Samek, W., Schneider, J., Speith, T., and Stumpf, S. (2024). Explainable artificial intelligence (XAI) 2.0: A manifesto of open challenges and interdisciplinary research directions. *Information Fusion*, 106:102301.

[Lopes et al. 2023] Lopes, C. R., Minetto, R., Delgado, M. R., and Silva, T. H. (2023). PerceptSent - exploring subjectivity in a novel dataset for visual sentiment analysis. *IEEE Transactions on Affective Computing*, 14(3).

[Mersha et al. 2024] Mersha, M., Lam, K., Wood, J., AlShami, A. K., and Kalita, J. (2024). Explainable artificial intelligence: A survey of needs, techniques, applications, and future direction. *Neurocomputing*, 599:128111.

[Mondshine et al. 2025] Mondshine, I., Paz-Argaman, T., and Tsarfaty, R. (2025). Beyond english: The impact of prompt translation strategies across languages and tasks in multilingual LLMs. *arXiv preprint arXiv:2502.09331*.

[Oliveira et al. 2020] Oliveira, W. B. d., Dorini, L. B., Minetto, R., and Silva, T. H. (2020). OutdoorSent: Sentiment analysis of urban outdoor images by using semantic and deep features. *ACM Trans. on Information Systems*, 38(3).

[OpenAI 2023a] OpenAI (2023a). GPT-4 technical report. *arXiv preprint arXiv:2303.08774*.

[OpenAI 2023b] OpenAI (2023b). GPT-4V(ision) system card. `https://cdn.openai.com/papers/GPTV_System_Card.pdf`. Acesso em: 3 dez. 2025.

[Pal et al. 2025] Pal, R., Kar, S., Prasad, D. K., and Sekh, A. A. (2025). Multilingual visual question answering for visually impaired people. *Discover Artificial Intelligence*, 5.

[Radford et al. 2021] Radford, A., Kim, J. W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., Sastry, G., Askell, A., Mishkin, P., Clark, J., et al. (2021). Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PmLR.

[Radford et al. 2018] Radford, A., Narasimhan, K., Salimans, T., Sutskever, I., et al. (2018). Improving language understanding by generative pre-training. *OpenAI blog*.

[Radford et al. 2019] Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I., et al. (2019). Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9.

[Raschka 2024] Raschka, S. (2024). Understanding Multimodal LLMs. https://magazine.sebastianraschka.com/p/understanding-multimodal-llms. Acesso em: 16 out. 2025.

[Team et al. 2023] Team, G., Anil, R., Borgeaud, S., Alayrac, J.-B., Yu, J., Soricut, R., Schalkwyk, J., Dai, A. M., Hauth, A., Millican, K., et al. (2023). Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*.

[Touvron et al. 2023] Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.-A., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., et al. (2023). LLaMA: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

[Vaswani et al. 2017] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, 30.

[Vinyals et al. 2015] Vinyals, O., Toshev, A., Bengio, S., and Erhan, D. (2015). Show and tell: A neural image caption generator. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3156–3164.

[Wang et al. 2022] Wang, X., Wei, J., Schuurmans, D., Le, Q., Chi, E., Narang, S., Chowdhery, A., and Zhou, D. (2022). Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*.

[Wei et al. 2022a] Wei, J., Tay, Y., Bommasani, R., Raffel, C., Zoph, B., Borgeaud, S., Yogatama, D., Bosma, M., Zhou, D., Metzler, D., et al. (2022a). Emergent abilities of large language models. *arXiv preprint arXiv:2206.07682*.

[Wei et al. 2022b] Wei, J., Wang, X., Schuurmans, D., Bosma, M., Xia, F., Chi, E., Le, Q. V., Zhou, D., et al. (2022b). Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

[Wolf et al. 2020] Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., et al. (2020). Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations*, pages 38–45.

[Wolfe 2024] Wolfe, C. R. (2024). Decoder-only transformers: The workhorse of generative LLMs. `https://cameronrwolfe.substack.com/p/decoder-only-transformers-the-workhorse`. Acesso em: out. 2025.

[Wu et al. 2023] Wu, J., Gan, W., Chen, Z., Wan, S., and Yu, P. S. (2023). Multimodal large language models: A survey. In *2023 IEEE International Conference on Big Data (BigData)*, pages 2247–2256. IEEE.

[Xiao et al. 2023] Xiao, G., Lin, J., Seznec, M., Wu, H., Demouth, J., and Han, S. (2023). SmoothQuant: Accurate and efficient post-training quantization for large language models. In *International conference on machine learning*, pages 38087–38099. PMLR.

[Yao et al. 2022] Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K. R., and Cao, Y. (2022). ReAct: Synergizing reasoning and acting in language models. In *The eleventh international conference on learning representations*.

[Yin et al. 2024] Yin, S., Fu, C., Zhao, S., Li, K., Sun, X., Xu, T., and Chen, E. (2024). A survey on multimodal large language models. *National Science Review*, 11(12):nwae403.

[Zhang et al. 2023] Zhang, H., Li, X., and Bing, L. (2023). Video-LLaMA: An instruction-tuned audio-visual language model for video understanding. *arXiv preprint arXiv:2306.02858*.

[Zhang et al. 2025] Zhang, X., Chen, Y., Yeh, S., and Li, S. (2025). MetaMind: Modeling human social thoughts with metacognitive multi-agent systems. *arXiv preprint arXiv:2505.18943*.

[Zhu et al. 2023] Zhu, D., Chen, J., Shen, X., Li, X., and Wang, X. (2023). MiniGPT-4: Enhancing vision-language understanding with advanced large language models. *arXiv preprint arXiv:2304.10592*.

A. Exemplos de *pipelines* avançados

A.1. *Pipeline* com o arcabouço LangChain

Essa seção apresenta uma possível implementação das etapas funcionais da *chain* e a criação da cadeia sequencial no LangChain.

```python
from typing import TypedDict, Optional, List, Any
from langchain.schema.runnable import RunnableLambda
from langchain.chains import SequentialChain

class PipelineState(TypedDict, total=False):
    image: Any
    text: str
    roi: Optional[Any]
    ner_entities: dict
    rag_docs: List[dict]
    prompt: str
    model_output: str
    error: Optional[str]
```

```python
def preprocess_node(state: PipelineState) -> PipelineState:
    image = enhance_image(state["image"])
    roi = detect_and_crop_bird(image)
    if roi is None:
        state["error"] = "Nenhuma ave detectada na imagem"
        return state
    state["roi"] = roi
    return state
```

```python
def ner_node(state: PipelineState) -> PipelineState:
    ents = run_ner(state["text"])
    state["ner_entities"] = ents
    return state
```

```python
def rag_node(state: PipelineState) -> PipelineState:
    roi, ents = state["roi"], state["ner_entities"]
    query_embedding = embed_multimodal(roi, ents)
    docs = vector_store_search(query_embedding, filters=ents)
    state["rag_docs"] = docs
    return state
```

```python
def prompt_prep_node(state: PipelineState) -> PipelineState:
    examples = format_as_examples(state["rag_docs"])
    prompt = assemble_prompt(examples, state["ner_entities"])
    state["prompt"] = prompt
    return state
```

```python
def mlmmodel_node(state: PipelineState) -> PipelineState:
    result = call_multimodal_model(
        image=state["roi"],
        prompt=state["prompt"]
    )
    state["model_output"] = result
    return state
```

```python
def error_handler(state: PipelineState) -> PipelineState:
    print("Erro no pipeline:", state.get("error"))
    return state
```

```python
preprocess_chain = RunnableLambda(preprocess_node)
ner_chain = RunnableLambda(ner_node)
rag_chain = RunnableLambda(rag_node)
prompt_chain = RunnableLambda(prompt_prep_node)
model_chain = RunnableLambda(mlmmodel_node)
error_chain = RunnableLambda(error_handler)
```

```
main_chain = SequentialChain(
    chains=[
        preprocess_chain,
        ner_chain,
        rag_chain,
        prompt_chain,
        model_chain
    ],
    input_variables=["image", "text"],
    output_variables=["model_output"]
)
```

```
def run_pipeline(image, text):
    state = {"image": image, "text": text}
    state = preprocess_node(state)
    if state.get("error"):
        return error_handler(state)
    for node in [ner_node, rag_node, prompt_prep_node, mlmmodel_node]:
        state = node(state)
        if state.get("error"):
            return error_handler(state)
    return state
```

```
result = run_pipeline(input_image, input_text)
print("Resultado final:", result.get("model_output"))
```

A.2. *Pipeline* com o arcabouço LangGraph

Nesta seção, apresenta-se uma possível implementação das funcionalidades de cada nó do grafo de estados, a instanciação do grafo, definição de suas conexões, e, por fim, sua execução.

```
from typing import TypedDict, Optional, List, Any
from langgraph.graph import StateGraph, Node
from langgraph.graph import send, conditional_edge  # helpers para fluxo condicional
# (dependendo da versão do LangGraph usada, os nomes das APIs podem variar)

class PipelineState(TypedDict, total=False):
    image: Any                # imagem bruta (e.g. PIL Image, numpy array, etc)
    text: str                 # descrição textual da imagem
    roi: Optional[Any]        # imagem da região de interesse (ave detectada)
```

```
    ner_entities: dict         # entidades extraídas do texto
    rag_docs: List[dict]       # documentos recuperados pelo RAG
    prompt: str                # prompt construído para o MLLM
    model_output: str          # saída final do modelo
    error: Optional[str]       # mensagem de erro, se houver
```

```
def preprocess_node(state: PipelineState) -> PipelineState:
    image = state["image"]
    # Aplica realce de contraste, nitidez, normalização etc
    image = enhance_image(image)
    # Processa a imagem, detecta bounding box da ave, recorta região
    roi = detect_and_crop_bird(image)  # retorna None se não detectar
    if roi is None:
        state["error"] = "Nenhuma ave detectada na imagem"
        return state
    state["roi"] = roi
    return state
```

```
def ner_node(state: PipelineState) -> PipelineState:
    text = state["text"]
    ents = run_ner(text)
    state["ner_entities"] = ents
    return state
```

```
def rag_node(state: PipelineState) -> PipelineState:
    roi = state["roi"]
    ents = state["ner_entities"]
    # gera embedding multimodal (imagem + texto) ou combinação
    query_embedding = embed_multimodal(roi, ents)
    docs = vector_store_search(query_embedding, filters=ents)
    state["rag_docs"] = docs
    return state
```

```python
def prompt_prep_node(state: PipelineState) -> PipelineState:
    docs = state["rag_docs"]
    # transforma os docs em exemplos (texto + imagens associadas)
    examples = format_as_examples(docs)
    prompt = assemble_prompt(examples, state["ner_entities"])
    state["prompt"] = prompt
    return state
```

```python
def mlmmodel_node(state: PipelineState) -> PipelineState:
    image = state["roi"]
    prompt = state["prompt"]
    resp = call_multimodal_model(image=image, prompt=prompt)
    state["model_output"] = resp
    return state
```

```python
def error_node(state: PipelineState) -> PipelineState:
    # opcional: levantar exceção ou retornar um estado final com mensagem
    return state
```

```python
graph = StateGraph[PipelineState]()

n_pre = graph.add_node(Node(preprocess_node, name="Preprocess"))
n_ner = graph.add_node(Node(ner_node, name="NER"))
n_rag = graph.add_node(Node(rag_node, name="RAG"))
n_prompt = graph.add_node(Node(prompt_prep_node, name="PromptPrep"))
n_model = graph.add_node(Node(mlmmodel_node, name="MLLM"))
n_err = graph.add_node(Node(error_node, name="Error"))
```

```python
graph.add_edge(n_pre, n_ner)
graph.add_edge(n_ner, n_rag)
graph.add_edge(n_rag, n_prompt)
graph.add_edge(n_prompt, n_model)
```

```
# condição de erro: se no preprocess estado.error está definido → pula para error_node
graph.add_conditional_edge(
    from_node=n_pre,
    to_node=n_err,
    condition=lambda state: state.get("error") is not None
)
# caso contrário, segue para n_ner
graph.add_edge(n_pre, n_ner)
```

```
graph.set_start(n_pre)
```

```
initial_state: PipelineState = {"image": input_image, "text": input_text}
final_state = graph.run(initial_state)
final_state.model_output # contém a predição ou final_state.error a mensagem de falha
```