UNIVERSIDADE FEDERAL DE GOIÁS / INSTITUTO DE INFORMÁTICA

Modelos de Visão-Linguagem Compactos

Fine-Tuning e Otimização para Execução em Dispositivos de Borda

Victor Lucas Sousa Arantes

UFG
UNIVERSIDADE FEDERAL DE GOIÁS

UNIVERSIDADE FEDERAL DE GOIÁS (UFG)
INSTITUTO DE INFORMÁTICA (INF)

VICTOR LUCAS SOUSA ARANTES

Modelos de Visão-Linguagem Compactos

Fine-Tuning e Otimização para Execução em Dispositivos de Borda

Goiânia
2025

UNIVERSIDADE FEDERAL DE GOIÁS
INSTITUTO DE INFORMÁTICA

TERMO DE CIÊNCIA E DE AUTORIZAÇÃO PARA DISPONIBILIZAR VERSÕES ELETRÔNICAS DE TRABALHO DE CONCLUSÃO DE CURSO DE GRADUAÇÃO NO REPOSITÓRIO INSTITUCIONAL DA UFG

Na qualidade de titular dos direitos de autor, autorizo a Universidade Federal de Goiás (UFG) a disponibilizar, gratuitamente, por meio do Repositório Institucional (RI/UFG), regulamentado pela Resolução CEPEC no 1240/2014, sem ressarcimento dos direitos autorais, de acordo com a Lei no 9.610/98, o documento conforme permissões assinaladas abaixo, para fins de leitura, impressão e/ou download, a título de divulgação da produção científica brasileira, a partir desta data.

O conteúdo dos Trabalhos de Conclusão dos Cursos de Graduação disponibilizado no RI/UFG é de responsabilidade exclusiva dos autores. Ao encaminhar(em) o produto final, o(s) autor(a)(es)(as) e o(a) orientador(a) firmam o compromisso de que o trabalho não contém nenhuma violação de quaisquer direitos autorais ou outro direito de terceiros.

1. Identificação do Trabalho de Conclusão de Curso de Graduação (TCCG)

Nome(s) completo(s) do(a)(s) autor(a)(es)(as): VICTOR LUCAS SOUSA ARANTES

Título do trabalho: Modelos de Visão-Linguagem Compactos

Fine-Tuning e Otimização para Execução em Dispositivos de Borda

2. Informações de acesso ao documento (este campo deve ser preenchido pelo orientador) Concorda com a liberação total do documento [X] SIM [ ] NÃO[1]

[1] Neste caso o documento será embargado por até um ano a partir da data de defesa. Após esse período, a possível disponibilização ocorrerá apenas mediante: a) consulta ao(à)(s) autor(a)(es)(as) e ao(à) orientador(a); b) novo Termo de Ciência e de Autorização (TECA) assinado e inserido no arquivo do TCCG. O documento não será disponibilizado durante o período de embargo.

Casos de embargo:

- Solicitação de registro de patente;

- Submissão de artigo em revista científica;

- Publicação como capítulo de livro.

Obs.: Este termo deve ser assinado no SEI pelo orientador e pelo autor.

Documento assinado eletronicamente por **Victor Lucas Sousa Arantes**, **Discente**, em 05/02/2026, às 16:15, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020.

Documento assinado eletronicamente por **Fernando Marques Federson**, **Professor do Magistério Superior**, em 13/03/2026, às 11:46, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020 .

VICTOR LUCAS SOUSA ARANTES

Modelos de Visão-Linguagem Compactos

Fine-Tuning e Otimização para Execução em Dispositivos de Borda

Relatório final de Trabalho de Conclusão de Curso, apresentado à Universidade Federal de Goiás, como parte das exigências para a obtenção do título de Bacharel em Inteligência Artificial.

Orientador: Prof. Dr. Fernando Marques Federson

Goiânia

2025

Ficha de identificação da obra elaborada pelo autor, através do Programa de Geração Automática do Sistema de Bibliotecas da UFG.

ARANTES, VICTOR LUCAS SOUSA
Modelos de Visão-Linguagem Compactos [manuscrito]: Fine-Tuning e Otimização para Execução em Dispositivos de Borda / VICTOR LUCAS SOUSA ARANTES. - 2025.
89 f.: 2025

Orientador: Prof. Dr. Fernando Marques Federson
Trabalho de Conclusão de Curso (Graduação) - Universidade Federal de Goiás, Instituto de Informática (INF), Inteligência Artificial, Goiânia, 2025.

1. Inteligência Artificial. 2. Modelos de Visão-linguagem. 3. Dispositivoa de Borda.

I. Federson, Fernando Marques , orient. II. Título.

CDU 004

VICTOR LUCAS SOUSA ARANTES

Modelos de Visão-Linguagem Compactos

Fine-Tuning e Otimização para Execução em Dispositivos de Borda

Relatório final de Trabalho de Conclusão de Curso, apresentado à Universidade Federal de Goiás, como parte das exigências para a obtenção do título de Bacharel em Inteligência Artificial.

Data da Aprovação: 09 de dezembro de 2025.

Prof. Dr. Fernando Marques Federson
Orientador (INF-UFG)

Prof. Dr. Aldo André Díaz Salazar
Coordenador de TCC do BIA (INF-UFG)

Prof. Dr. Anderson da Silva Soares
Coordenador do BIA (INF-UFG)

Me. Juliana Resplande Sant'anna Gomes
(INF-UFG)

VICTOR LUCAS SOUSA ARANTES

Modelos de Visão-Linguagem Compactos

Fine-Tuning e Otimização para Execução em Dispositivos de Borda

RESUMO

Este Relatório de Conclusão de Curso tem como objetivo reunir os resultados da minha jornada para me tornar um especialista em **Modelos de Visão-Linguagem Compactos**. Uma ilustração e sua narrativa descrevem os períodos de trabalho. Os Apêndices contêm os Termos de Aceite de Entrega e os resultados obtidos durante cada período de trabalho.

Palavras-chave: Inteligência artificial; Modelos de visão-linguagem ; Dispositivo de borda.

ABSTRACT

This Course Completion Report aims to bring together the results of my journey to become an expert in **Compact Vision-Language Models**. An illustration and its narrative describe the work periods. The Appendices contain the Delivery Acceptance Terms and the results obtained during each work period.

Keywords: Artificial intelligence; Vision-language models; Edge device.

Goiânia

2025

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DE GOIÁS
INSTITUTO DE INFORMÁTICA

Minha Jornada

Victor Lucas Sousa Arantes

Especialista em: Modelos de Visão-Linguagem Compactos

Instituto de Informática (INF) - UFG
Alameda Palmeiras, Quadra D, Câmpus Samambaia
Fone: (62) 3521-1181 / Fax: (62) 3521-1182

MINHA JORNADA

Nome: **Victor Lucas Sousa Arantes**

Especialidade: **Modelos de Visão-Linguagem Compactos**

Objetivo deste documento

Durante o processo da disciplina Residência em IA[1], foram gerados diversos resultados na construção da minha especialização. A cada semana, um conjunto de resultados foi formalizado por um Termo de Aceite de Entrega e avaliado por uma banca, considerando o planejado e o realizado para o período. Este documento tem como objetivo descrever esses resultados obtidos, fazendo referência aos Termos de Aceite de Entrega e seus documentos associados.

Minha Jornada

Minha jornada começou nas **Semanas 1 e 2**, em que me dediquei primeiro a explorar a grande área de modelos multimodais e, em seguida, a afunilar esse interesse até chegar aos **Modelos de Visão-Linguagem Compactos**. Na **Semana 1**, estudei o artigo *A Survey on Multimodal Large Language Models*[2], revisitei artigos sobre modelos multimodais que eu já tinha lido anteriormente e registrei novas notas sobre modelos como CLIP, ALIGN e SigLIP, além de experimentar o CLIP na prática em um notebook oficial da OpenAI, o que me ajudou a entender de forma como estes modelos relacionam imagens e textos. A formação da disciplina de Processamento de Linguagem Natural foi essencial para interpretar a parte textual desses modelos e conectar melhor os conceitos de Linguagem com os de Visão Computacional; os artigos revisitados e as anotações produzidas nessa fase inicial estão relacionados no **Apêndice 1**. Já na **Semana 2**, a partir dessas leituras e reflexões, defini de forma mais clara que meu foco seria em **Modelos de Visão-Linguagem**

[1] Dez Semanas, entre setembro de 2025 e dezembro de 2025.
[2] YIN, Shukang *et al. A Survey on Multimodal Large Language Models*. National Science Review, v. 11, n. 12, p. 403, 14 nov. 2024.

Compactos e passei a estudar o artigo *A Survey on Efficient Vision-Language Models*[3], que se tornou uma das minhas principais referências para o tema. Ao perceber que ainda me faltava base para aproveitar esse material em profundidade, iniciei o curso de Visão Computacional da Hugging Face até o módulo dedicado a modelos multimodais, registrando os pontos que dialogavam diretamente com o tema da especialização. As anotações desse curso e a síntese do *survey*, que organizam as principais técnicas e direções de pesquisa, encontram-se descritas no **Apêndice 2**.

Nas **Semanas 3 e 4**, com a área de interesse já definida, concentrei-me em revisar artigos recentes e mapear as principais arquiteturas compactas de Modelos de Visão-Linguagem (VLMs). Inicialmente, eu havia planejado voltar às bases teóricas de técnicas como LoRA e destilação, mas optei por uma abordagem diferente: levantar quais modelos surgiram após o *survey* que eu havia lido e entender que soluções de eficiência estavam sendo propostas. Para isso, organizei uma tabela com os principais trabalhos e modelos, classificando as tarefas que realizam e registrando, a partir de uma leitura inicial, quais abordagens utilizam para reduzir custo computacional (como condensar *tokens* visuais, comprimir imagens ou simplificar componentes da arquitetura); essa tabela está relacionada no **Apêndice 3.** Em paralelo, aprofundei o estudo de algumas dessas linhas de pesquisa, incluindo propostas que adaptam *Large Language Models* (LLMs) para incorporar Visão sem um encoder dedicado, métodos que condensam os *tokens* visuais em representações mais compactas nas primeiras camadas e arquiteturas desenhadas desde o início com a filosofia de serem “small-first”, combinando compressão de imagem e ajustes cuidadosos de dados e *reasoning*. Também acompanhei a evolução de versões mais recentes desses modelos por meio de blogs técnicos e consultei *leaderboards* específicos para VLMs, o que ajudou a situar essas arquiteturas no cenário atual de *benchmarks*. As principais anotações dessas leituras e experimentações encontram-se detalhadas no **Apêndice 3**, que registra esse movimento de mapeamento das soluções atuais em VLMs Compactos.

[3] SHINDE, Gaurav *et al. A Survey on Efficient Vision-Language Models*. arXiv, , 1 jul. 2025. Disponível em: <http://arxiv.org/abs/2504.09724>. Acesso em: 3 set. 2025.

Na **Semana 5**, aprofundei o estudo em VLMs Compactos com foco em cenários de execução em dispositivos móveis e ambientes com recursos limitados. Analisei abordagens que fazem um co-projeto entre algoritmo e sistema, estudando estratégias para reduzir custo computacional sem comprometer excessivamente o desempenho, como o ajuste mais cuidadoso da resolução das imagens para evitar aumentos desnecessários, mecanismos de redução do número de tokens visuais e esquemas de processamento em blocos menores para melhor se adaptar às restrições de aceleração em hardware. Em paralelo, examinei de forma detalhada o código de um VLM Compacto com implementação educacional em PyTorch. Consegui entender o fluxo de pré-treinamento, desde o preparo dos dados até a etapa de otimização, o que me deu uma visão mais clara de como esses modelos são estruturados na prática. Por fim, estudei propostas recentes de raciocínio multimodal que introduzem *tokens* visuais latentes ao longo da cadeia de pensamento, funcionando como "esboços internos" que auxiliam o modelo a combinar melhor pistas textuais e visuais, com resultados especialmente interessantes em arquiteturas compactas. Os códigos analisados, bem como as sínteses dessas leituras e experimentos, estão descritos no **Apêndice 4**.

Nas **Semanas 6 e 7**, avancei da etapa de estudo de modelos compactos para a experimentação de técnicas inspiradas na literatura, começando por finalizar o pré-treinamento e a avaliação de um NanoVLM[4] seguindo a receita proposta no próprio modelo. Mesmo com um tempo de treinamento reduzido, o desempenho obtido em um *benchmark* multimodal ficou próximo ao relatado como referência para esse tipo de modelo, o que foi suficiente para que eu entendesse, na prática, o fluxo completo de treinamento, avaliação automática com LLMs e testes interativos. Em paralelo, aprofundei o estudo de métodos de raciocínio multimodal que procuram melhorar a cadeia de pensamento dos VLMs a partir de respostas curtas, combinando geração sintética de passos intermediários de raciocínio com técnicas de alinhamento por reforço, e explorei propostas que estruturam o raciocínio visual como uma sequência de manipulações explícitas sobre a imagem (localizar, recortar, contar, aplicar OCR, entre outras ações), bem como abordagens que substituem a arquitetura Transformer por modelos de espaço de estados no contexto

[4] WIEDMANN, L.; GOSTHIPATY, A. R.; MARAFIOTI, A. *nanoVLM.* 2025. Repositório GitHub. Disponível em: https://github.com/huggingface/nanoVLM. Acesso em: 25 nov. 2025.

multimodal. Motivado por essas leituras, iniciei uma primeira tentativa de replicar e adaptar uma técnica de compactação baseada em “tokens de registro”, vista no artigo do método VICTOR[5], em um VLM de pequeno porte, partindo de uma implementação simples e incorporando gradualmente as modificações necessárias na arquitetura. Embora essa versão inicial ainda exija refinamentos, já foi possível realizar treinamentos de teste e verificar a viabilidade dessa linha de trabalho. Os resultados de avaliação do NanoVLM, as anotações sobre as diferentes formas de raciocínio multimodal e os registros da primeira implementação com tokens de registro estão agrupados no **Apêndice 5**.

Nas **Semanas 8 e 9**, mantive o foco em VLMs Compactos, mas passei de um olhar mais conceitual para um esforço mais direto de implementação e de aproximação com as aplicações. Na **Semana 8**, continuei tentando implementar, em um modelo compacto, a técnica de resumir os *tokens* de visão em um conjunto reduzido de *tokens* de registro nas primeiras camadas do modelo de linguagem, explorando combinações de *backbones*, diferentes quantidades de *tokens* de registro e pontos distintos da arquitetura em que esses *tokens* seriam condensados. Apesar das tentativas, o modelo não convergiu de forma satisfatória e a adaptação desse mecanismo a um código minimalista em PyTorch se mostrou mais complexa do que o previsto. Isto me levou a reconsiderar a viabilidade de seguir replicando essa abordagem específica e passar a olhar com mais atenção para alternativas com códigos e *pipelines* já disponibilizados publicamente, como métodos que incorporam visão em LLMs de forma *encoder-free* usando LoRA e destilação. Em paralelo, comecei a aproximar o estudo das **aplicações**, analisando um modelo leve de *GUI Grounding* que adapta VLMs compactos para localizar elementos de interface gráfica (web, mobile, desktop) a partir de comandos em linguagem natural, com código e *datasets* acessíveis para reprodução em GPU única. As anotações dessa leitura, bem como o registro das dificuldades e aprendizados com os *tokens* de registro, estão organizados no **Apêndice 6**. Já na **Semana 9**, com esse panorama acumulado, foquei em replicar, de forma mais sistemática, uma *pipeline* de Fine-tuning para *GUI Grounding* em um modelo compacto pré-treinado com capacidades de *grounding*, conseguindo reproduzir os resultados de

[5] WEN, Yuxin *et al. Efficient Vision-Language Models by Summarizing Visual Tokens into Compact Registers.* arXiv, , 17 out. 2024. Disponível em: <http://arxiv.org/abs/2410.14072>. Acesso em: 17 set. 2025.

benchmarks relatados pelos autores e entendendo melhor como um *dataset* relativamente pequeno, mas bem adaptado à tarefa, pode especializar um VLM para interfaces gráficas. A partir daí, comecei a investigar como técnicas semelhantes poderiam ser trazidas para modelos ainda menores. Explorei tentativas de adaptação de tutoriais de detecção de objetos para LLMs e, em seguida, passei a estudar uma *pipeline* que treina VLMs compactos para gerar diretamente coordenadas de ação em interfaces e servir de base para agentes de computador multi-etapa, realizando experimentos iniciais com modelos de 256M e 500M de parâmetros. Esses experimentos com *GUI Grounding*, agentes para uso de computador e os registros sobre limitações, reproduções de resultados e próximas possibilidades estão descritos no **Apêndice 7**.

Na **Semana 10**, decidi direcionar meus esforços para entender como levar modelos de Visão-Linguagem Compactos para dispositivos de borda, em especial o celular. Retomei estudos sobre co-projeto entre algoritmo e sistema em VLMs voltados para dispositivos móveis e, a partir daí, foquei em práticas de inferência com motores leves, o que me levou à necessidade de converter modelos para um formato unificado de pesos quantizados adequado a esse tipo de ambiente. Esse processo exigiu um mergulho em formatos como GGUF e em técnicas de *Post-Training Quantization* (PTQ), o que acabou ampliando minha compreensão sobre como reduzir o custo de memória e de computação preservando, na medida do possível, a qualidade das respostas. Com isso, consegui realizar testes tanto com modelos relativamente pequenos (na ordem de centenas de milhões de parâmetros), quanto com modelos mais robustos, avaliando na prática os limites e compromissos de executar VLMs em hardware restrito. Em paralelo, revisitei o modelo de *GUI Grounding* treinado anteriormente, realizando testes manuais que mostraram que ele havia aprendido corretamente o padrão de formatação das respostas, mas ainda apresentava alguns erros nas coordenadas finais. A partir dessa observação, passei a considerar, como possibilidade de trabalho futuro, um esquema de treinamento orientado por recompensa em que a distância entre a coordenada correta e a predita serviria como sinal de melhoria para o modelo. As anotações sobre quantização, os experimentos de execução em dispositivos de borda e as reflexões sobre ajustes de treinamento em *GUI Grounding* estão reunidos no **Apêndice 8**.

Ao olhar para tudo que vivi ao longo destas **Semanas**, percebo o quanto essa jornada representou, para mim, uma primeira aproximação real com a pesquisa acadêmica. Eu comecei o processo sem uma base consolidada em leitura de artigos científicos, pois ao longo da graduação tinha lido poucos trabalhos. Aos poucos, fui aprendendo a escolher *papers*, organizar referências, fazer anotações mais críticas e construir um vocabulário próprio da área. Nesse caminho, aprofundei conhecimentos em Processamento de Linguagem Natural e em Visão Computacional, conteúdos que eu já havia conhecido nas disciplinas, mas que ainda não tinha tido oportunidade de explorar com tanta profundidade. Estudar modelos de Visão-Linguagem Compactos, acompanhar propostas recentes, tentar entender arquiteturas, técnicas de eficiência e aplicações práticas foi algo que me instigou bastante. Descobri uma área em que eu realmente gosto de ler, de investigar e de testar ideias. Termino esta jornada feliz por ter desenvolvido não só competências técnicas, mas também uma postura mais curiosa e autônoma diante da pesquisa, com vontade de continuar estudando, contribuindo e me aprofundando nesse campo.

Finalmente, gostaria de registrar minha profunda gratidão a todas as pessoas que me acompanharam nessa jornada. Agradeço especialmente ao professor Fernando Federson, que foi um guia tão sensível e competente durante todo esse processo. As conversas que tive com ele ao longo da graduação, sobretudo nos momentos iniciais e finais do curso, influenciaram profundamente a maneira como eu enxergo a área, a universidade e a minha própria vida. Estendo meu agradecimento aos professores Cedric Luiz e Leonardo Alves, cuja dedicação em sala de aula, nos projetos e na Residência em IA contribuíram muito para a minha formação acadêmica e profissional. Reconheço o privilégio de ter sido aluno de cada um dos meus professores no Bacharelado em Inteligência Artificial. Também agradeço aos meus amigos, que foram minha companhia no dia a dia, em grupos de estudo, tarefas, provas e desafios do curso. Deixo ainda um agradecimento muito especial à minha companheira, Sofia Costa, que esteve ao meu lado, compartilhando preocupações, comemorando cada pequena conquista e me ajudando sempre que foi preciso. Por fim, e mais importante, não poderia deixar de mencionar minha família, que sempre esteve por perto mesmo à distância, apoiando minha decisão de mudar de cidade para estudar e me

oferecendo todo o suporte necessário, em todos os aspectos, no decorrer da minha graduação.

APÊNDICE 1

Termo de Aceite de Entrega

Objetivo deste documento

Este documento faz parte do Processo da disciplina Residência em IA e tem como objetivo formalizar o aceite da entrega considerando o planejado e o realizado para o período.

Data da Reunião ("Gate") de aprovação: 4 de set. de 2025

Participantes da Entrega [matriculados em Residência em IA]:

VICTOR LUCAS SOUSA ARANTES

Entrega: [descrever a ENTREGA - requisitos e produtos gerados: links para textos, códigos, vídeos etc.]

Grande área: Modelos Multimodais

Encontrei esse artigo(A Survey on Multimodal Large Language Models) e gerei anotações.

Enquanto lia, percebi alguns termos que eu já tinha visto em alguns artigos que li meses atrás, aí fiz uma releitura rápida e gerei algumas notas sobre o CLIP, ALIGN, SIGLIP.
Também interagi com o CLIP por meio desse notebook no github deles.

Procurei alguns pesquisadores da área e eles me recomendaram fazer esse curso de visão e também a tentar treinar um NanoVLM.

Pesquisando um pouco mais, me interessei pela área de vision language models compactos e encontrei esse artigo que trata um pouco disso (A Survey on Efficient Vision-Language Models).

Planejamento: [descrever o que pretende fazer para realizar a próxima ENTREGA]

- **Fazer a leitura do artigo** A Survey on Efficient Vision-Language Models
- **Fazer até a unidade 4 do curso de visão do HF**
- **Buscar mais fontes sobre modelos de visão-linguagem compactos**

Observação: [caso precise fazer alguma observação, de qualquer "natureza"]

ACEITE DA ENTREGA:

CEDRIC LUIZ DE CARVALHO: Go!

Instituto de Informática (INF) - UFG
Alameda Palmeiras, Quadra D, Câmpus Samambaia
Fone: (62) 3521-1181 / Fax: (62) 3521-1182

Fundamentos de Modelos Multimodais

Para estudar modelos multimodais, resolvi buscar modelos que foram importantes para o desenvolvimento da área. Em minhas buscas, encontrei o modelo CLIP (Contrastive Language-Image Pre-Training) que propõe o aprendizado de representações visuais transferíveis via supervisão de linguagem natural (texto cru). Diferente de sistemas de visão tradicionais treinados para prever um conjunto fixo de categorias, o CLIP é treinado para prever qual legenda (dentro de um lote) corresponde a qual imagem.

O artigo apresenta um mecanismo de treinamento por meio de uma *Contrastive Loss,* em que o modelo recebe um *batch* de N pares de imagem e tem como objetivo maximizar a similaridade de cossenos dos N pares corretos e minimizar a dos N^2 - N pares incorretos, utilizando uma loss simétrica de entropia cruzada.

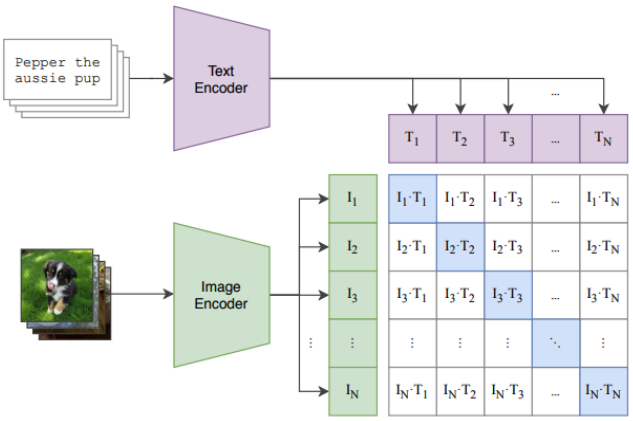


Figura 1.1. Esquema do treinamento contrastivo do CLIP, com encoders de texto e imagem gerando uma matriz de similaridade N×N entre pares imagem–texto.[6]

A arquitetura possui dos encoders, sendo um *Image Encoder* (testaram ResNet-50 e Vision Transformer - ViT) e um *Text Encoder* (Transformer). As representações são projetadas para um espaço de embeddings multimodal.

Uma das maiores contribuições foi o Zero-Shot Transfer, o modelo consegue resolver tarefas novas sem nunca ter sido treinado especificamente nelas. Para classificação, o CLIP utiliza *prompt engineering* (ex: "a photo of a {label}") para criar representações textuais das classes e compará-las com a imagem.

[6] Reproduzido de RADFORD et al. (2021).

Foram observadas limitações com conceitos *fine-grained*, como contar objetos ou diferenciar variações sutis, além de problemas com polissemia quando o contexto textual é curto.

A fim de melhorar o vocabulário e conhecer mais sobre os métodos que esses modelos multimodais utilizavam no treinamento e arquitetura, busquei artigos que tentavam melhorar o modelo CLIP. Entre eles, o modelo ALIGN (A Large-scale ImaGe and Noisy-text embedding model), que ao invés de usar datasets curados (como MSCOCO), os autores criam um dataset massivo e ruidoso de mais de 1 bilhão de pares imagem-texto (alt-texts). O estudo demonstrou que a escala compensa o ruído nos dados, conseguindo forte desempenho em tarefas de classificação (ImageNet, VTAB) e recuperação de imagem-texto (Flickr30K, MSCOCO), superando *baselines* em diversos cenários *zero-shot*.

Outro modelo muito importante para a área foi o SigLIP (Sigmoid Loss for Language Image Pre-Training), que tem como diferença fundamental a função de perda adotada. Em vez da loss contrastiva com softmax e normalização global utilizada no CLIP, os autores propõem uma *sigmoid loss* calculada par a par entre imagem e texto. Cada par positivo ou negativo é tratado como uma pequena tarefa de classificação binária, sem precisar comparar explicitamente todas as combinações possíveis do lote.

Do ponto de vista computacional, essa mudança reduz o acoplamento entre os exemplos do batch e elimina a dependência quadrática da matriz de similaridades N×N, diminuindo também a necessidade de comunicação entre GPUs. Com isso, o treinamento passa a ser mais flexível em relação ao tamanho do batch, funcionando bem tanto com batches muito grandes quanto com batches menores. Os autores mostram que, com essa alteração relativamente simples, o modelo alcança desempenho competitivo em zero-shot no ImageNet (cerca de 84,5% de acurácia), com maior eficiência de treinamento. Na minha leitura, o SigLIP é um exemplo de como ajustes na função de perda podem impactar diretamente a escalabilidade dos VLMs e influenciar o desenho de arquiteturas mais recentes, inclusive em cenários de modelos compactos para dispositivos de borda.

Em paralelo foi feita a leitura do artigo *A Survey on Multimodal Large Language Models*, que contém uma visão mais geral da área, apresentando os MLLMs como modelos que utilizam um LLM como “cérebro” e organizam a arquitetura em três blocos principais: um *modality encoder* (por exemplo, um ViT em estilo CLIP) para processar as entradas visuais,

um *LLM backbone* (como LLaMA ou Vicuna) responsável pelo raciocínio em linguagem natural, e uma *modality interface* (conector) que faz a ponte entre esses dois componentes, seja por projeções simples, módulos baseados em tokens intermediários ou mecanismos de fusão de *features*.

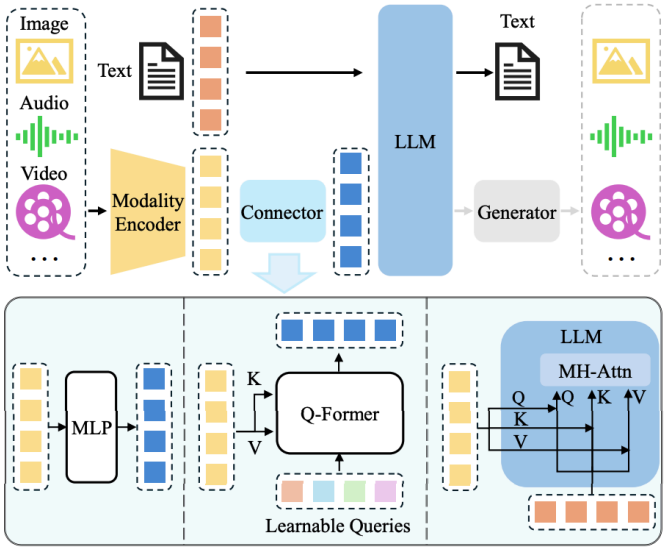


Figura 1.2. Arquitetura típica de um MLLM[7]

O artigo também descreve três estágios de treinamento recorrentes nesses modelos: o *pre-training* para alinhar imagem e texto em grande escala, o *instruction tuning* para adaptar o modelo a instruções e tarefas variadas (como como VQA – *Visual Question Answering*, e captioning em formato de diálogo) e o *alignment tuning* com técnicas como RLHF ou DPO para aproximar o comportamento do modelo das preferências humanas e reduzir alucinações. Por fim, o survey discute em detalhes diferentes tipos de alucinação multimodal (de existência, de atributos e de relacionamentos entre objetos) e apresenta métricas específicas, como POPE e CHAIR, que ajudam a avaliar se as respostas do modelo são realmente consistentes com o conteúdo visual.

[7] Reproduzido de YIN et al. (2024).

Referências

CLIP

RADFORD, Alec *et al.* **Learning Transferable Visual Models From Natural Language Supervision**. arXiv, , 26 fev. 2021. Disponível em: <http://arxiv.org/abs/2103.00020>. Acesso em: 2 dez. 2025

ALIGN

JIA, Chao *et al.* **Scaling Up Visual and Vision-Language Representation Learning With Noisy Text Supervision**. arXiv, , 11 jun. 2021. Disponível em: <http://arxiv.org/abs/2102.05918>. Acesso em: 16 abr. 2025

SigLIP

ZHAI, Xiaohua *et al.* **Sigmoid Loss for Language Image Pre-Training**. arXiv, , 27 set. 2023. Disponível em: <http://arxiv.org/abs/2303.15343>. Acesso em: 16 abr. 2025

Survey MLLMs

YIN, Shukang *et al.* A Survey on Multimodal Large Language Models. **National Science Review**, v. 11, n. 12, p. nwae403, 14 nov. 2024.

APÊNDICE 2

Termo de Aceite de Entrega

Objetivo deste documento

Este documento faz parte do Processo da disciplina Residência em IA e tem como objetivo formalizar o aceite da entrega considerando o planejado e o realizado para o período.

Data da Reunião ("Gate") de aprovação: 11 de set. de 2025

Participantes da Entrega [matriculados em Residência em IA]:

VICTOR LUCAS SOUSA ARANTES

Entrega: [descrever a ENTREGA - requisitos e produtos gerados: links para textos, códigos, vídeos etc.]

Área de Interesse: Modelos multimodais → Modelos de visão-linguagem → Modelos de visão-linguagem compactos

Nessa Semana, assim como no planejamento passado, comecei com a leitura do A Survey on Efficient Vision-Language Models, porém senti que me faltava um pouco de bagagem na área de Visão Computacional.

Portanto, decidi fazer o curso que os pesquisadores da área haviam me recomendado antes de terminar a leitura. Fui até o módulo 4 que tratava sobre modelos multimodais e gerei algumas anotações sobre os tópicos que achei interessantes.

Terminei a leitura do Survey, conheci algumas técnicas, modelos e aplicações. Grifei as partes que achei importante e também gerei uma nota agrupando os conteúdos.

De acordo com as minhas leituras, ainda me falta bagagem técnica em NLP, percebi que conhecia os nomes das técnicas mas não sabia a fundo como funcionam.

Planejamento: [descrever o que pretende fazer para realizar a próxima ENTREGA]

Na próxima semana planejo seguir uma abordagem
Fundamento -> Técnica -> Aplicação

Para isso, pretendo ler esses papers:
Attention Is All You Need
LoRA: Low-Rank Adaptation of Large Language Models
DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter

Além disso, pretendo iniciar a organização dos meus conhecimentos por meio de um mapa mental.

Observação: **[caso precise fazer alguma observação, de qualquer "natureza"]**

Survey de Modelos de Visão-Linguagem Eficientes e Revisão Inicial

Depois de estudar CLIP, ALIGN, SigLIP e o survey de MLLMs, senti necessidade de entender melhor como esses modelos podem ser tornados mais leves e eficientes para rodar em dispositivos de borda. Por isso, li o artigo A Survey on Efficient Vision-Language Models, que organiza o tema de forma bem estruturada e foca justamente em otimizações para VLMs em cenários com memória e processamento limitados (como celulares, placas embarcadas e GPUs menores).

O artigo propõe uma visão geral em que as técnicas de eficiência são divididas em alguns blocos principais: métodos aplicados antes do deploy (como quantização, low-rank approximation, pruning e knowledge distillation), estratégias de *fine-tuning* eficiente (PEFT e MEFT), otimizações em tempo de execução (redução de tokens, adaptação em teste) e abordagens de treinamento distribuído e preservação de privacidade, além de uma discussão sobre VLMs compactos recentes e o *trade-off* entre desempenho e custo computacional.

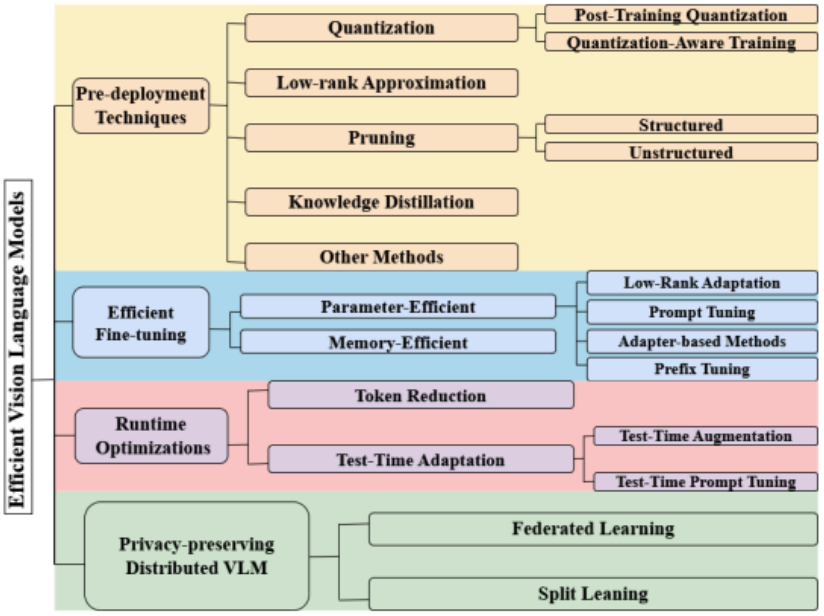


Figura 2.1. Visão geral das principais categorias de técnicas para tornar Vision-Language Models mais eficientes (pré-deploy, fine-tuning eficiente, otimizações em tempo de execução, modelos distribuídos e arquiteturas compactas).[8]

Uma das técnicas mais enfatizadas é a quantização, que reduz a precisão numérica de pesos e ativações (por exemplo, de FP32 para FP16 ou INT8) para diminuir memória e acelerar a inferência. O artigo discute dois eixos importantes: a diferença entre quantização simétrica e assimétrica (com ou sem zero-point) e o contraste entre PTQ (Post-Training Quantization) e QAT (Quantization-Aware Training). No PTQ, o modelo já treinado é quantizado depois, de forma mais simples e rápida, mas com risco maior de perda de acurácia. No QAT, a quantização é simulada durante o treinamento, o que tende a preservar melhor o desempenho, à custa de mais tempo de treino.

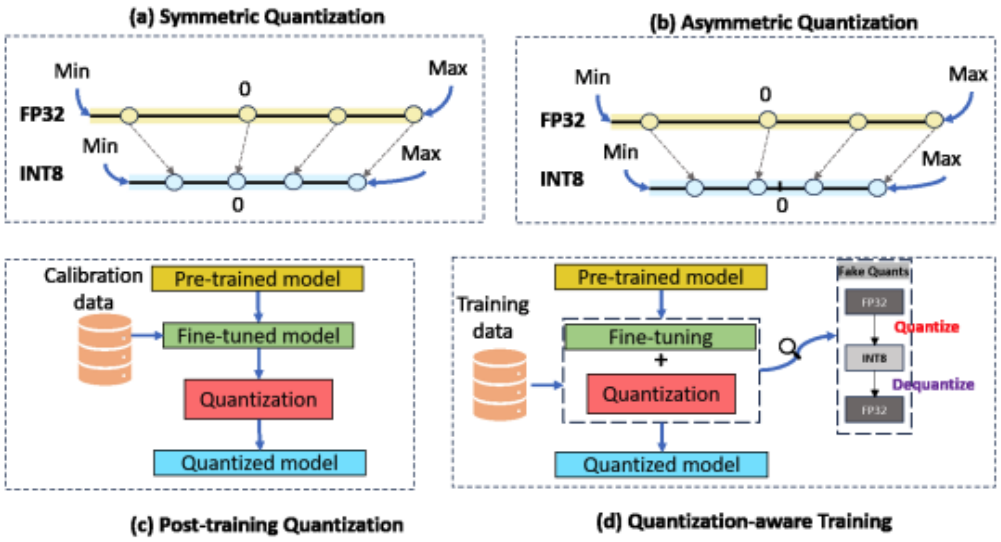


Figura 2.2. Esquema comparando quantização simétrica/assimétrica e as abordagens PTQ e QAT em modelos de visão-linguagem.[9]

Outra família de técnicas descrita é a de aproximações de baixo rank (Low-Rank Approximation), em que matrizes grandes do modelo são aproximadas por produtos de matrizes menores. A intuição é reduzir o número de parâmetros e operações (FLOPs) mantendo a maior parte da informação. De forma parecida, o *pruning* remove pesos considerados pouco importantes. O survey diferencia *structured pruning* (remover colunas, canais, *heads* ou até *tokens* inteiros, o que é mais amigável para o hardware) de

[8] Reproduzido de SHINDE et al. (2025)
[9] Reproduzido de SHINDE et al. (2025)

unstructured pruning (remover pesos isolados, gerando matrizes esparsas mais difíceis de explorar na prática). Em ambos os casos, existe um limite, podar demais leva a uma queda acentuada na qualidade do modelo.

O texto também destaca a *knowledge distillation* (KD) como ferramenta central para construir VLMs menores. Em linhas gerais, um modelo aluno aprende a imitar um modelo professor maior. São descritas três formas principais: destilação baseada em respostas (imitar as saídas finais, com soft labels), baseada em features (aproximar ativações intermediárias) e baseada em relações (preservar distâncias e similaridades entre embeddings, em vez de valores absolutos). Essas estratégias permitem treinar modelos compactos que ainda se beneficiam do conhecimento multimodal de um VLM grande pré-treinado.

Além do pré-deploy, o survey abrange técnicas de fine-tuning eficiente. Em PEFT (Parameter-Efficient Fine-Tuning) entram métodos como LoRA, prompt tuning, adapters e prefix tuning, que ajustam apenas uma pequena fração dos parâmetros em vez de atualizar todo o modelo. Já em MEFT (Memory-Efficient Fine-Tuning), o foco é reduzir o consumo de memória durante o treinamento, por exemplo congelando partes do modelo ou combinando fine-tuning com quantização e pruning.

Na parte de otimizações em tempo de execução, o artigo discute redução de tokens (por exemplo, eliminar tokens visuais redundantes antes da cross-attention) e técnicas de adaptação em teste, como Test-Time Adaptation e Test-Time Prompt Tuning, que ajustam o comportamento do modelo na hora da inferência sem um novo treinamento completo. Por fim, há uma seção sobre VLMs distribuídos e privacidade, apresentando cenários de Federated Learning e Split Learning, nos quais o modelo é treinado ou executado em múltiplos dispositivos sem que os dados brutos precisem sair de cada cliente.

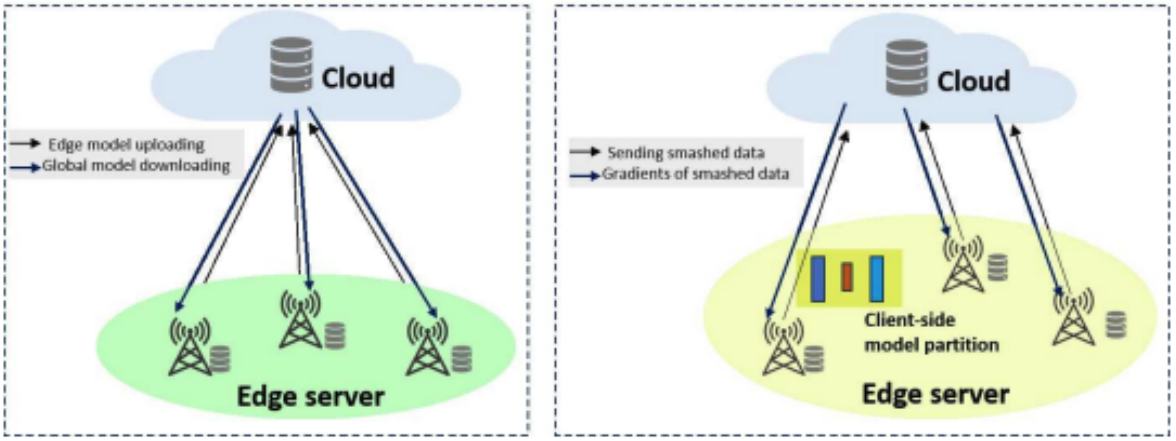


Figura 2.3. Exemplo de arquitetura distribuída para VLMs, comparando Federated Learning (atualizações de modelo) e Split Learning (divisão do modelo entre cliente e servidor).[10]

Concluo dessas leituras que a eficiência em modelos de visão-linguagem não depende de uma única técnica, mas de uma combinação de decisões, como arquitetura, tipo de quantização, nível de pruning admissível, como realizar distilação, qual estratégia de fine-tuning usar e até como o modelo será executado em produção. Esse panorama foi importante para eu começar a conectar os VLMs “clássicos” que estudei (como CLIP, ALIGN e SigLIP) com a questão prática de levá-los para dispositivos de borda, tema central do meu TCC.

Nessa Semana, também foi feito o curso de visão computacional do Hugging Face até o módulo 4, que trata de modelos multimodais. As anotações geradas ao longo do processo estão salvas nessa pasta:
https://drive.google.com/drive/folders/1qRzjgh4TW3PkONKlCIrJF3x8KYFZYwmh?usp=sharing

[10] Reproduzido de SHINDE et al. (2025)

Referências

Survey de VLMs eficientes

SHINDE, Gaurav *et al.* **A Survey on Efficient Vision-Language Models**. arXiv, , 1 jul. 2025. Disponível em: <http://arxiv.org/abs/2504.09724>. Acesso em: 3 set. 2025

Curso de visão computacional

HUGGING FACE. ***Community Computer Vision Course***. Disponível em: <https://huggingface.co/learn/computer-vision-course/unit0/welcome/welcome>. Acesso em: 2 dez. 2025.

APÊNDICE 3

Termo de Aceite de Entrega

Objetivo deste documento

Este documento faz parte do Processo da disciplina Residência em IA e tem como objetivo formalizar o aceite da entrega considerando o planejado e o realizado para o período.

Data da Reunião ("Gate") de aprovação: 17 de set. de 2025

Participantes da Entrega [matriculados em Residência em IA]:

VICTOR LUCAS SOUSA ARANTES

Entrega: [descrever a ENTREGA - requisitos e produtos gerados: links para textos, códigos, vídeos etc.]

Área de Interesse: Modelos multimodais → Modelos de visão-linguagem → Modelos de visão-linguagem compactos

Na Semana passada eu havia terminado a leitura do survey (A Survey on Efficient Vision-Language Models) e planejado para essa Semana ler alguns papers que trouxeram técnicas como LoRA e Destilação. No entanto, resolvi mudar a abordagem e em vez de voltar para a base das técnicas, pular para o que está sendo desenvolvido depois do Survey e que técnicas eles utilizam.

Nesse sentido, iniciei a construção de uma tabela com os avanços da área, papers e modelos que saíram após o estudo que li. Nessa tabela tentei classificar as tarefas que esses VLMs realizam, e preencher sobre as técnicas utilizadas usando o que li no abstract, na conclusão e em uma leitura superficial dos artigos.

Para aprofundar o estudo nas técnicas que eu mencionei anteriormente, li o artigo Vision as LoRA que explica um novo modelo de arquitetura "encoder free", adaptando um LLM para adicionar capacidades visuais por meio de LoRA, Destilação em bloco e máscara bidirecional para os tokens de visão. Minhas notas estão salvas nesse arquivo.

Planejamento: [descrever o que pretende fazer para realizar a próxima ENTREGA]

Continuar buscando por novos trabalhos na área para complementar a tabela.
Ler papers:
Efficient Vision-Language Models by Summarizing Visual Tokens into Compact Registers (Victor)
SmolVLM: Redefining small and efficient multimodal models **e** SmolVLM2
Começar a testar implementações

Observação: [caso precise fazer alguma observação, de qualquer "natureza"]

ACEITE DA ENTREGA:

CEDRIC LUIZ DE CARVALHO: Go!

Termo de Aceite de Entrega

Objetivo deste documento

Este documento faz parte do Processo da disciplina Residência em IA e tem como objetivo formalizar o aceite da entrega considerando o planejado e o realizado para o período.

Data da Reunião ("Gate") de aprovação: 24 de set. de 2025

Participantes da Entrega [matriculados em Residência em IA]:

VICTOR LUCAS SOUSA ARANTES

Entrega: [descrever a ENTREGA - requisitos e produtos gerados: links para textos, códigos, vídeos etc.]

Área de Interesse: Modelos multimodais → Modelos de visão-linguagem → Modelos de visão-linguagem compactos

Nessa Semana, assim como no planejamento passado, li o paper Victor: Efficient Vision-Language Models by Summarizing Visual Tokens into Compact Registers, que busca a eficiência condensando tokens visuais em register tokens aprendíveis e descartando os tokens originais nas primeiras camadas do modelo. Um método que parece "simples de implementar", porém não encontrei nada a nível de código para olhar com mais atenção, além do pseudo-código do paper. Notas

Além disso, também fiz a leitura do artigo SmolVLM: Redefining small and efficient multimodal models. Achei muito interessante essa filosofia chamada por eles de "small-first", que foca em reprojetar a arquitetura desde o início para compactos. Eles utilizam técnicas de compressão de imagem na entrada (pixel shuffle, image splitting) e um tuning "cuidadoso" (tokens aprendidos, pouco Chain-of-Thought, balanceamento de dados). Notas

Já no SmolVLM2, como não saiu paper, apenas li o blog e interagi com uma das versões pelo colab.

Depois da leitura do SmolVLM, fiquei com curiosidade sobre as estratégias de "reasoning multimodal" que podem ser aplicadas nos VLMs. Além de conhecer o Open VLM Leaderboard.

Planejamento: [descrever o que pretende fazer para realizar a próxima ENTREGA]

Pesquisar artigos que tratam sobre reasoning multimodal em modelos compactos e escolher mais 2 para ler.
Ler os artigos
- Machine Mental Imagery: Empower Multimodal Reasoning with Latent Visual Tokens
- BlueLM-V-3B: Algorithm and System Co-Design for Multimodal Large Language Models on Mobile Devices

Observação: [caso precise fazer alguma observação, de qualquer "natureza"]

ACEITE DA ENTREGA:

CEDRIC LUIZ DE CARVALHO: Go!

Instituto de Informática (INF) - UFG
Alameda Palmeiras, Quadra D, Câmpus Samambaia
Fone: (62) 3521-1181 / Fax: (62) 3521-1182

Mapeamento Inicial do Ecossistema de VLMs Compactos

Com a área de interesse em Modelos de Visão-Linguagem Compactos já definida, nas Semanas 3 e 4 foquei em mapear o que estava surgindo de mais recente após os surveys que eu havia lido. Em vez de voltar apenas para a base teórica de técnicas como LoRA e destilação, optei por levantar quais modelos novos tinham sido propostos e que tipos de soluções de eficiência vinham aparecendo na prática.

Como ponto de partida, organizei uma tabela de papers e modelos em uma planilha, tentando reunir os principais VLMs compactos que encontrei. Nessa tabela, para cada trabalho registrei o nome do modelo, ano, tamanho aproximado (número de parâmetros), tipo de tarefa principal (por exemplo, VQA, captioning, vídeo, documento, OCR), arquitetura adotada (dual-encoder, self-attention unificado, encoder-free), e as técnicas de eficiência utilizadas (compressão de tokens visuais, compressão de imagem, balanceamento encoder–LM, uso de encoders menores, distilação, quantização, LoRA etc.). Também marquei observações rápidas sobre o foco de cada artigo (por exemplo, “compressão agressiva de tokens”, “arquitetura small-first”, “LLM adaptado para visão sem encoder dedicado”). Essa tabela pode ser consultada no link a seguir https://docs.google.com/spreadsheets/d/15fYGYHkCoTvQZU6LZCZw7qqMWnE67EklkGR5B5SABCY/edit?usp=sharing.

Em seguida, aprofundei a leitura de alguns desses trabalhos, começando pelo artigo Vision as LoRA (VoRA), que propõe uma arquitetura “encoder free” para transformar um LLM em um MLLM sem utilizar um encoder de visão externo. Em vez de ter uma torre de visão separada, o VoRA adiciona camadas LoRA específicas de visão nos primeiros blocos do próprio LLM, congelando os pesos originais de linguagem e treinando apenas as matrizes de baixo rank e uma camada de embedding visual. A ideia é que essas LoRAs passem a “codificar” a modalidade de visão, permitindo que, após o treinamento, elas sejam fundidas de volta nos pesos originais, de modo que, na inferência, reste apenas o LLM ajustado mais o embedding visual, sem aumento de complexidade estrutural.

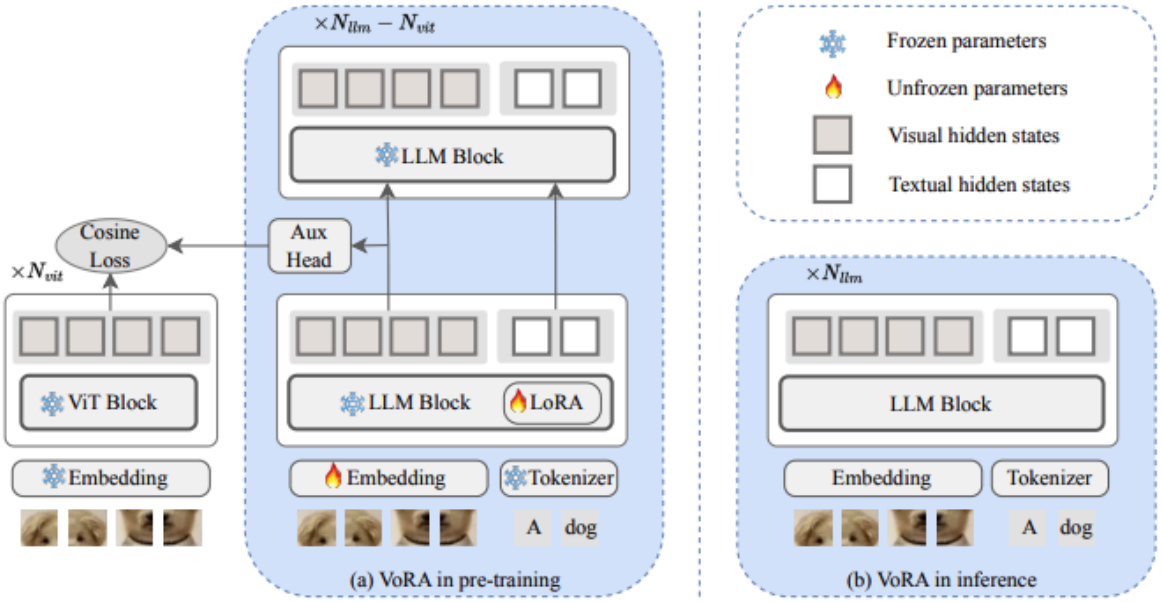


Figura 3.1. Esquema do pré-treinamento do VoRA, em que um ViT atua como professor: suas representações são alinhadas bloco a bloco com as camadas iniciais do LLM enriquecidas com LoRA, via destilação.[11]

Durante o pré-treino, o VoRA utiliza uma estratégia de *block-wise distillation* na qual as ativações intermediárias dos primeiros blocos do LLM são forçadas a se alinhar com as ativações correspondentes de um ViT professor. Isso acelera o aprendizado visual e reduz a necessidade de grandes quantidades de dados puramente de visão. Outra decisão importante é a adoção de uma máscara de atenção bidirecional apenas para os tokens de visão (enquanto os tokens de texto permanecem com máscara causal). A justificativa é que não faz sentido o LLM “prever o próximo patch de imagem”, em vez disso, é mais eficiente permitir que todos os patches visuais se vejam mutuamente para capturar melhor o contexto espacial.

[11] Reproduzido de WANG, Han *et al. (2025)*

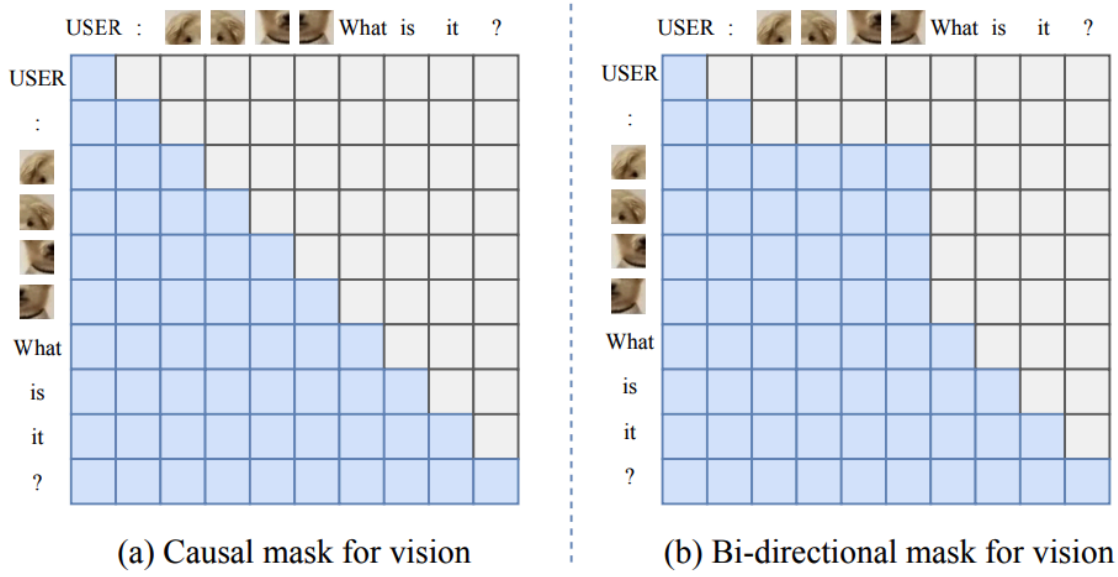


Figura 3.2. Comparação entre máscara causal e máscara bidirecional para tokens de visão no VoRA; a segunda permite que todos os patches visualizem uns aos outros, melhorando a compreensão global da imagem.[12]

O artigo também discute questões de dados. Para preservar a capacidade de seguir instruções, os autores misturam dados de imagem-texto com dados de instrução puramente textuais, e complementam lacunas de conhecimento visual (por exemplo, marcos geográficos) com um subconjunto do Google Landmarks Dataset v2, reconhecendo limitações em domínios como celebridades e obras de arte. Nos benchmarks, o VoRA mostra que um LLM enriquecido com LoRA de visão consegue competir com arquiteturas tradicionais baseadas em encoder, com menor custo computacional e maior flexibilidade de resolução.

Referências

WANG, Han *et al.* **Vision as LoRA**. arXiv, , 26 mar. 2025. Disponível em: <http://arxiv.org/abs/2503.20680>. Acesso em: 17 set. 2025

[12] Reproduzido de WANG, Han *et al. (2025)*

Arquiteturas Encoder-Free e Compactação via Tokens de Registro

Depois do VoRA, foquei em um segundo tipo de abordagem de eficiência: condensar tokens visuais. O artigo Efficient Vision-Language Models by Summarizing Visual Tokens into Compact Registers (Victor) propõe introduzir um pequeno conjunto de *register tokens* aprendíveis que resumem a informação visual nas primeiras camadas do modelo. A arquitetura segue o estilo LLaVA: um encoder de visão (por exemplo, CLIP ViT-L) projeta a imagem em tokens visuais, esses tokens recebem ao final um conjunto de *registers*, e o LLM processa tudo junto nas primeiras camadas. A cada passo, a atenção é incentivada a "transferir" a informação visual dos muitos tokens originais para poucos registers, após k camadas, os tokens de visão são descartados e apenas os *registers* (mais os tokens de texto) seguem até o fim da rede.

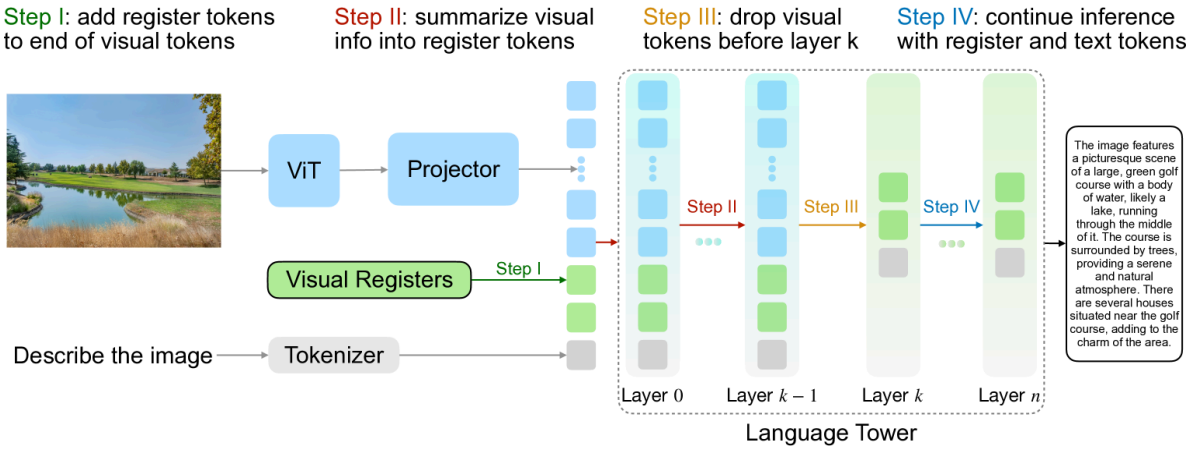


Figura 3.3. Visão geral do método Victor, mostrando as quatro etapas: anexar registers após os tokens visuais, usar as primeiras camadas do LLM para condensar a informação, descartar os tokens visuais e continuar a inferência apenas com registers e texto.[13]

Nos experimentos, os autores mostram que, com apenas 8 registers visuais (cerca de 1% do número original de tokens de visão), o modelo tem queda inferior a 4% de acurácia,

[13] Reproduzido de WEN, Yuxin *et al. (2024)*

enquanto reduz em aproximadamente 43% o tempo total de treino e aumenta em 3,3× o throughput de inferência, em comparação com o modelo base. O trabalho compara o método com outros resamplers (como FastV e Perceiver Resampler), destacando que o Victor mantém desempenho mais estável mesmo quando o número de tokens é reduzido de forma agressiva. Como limitações, os autores apontam que compressões muito fortes podem prejudicar tarefas que exigem localização precisa (como OCR e contagem de objetos) e que o método ainda não é “training-free”, pois depende de um treinamento específico para aprender a usar os registers.

Por fim, estudei o artigo SmolVLM: Redefining small and efficient multimodal models, que apresentou uma família de VLMs compactos (SmolVLM-256M, SmolVLM-500M e SmolVLM-2.2B) pensados desde o início com a filosofia *small-first*, em vez de serem apenas versões “encolhidas” de modelos grandes. Os autores exploram combinações entre três variantes do SmolLM2 (135M, 360M e 1,7B parâmetros) e duas variantes do encoder SigLIP (93M e 400M), mostrando que, para modelos pequenos, faz diferença balancear melhor a proporção de parâmetros entre encoder e LLM. Além disso, em muitos casos, um LLM proporcionalmente maior com um encoder de visão mais compacto traz melhor *trade-off* de eficiência.

Um ponto importante é a extensão da janela de contexto e a forma de lidar com o número de *tokens* visuais. Uma única imagem 512×512 com SigLIP-B/16 já gera 1024 tokens, então os autores aumentam o limite de contexto para 8k ou 16k tokens, ajustando os parâmetros de RoPE (Rotary Positional Encoding) e misturando dados de sequência longa e curta. Em paralelo, aplicam técnicas de compressão de *tokens*, como *pixel shuffle* (que reorganiza a imagem, reduzindo a resolução espacial e aumentando a profundidade de canal) e *image splitting* (dividir imagens de alta resolução em sub imagens, combinadas com uma versão reduzida da original), para reduzir o custo da *self-attention* sem perder muita informação visual.

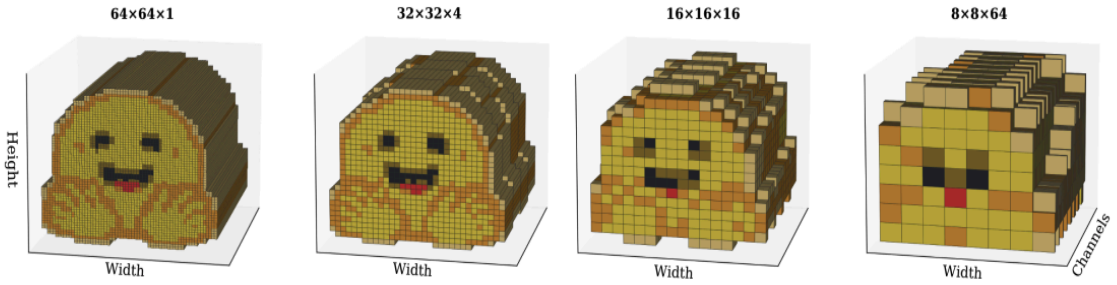


Figura 3.4. Exemplo de pixel shuffle utilizado no SmolVLM, reduzindo a resolução espacial da imagem e aumentando a profundidade de canal, o que ajuda a diminuir o número de tokens visuais preservando a densidade de informação.[14]

O trabalho também mostra que *small VLMs* se beneficiam de compressões mais agressivas (por exemplo, fatores de compressão maiores em *pixel shuffle*), desde que o tipo de tarefa não dependa de detalhes espaciais tão finos. Outra linha interessante é o cuidado com os componentes de *reasoning*: o uso de *tokens* posicionais aprendidos para as sub imagens, *system prompts* específicos para visão e vídeo, *tokens* de entrada e saída de mídia (intro/outro) para marcar o conteúdo visual, e uma quantidade controlada de dados de Chain-of-Thought (CoT), pois os autores observam que muito CoT pode piorar a performance de modelos compactos.

Nos resultados, o SmolVLM-256M consegue rodar com menos de 1 GB de memória de GPU durante a inferência, enquanto o SmolVLM-2.2B atinge desempenho competitivo com VLMs muito maiores consumindo o dobro de VRAM, inclusive em tarefas de vídeo. Isso reforça a mensagem de que número de parâmetros não é o único fator determinante de custo computacional, decisões de arquitetura (tipo de encoder, compressão de tokens, contexto, composição de dados) têm impacto direto no que é ou não possível rodar em dispositivos de borda. Ao longo dessas semanas, além dos artigos, consultei blogs técnicos e leaderboards específicos para VLMs (como o OpenVLM Leaderboard), o que ajudou a posicionar esses modelos compactos em relação a outros trabalhos de visão-linguagem em benchmarks atuais.

[14] Reproduzido de MARAFIOTI, Andrés *et al. (2025)*

Referências

WEN, Yuxin *et al.* **Efficient Vision-Language Models by Summarizing Visual Tokens into Compact Registers**. arXiv, , 17 out. 2024. Disponível em: <http://arxiv.org/abs/2410.14072>. Acesso em: 17 set. 2025

MARAFIOTI, Andrés *et al.* **SmolVLM: Redefining small and efficient multimodal models**. arXiv, , 7 abr. 2025. Disponível em: <http://arxiv.org/abs/2504.05299>. Acesso em: 17 abr. 2025

APÊNDICE 4

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DE GOIÁS
INSTITUTO DE INFORMÁTICA

Termo de Aceite de Entrega

Objetivo deste documento

Este documento faz parte do Processo da disciplina Residência em IA e tem como objetivo formalizar o aceite da entrega considerando o planejado e o realizado para o período.

Data da Reunião ("Gate") de aprovação: 1 de out. de 2025

Participantes da Entrega [matriculados em Residência em IA]:

VICTOR LUCAS SOUSA ARANTES

Entrega: [descrever a ENTREGA - requisitos e produtos gerados: links para textos, códigos, vídeos etc.]

Área de Interesse: Modelos de visão-linguagem compactos

Nessa Semana, assim como no planejamento passado, fiz a leitura do artigo BlueLM-V-3B: Algorithm and System Co-Design for Multimodal Large Language Models on Mobile Devices. Eles introduzem técnicas como o Relaxed Aspect Ratio Matching, que evita o aumento exagerado da resolução das imagens, o Token Downsampler para reduzir o número de tokens visuais e o Chunked Computing, que processa os tokens em blocos menores para se adaptar às limitações das NPUs. Também exploram quantização mista e uma pipeline "desacoplada". Notas

Além disso, também explorei o repositório NanoVLM, que eu havia levantado algumas Semanas atrás. Ele tem como foco ser um modelo "educacional", com uma implementação simples e legível em Pytorch. Passei por cada função e tentei entender como funcionava. Fiz o notebook que eles disponibilizam para um exemplo simplificado de treinamento train_nanoVLM.ipynb. Também tentei iniciar um treinamento igual ao recomendado no repositório, numa H100, através da plataforma Modal. Consegui fazer o início do treinamento mas não muito mais do que isso. Salvei aqui

Por fim, li também o artigo Machine Mental Imagery: Empower Multimodal Reasoning with Latent Visual Tokens, que apresenta o Mirage, um framework que permite ao modelo intercalar tokens visuais latentes com texto durante o raciocínio. Difere de modelos que geram imagens, o Mirage cria embeddings compactos que funcionam como "esboços internos" e são usados como pistas visuais no meio da cadeia de pensamento. Teve resultados em modelos compactos até melhores que no modelo principal comentado no paper. Notas

Planejamento: [descrever o que pretende fazer para realizar a próxima ENTREGA]

Continuar estudando reasoning multimodal.
Finalizar o treinamento de um NanoVLM e aprender sobre a parte de avaliação.
Complementar a minha tabela com novos papers.

Observação: [caso precise fazer alguma observação, de qualquer "natureza"]

ACEITE DA ENTREGA:

CEDRIC LUIZ DE CARVALHO: Go!

Instituto de Informática (INF) - UFG
Alameda Palmeiras, Quadra D, Câmpus Samambaia
Fone: (62) 3521-1181 / Fax: (62) 3521-1182

VLMs Compactos para Dispositivos Móveis e Raciocínio Multimodal

Aqui o foco passou a ser entender melhor como modelos multimodais podem, ao mesmo tempo, raciocinar melhor e rodar em dispositivos com recursos limitados. Para isso, li o artigo BlueLM-V-3B: Algorithm and System Co-Design for Multimodal Large Language Models on Mobile Devices, aprofundei as ideias do trabalho Machine Mental Imagery (Mirage) .

No caso do BlueLM-V-3B, o objetivo é criar um MLLM de aproximadamente 3 bilhões de parâmetros pensado desde o início para rodar em celulares, combinando um LLM de 2,7B com um encoder SigLIP de 400M parâmetros. O artigo propõe um desenho conjunto de algoritmo e sistema (co-design), em vez de olhar apenas para compressão de modelo. Entre as principais contribuições estão:

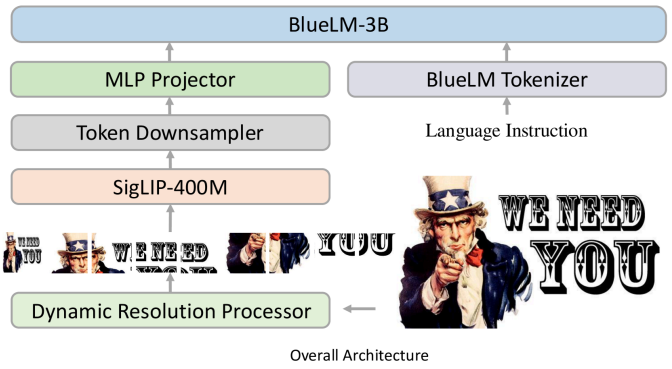


Figura 4.1. Arquitetura geral do BlueLM-V-3B, seguindo o estilo LLaVA (encoder de visão + projetor + LLM), mas com módulos adicionais de dynamic resolution, token downsampling e otimizações específicas para NPU.[15]

- Relaxed Aspect Ratio Matching, que evita inflar excessivamente a resolução das imagens e, consequentemente, o número de tokens visuais
- Token Downsampler baseado em janelas 2×2 para reduzir a sequência de tokens antes de chegar no LLM
- Chunked Computing, que processa blocos de 128 tokens por vez para se adaptar às limitações das NPUs

[15] Reproduzido de LU, Xudong *et al. (2024)*

- Quantização mista (INT4 para o LLM, INT8 para o ViT e precisões mais altas para ativações) combinado a uma pipeline “desacoplada”, em que a visão é processada o mais cedo possível para liberar memória.

Com esse conjunto de decisões, o BlueLM-V-3B atinge desempenho competitivo em benchmarks como OpenCompass, ao mesmo tempo em que consegue gerar cerca de 24 tokens/s em um chip móvel (MediaTek Dimensity 9300) usando apenas 2,2 GB de RAM.

Também fiz a leitura do aritgo Machine Mental Imagery: Empower Multimodal Reasoning with Latent Visual Tokens, que apresenta o framework Mirage. A motivação é que VLMs atuais ainda têm dificuldade em tarefas de raciocínio espacial e multimodal porque, na prática, o decoding é apenas textual: todo o raciocínio visual precisa ser “verbalizado”. O Mirage propõe que o modelo possa “pensar visualmente” gerando *tokens* visuais latentes intercalados com texto durante a cadeia de pensamento. Quando o modelo decide raciocinar visualmente, ele produz um *token* especial cujo *hidden state* é reinterpretado como um *embedding* visual compacto e reinserido no contexto, sem precisar renderizar imagens reais. O treinamento é feito em duas etapas supervisionadas (primeiro alinhando os latentes com embeddings de imagens auxiliares e depois relaxando essa restrição para focar na tarefa) seguido de uma fase de aprendizado por reforço com GRPO, usando recompensas de acurácia e de formato da resposta. Nos experimentos, o Mirage melhora de forma consistente o desempenho em benchmarks de planejamento e raciocínio espacial e, no caso de modelos menores (aproximadamente 3 bilhões de parâmetros), os ganhos relativos são de 5 a 10 pontos percentuais em algumas tarefas.

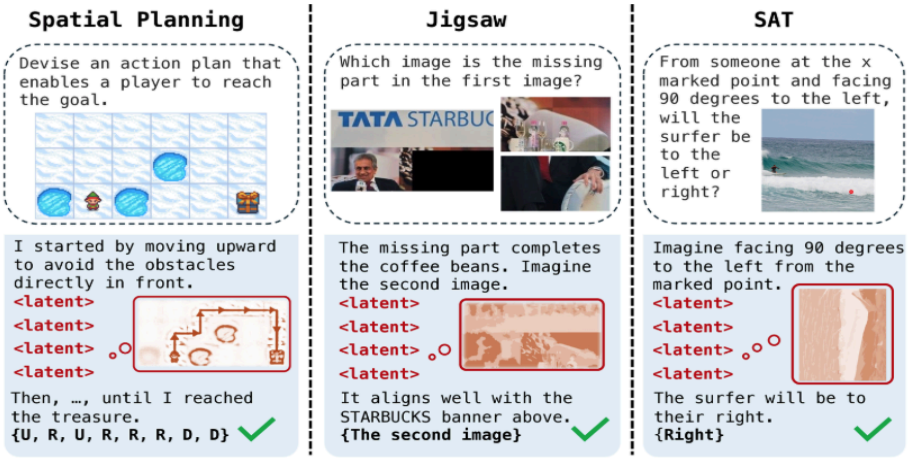


Figura 4.2. Esquema do Mirage, ilustrando como o modelo intercala tokens textuais e tokens visuais latentes ao longo da cadeia de raciocínio, sem precisar gerar imagens em pixels.[16]

Por fim, retomei o repositório NanoVLM, que tem como objetivo oferecer uma implementação educacional e legível de um VLM compacto em PyTorch. Percorri as principais funções do código, focando em entender o fluxo de dados (carregamento do dataset, construção do encoder de visão, conector e LLM) e rodei o notebook de exemplo *train_nanoVLM.ipynb*, seguindo o passo a passo sugerido pelos autores. Também iniciei um treinamento mais completo em uma GPU H100 via plataforma Modal, reproduzindo a configuração recomendada no repositório. consegui iniciar o processo, mas ainda sem concluir um treino longo com avaliação completa. Esse exercício foi importante para visualizar na prática como muitos dos conceitos estudados nos artigos (como compressão de tokens, projetores e *instruction tuning* multimodal) aparecem em um código real, servindo como ponte entre os modelos de pesquisa mais complexos e uma base compacta que posso adaptar em trabalhos futuros.

Referências

LU, Xudong *et al.* **BlueLM-V-3B: Algorithm and System Co-Design for Multimodal Large Language Models on Mobile Devices**. arXiv, , 16 nov. 2024. Disponível em: <http://arxiv.org/abs/2411.10640>. Acesso em: 24 set. 2025

YANG, Zeyuan *et al.* **Machine Mental Imagery: Empower Multimodal Reasoning with Latent Visual Tokens**. arXiv, , 20 jun. 2025. Disponível em: <http://arxiv.org/abs/2506.17218>. Acesso em: 31 ago. 2025

WIEDMANN, L.; GOSTHIPATY, A. R.; MARAFIOTI, A. **nanoVLM**. 2025. Repositório GitHub. Disponível em: <https://github.com/huggingface/nanoVLM>. Acesso em: 25 nov. 2025.

[16] Adaptado de YANG, Zeyuan *et al. (2025)*

APÊNDICE 5

Termo de Aceite de Entrega

Objetivo deste documento

Este documento faz parte do Processo da disciplina Residência em IA e tem como objetivo formalizar o aceite da entrega considerando o planejado e o realizado para o período.

Data da Reunião ("Gate") de aprovação: 8 de out. de 2025

Participantes da Entrega [matriculados em Residência em IA]:

VICTOR LUCAS SOUSA ARANTES

Entrega: [descrever a ENTREGA - requisitos e produtos gerados: links para textos, códigos, vídeos etc.]

Área de Interesse: Modelos de visão-linguagem compactos

Nessa Semana, assim como no planejamento passado, finalizei o treinamento de um NanoVLM, assim como eles descrevem no repositório. Treinei por apenas 6 horas em uma H100 e obtive resultados satisfatórios para um modelo "educacional", eles relatam uma pontuação de aproximadamente 35% no MMSTAR e eu consegui de 29%(tabelas), foi um tanto abaixo, principalmente considerando que eu usei uma versão maior do LM e do Vision Encoder, porém foi o suficiente para aprender um pouco sobre as etapas de avaliação e testar o modelo. Fiz alguns testes na T4 do colab, e a avaliação com LLM-EVAL foi feita na L4, também do colab.

Também fiz a leitura do artigo Improve Vision Language Model Chain-of-thought Reasoning, que propõe melhorar o reasoning usando apenas as respostas curtas (que tem na maioria dos datasets de VQA por exemplo). A técnica combina geração de dados de raciocínio sintéticos pelo GPT-4o e alinhamento por reforço com outcome-based DPO, que pega o acerto final para servir como feedback. Os autores liberam dataset, código e checkpoints.

No meio dos meus estudos de reasoning me deparei com poucos artigos realmente focados para modelos menores, acabei tendo também que pesquisar via youtube/tutoriais e links dos meus colegas para entender termos específicos, principalmente em relação a RLHF. Porém, alguns artigos me chamaram atenção, que foi o CogCoM (Chain-of-Manipulations Reasoning) que pelo que eu consegui ler no abstract propõe uma nova forma de raciocínio para VLMs, na qual o modelo aprende a pensar executando ações visuais em vez de apenas gerar o texto de output, ele realiza manipulações interativas na imagem (como ocr, zoom, caixas, contar…). Pelo que vi não fazem o treinamento num modelo pequeno, mas me despertou curiosidade de como seria.

Outro paper que me chamou atenção e eu venho deixado passar é o VL-Mamba: Exploring State Space Models for Multimodal Learning que aplica um novo modelo de arquitetura em vez da "transformer-based", acredito que conhecer esse tipo de arquitetura deve me ajudar na minha trilha para me tornar "especialista".

Depois de conseguir treinar o meu modelo NanoVLM, assim como comentei no gate passado, fiquei interessado em tentar replicar alguma técnica que estudei. Gostaria de implementar algo com os tokens de

registro, assim como no artigo Efficient Vision-Language Models by Summarizing Visual Tokens into Compact Registers. Escolhi essa para começar pois parece não muito difícil de implementar, dado que meus conhecimentos nos frameworks da área ainda estão sendo formados, e por ser computacionalmente "barata".

Planejamento: [descrever o que pretende fazer para realizar a próxima ENTREGA]

Na próxima Semana planejo:
Ler o artigo CogCoM
Ler o artigo do VL-Mamba
Começar estudar formas de implementar tokens de registro para VLMs compactos.

Observação: [caso precise fazer alguma observação, de qualquer "natureza"]

Gostei muito de treinar um modelo, foi muito especial aprender como usar GPUs na nuvem (gastar um pouquinho de dinheiro por besteira minha rs), testar meu modelo com imagens que tirei do meu celular, e avaliar ele num benchmark.

ACEITE DA ENTREGA:

CEDRIC LUIZ DE CARVALHO: Go!

Termo de Aceite de Entrega

Objetivo deste documento

Este documento faz parte do Processo da disciplina Residência em IA e tem como objetivo formalizar o aceite da entrega considerando o planejado e o realizado para o período.

Data da Reunião ("Gate") de aprovação: **15 de out. de 2025**

Participantes da Entrega **[matriculados em Residência em IA]**:

VICTOR LUCAS SOUSA ARANTES

Entrega: **[descrever a ENTREGA - requisitos e produtos gerados: links para textos, códigos, vídeos etc.]**

Área de Interesse: Modelos de Visão-Linguagem Compactos

Na Semana passada, eu havia feito um pré-treinamento de um NanoVLM, e consegui avaliar ele em alguns benchmarks. Além de continuar meus estudos em reasoning para VLMs.

Nessa Semana, assim como no planejamento passado, eu li o artigo CogCoM: A Visual Language Model with Chain-of-Manipulations Reasoning, que propõe o conceito de Chain of Manipulations (CoM), um mecanismo que estrutura o raciocínio visual-linguístico em etapas manipuláveis. A ideia dele é permitir que o modelo faça operações sobre a imagem (como localizar, recortar, contar ou ocr) antes de chegar à resposta final. Eles criam um novo formato de dado e uma pipeline de geração com O GPT4v e alguns modelos de Grounding e OCR criando 70 mil amostras e mais 7 mil amostras de problemas de matemática/geometria/gráficos anotados manualmente. Notas

Também fiz a leitura do artigo VL-Mamba: Exploring State Space Models for Multimodal Learning, que explora a substituição da arquitetura transformers por State Space Models no domínio multimodal. O modelo usa um Mamba LLM pré treinado, uma torre de visão (SigLIP-SO ou CLIP-ViT) e um MultiModal Connector (MMC) com o módulo Vision Selective Scan (VSS), que serve para converter sequências bidimensionais não causais (imagens) em representações lineares adequadas ao processamento sequencial do SSM. O artigo apresenta bons resultados, competitivos até com modelos maiores, mas não colocam o código, hiperparâmetros nem os datasets de treinamento, tornando muito difícil reproduzir, além de que o artigo mesmo mostra que o custo computacional para treinar é muito alto (8 A800). Notas

No planejamento passado também tinha citado que ia começar a procurar formas de implementar a ideia do artigo Efficient Vision-Language Models by Summarizing Visual Tokens into Compact Registers . Procurei implementar "do zero" com um smolLM2 e um SigLIP, porém estava com dificuldades para implementar funções que eu já tinha visto no repositório do NanoVLM. Por isso resolvi fazer um fork do repositório e tentar fazer a implementação por lá mesmo. Consegui implementar uma versão inicial, mas ainda falta um refinamento de algumas funções. Porém já consegui iniciar um treinamento de teste pelo colab.

Planejamento: **[descrever o que pretende fazer para realizar a próxima ENTREGA]**

Para a próxima Semana planejo:
Continuar minha implementação dos Register Tokens no NanoVLM
Caso dê certo, rodar alguns benchmarks com diferentes versões de modelos para avaliar o ganho e comparar com o NanoVLM padrão.
Ler Qwen-GUI-3B: A Lightweight Vision–Language Model for Cross-Resolution GUI Grounding
Ler ModernVBERT: Towards Smaller Visual Document Retrievers

Observação: **[caso precise fazer alguma observação, de qualquer "natureza"]**

Como não lembrava muito bem de como funciona o MAMBA vi alguns vídeos explicando e gostei desse aqui: MAMBA from Scratch: Neural Nets Better and Faster than Transformers

ACEITE DA ENTREGA:

CEDRIC LUIZ DE CARVALHO: Go!

Experimentos com NanoVLM e Avaliação em Benchmarks Multimodais

Finalizei o pré-treinamento e a avaliação de um NanoVLM seguindo a receita proposta pelos autores do modelo. Mantive a configuração padrão sugerida no código, com um tempo de treinamento reduzido em relação ao relatório original, mas suficiente para percorrer todo o fluxo: preparação de dados, pré-treinamento multimodal, checagem de overfitting e execução dos scripts de avaliação automática. Ao final, consegui testar o modelo tanto em modo interativo (fazendo perguntas sobre imagens) quanto em benchmarks padronizados.

Figura 5.1. Resposta do NanoVLM, mostrando que mesmo com pouco tempo de treinamento ele já adquiriu capacidades de OCR.

Para avaliar a qualidade multimodal, utilizei dois conjuntos de métricas: MME e MMSTAR, ambos amplamente usados em leaderboards de VLMs. O MME separa a pontuação em *Perception*, que mede a capacidade do modelo de reconhecer e descrever corretamente elementos visuais (objetos, atributos, relações básicas), e *Cognition*, que foca em aspectos mais “cognitivos”, como interpretação de cenas, compreensão de regras simples e raciocínio com base no contexto. Nessa avaliação, o NanoVLM que treinei alcançou um MME Perception Score de 627,78 e um MME Cognition Score de 207,14, valores compatíveis com o que se espera de um modelo compacto com poucas horas de pré-treino, ainda distante de grandes VLMs, mas já suficiente para análises qualitativas.

Já o MMSTAR organiza as tarefas em categorias mais detalhadas, o que ajuda a entender em quais tipos de raciocínio o modelo se sai melhor ou pior. A média geral obtida foi de 29,41% de acurácia, com variação entre os subconjuntos mostrados na tabela a seguir:

Métrica	Acurácia (%)
Média Geral	29.41
Percepção Grosseira	36.01
Raciocínio Lógico	35.87
Ciência e Tecnologia	30.58
Raciocínio de Instância	26.53
Percepção Fina	24.27
Matemática	23.23

Figura 5.2. Tabela de métricas no MMSTAR do NanoVLM treinado

Na minha leitura, esses números indicam que o NanoVLM lida relativamente melhor com perguntas sobre a estrutura geral da cena e raciocínios lógicos mais simples, mas ainda tem dificuldade em tarefas que exigem contagem precisa, leitura fina de detalhes visuais ou manipulação de expressões matemáticas.

Para situar esses resultados no cenário atual, consultei o OpenCompass Multimodal Leaderboard, que reúne diferentes modelos (de compactos a de grande porte) avaliados em MME, MMSTAR e outros benchmarks. Esse leaderboard pode ser acessado pelo link: https://rank.opencompass.org.cn/leaderboard-multimodal. A partir dessa comparação, foi possível observar que o meu NanoVLM fica abaixo dos modelos de referência de grande porte, mas se aproxima de outras arquiteturas compactas com objetivos parecidos, o que reforça o papel desse experimento como um passo de entendimento prático e não como tentativa de alcançar o estado da arte. Essa combinação de pré-treinamento, avaliação em benchmarks e testes interativos foi importante para consolidar conceitos estudados nos apêndices anteriores (como compressão, balanceamento *encoder*–LM e instrução multimodal) e preparar o terreno para as semanas seguintes, nas quais aprofundei o estudo de métodos de raciocínio multimodal e comecei a explorar modificações arquiteturais mais específicas, como o uso de *tokens* de registro.

Em paralelo, fiz a leitura do artigo Improve Vision Language Model Chain-of-thought Reasoning, que propõe melhorar o raciocínio de VLMs mesmo quando os datasets disponíveis trazem apenas respostas curtas (como é comum em VQA). A ideia central é aproveitar esses rótulos finais para construir cadeias de pensamento sintéticas: primeiro, os autores usam o GPT-4o para gerar explicações passo a passo a partir de pares imagem–pergunta–resposta, criando versões “com CoT” de bases originalmente enxutas; em seguida, aplicam um esquema de alinhamento por reforço chamado outcome-based

DPO, no qual diferentes cadeias de raciocínio são comparadas usando apenas o acerto ou erro da resposta final como *feedback*. Dessa forma, o modelo aprende a preferir trajetórias de raciocínio que levam ao resultado correto, sem depender de anotações humanas detalhadas. O trabalho também disponibiliza código, *checkpoints* e o conjunto de dados gerado, o que abre caminho para futuras reproduções e adaptações em cenários de VLMs compactos.

Referências

ZHANG, Ruohong *et al.* Improve Vision Language Model Chain-of-thought Reasoning. *In*: CHE, Wanxiang *et al.* (orgs.). ACL 2025. **Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)**. Vienna, Austria: Association for Computational Linguistics, jul. 2025. Disponível em: <https://aclanthology.org/2025.acl-long.82/>. Acesso em: 8 out. 2025

Raciocínio Multimodal Baseado em Ações e Arquiteturas Alternativas

Continuei as leituras sobre raciocínio multimodal, agora focando em trabalhos que estruturam explicitamente os passos de solução visual e em propostas que substituem o *backbone Transformer* por modelos de espaço de estados.

Um dos artigos estudados foi o CogCoM, que introduz o conceito de Chain of Manipulations (CoM), um mecanismo em que o VLM passa a resolver problemas visuais como uma sequência de manipulações sobre a imagem (por exemplo, OCR, grounding, crop/zoom in, contagem, traçar linhas auxiliares). Em vez de ir direto da imagem para a resposta, o modelo é treinado para produzir uma cadeia de passos intermediários, cada um com uma evidência visual associada. Para isso, os autores constroem uma pipeline de dados que combina o GPT-4 como “anotador linguístico” (gerando os passos em linguagem natural) com modelos de grounding e OCR (GroundingDINO e PaddleOCR), resultando em cerca de 70 mil exemplos automáticos, além de aproximadamente 7 mil amostras anotadas manualmente para problemas gráficos de matemática.

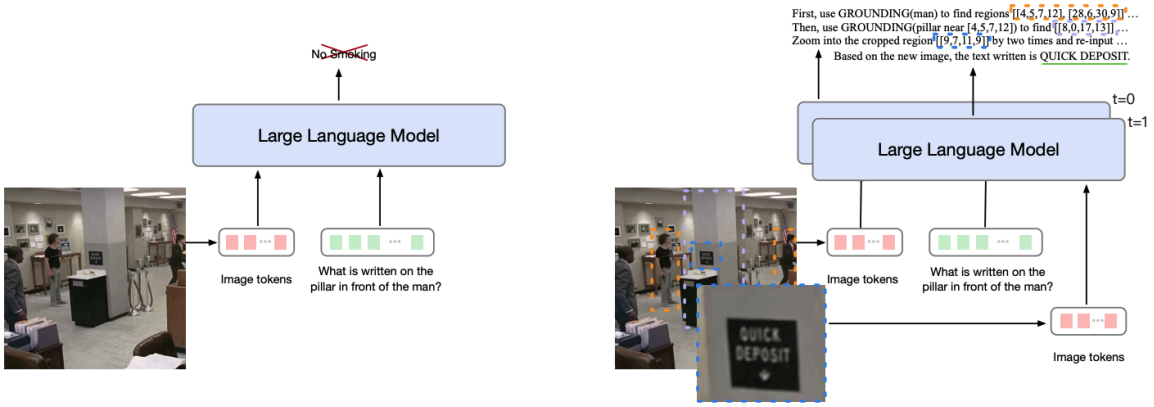


Figura 5.3. Exemplo de cadeia de manipulações no CogCoM: em vez de responder diretamente à pergunta "O que está escrito no pilar na frente do homem?", o modelo primeiro faz *grounding* do homem, depois do pilar próximo a ele, realiza *zoom* e recorte sucessivos da região relevante e reinsere a nova imagem no LLM, o que permite ler corretamente o texto "QUICK DEPOSIT", em contraste com o VLM base que responde "No Smoking".[17]

Também li o artigo VL-Mamba, que explora o uso de State Space Models (SSMs) no lugar de Transformers em modelos multimodais. A proposta é combinar um LLM baseado em Mamba com um encoder de visão do tipo ViT e um conector multimodal chamado MultiModal Connector (MMC), que incorpora o módulo Vision Selective Scan (VSS). Esse módulo faz varreduras bidirecionais ou em múltiplas direções sobre o mapa de patches da imagem, convertendo a sequência 2D não causal em uma representação 1D adequada ao processamento sequencial do SSM, sem depender de atenção quadrática. Nos experimentos, o VL-Mamba alcança desempenho competitivo com modelos Transformers de porte semelhante (e até supera versões maiores do LLaVA-1.5 em alguns benchmarks), mas o próprio artigo reconhece que o custo de treinamento ainda é alto (envolvendo múltiplas GPUs A800) e que a reprodutibilidade é limitada pela ausência de código, pesos e detalhes completos de treinamento.

Motivado tanto pelo método Victor estudado nas semanas anteriores, iniciei uma primeira tentativa de incorporar tokens de registro (register tokens) em um VLM de pequeno porte. A ideia foi adaptar o NanoVLM para que, nas primeiras camadas do modelo, os tokens visuais fossem gradualmente condensados em um pequeno conjunto de registers aprendíveis, que concentrariam a informação visual antes de prosseguir pelo restante do LLM. Inicialmente tentei implementar essa ideia "do zero" usando um SmolLM2 e um SigLIP separados, mas percebi que várias funções de data loading e de construção do modelo já estavam bem resolvidas no próprio repositório do NanoVLM. Por isso, optei por fazer um

[17] Reproduzido de QI, Ji *et al. (2024)*

fork e desenvolver a modificação diretamente ali, criando uma branch dedicada (feature/victor-register-tokens). A versão inicial do código de treino com register tokens encontra-se disponível neste repositório: https://github.com/arantesvictorl/nanoVLM/tree/feature/victor-register-tokens

Embora essa implementação ainda precise de refinamentos (especialmente na forma de descartar tokens visuais e no ajuste dos hiperparâmetros de treino), já foi possível rodar treinamentos de teste e verificar que a arquitetura se comporta de maneira estável, abrindo caminho para comparar futuramente o NanoVLM padrão com variantes que usam compressão via tokens de registro.

Referências

QI, Ji *et al.* CogCoM: A Visual Language Model with Chain-of-Manipulations Reasoning. *In*: THE THIRTEENTH INTERNATIONAL CONFERENCE ON LEARNING REPRESENTATIONS. **Anais**... 4 out. 2024. Disponível em: <https://openreview.net/forum?id=Fg0eo2AkST>. Acesso em: 8 out. 2025

QIAO, Yanyuan *et al.* VL-Mamba: Exploring State Space Models for Multimodal Learning. *In*: NEURIPS EFFICIENT NATURAL LANGUAGE AND SPEECH PROCESSING WORKSHOP. **Proceedings of The 4th NeurIPS Efficient Natural Language and Speech Processing Workshop**. PMLR, 10 dez. 2024. Disponível em: <https://proceedings.mlr.press/v262/qiao24a.html>. Acesso em: 14 out. 2025

APÊNDICE 6

Termo de Aceite de Entrega

Objetivo deste documento

Este documento faz parte do Processo da disciplina Residência em IA e tem como objetivo formalizar o aceite da entrega considerando o planejado e o realizado para o período.

Data da Reunião ("Gate") de aprovação: **23 de out. de 2025**

Participantes da Entrega **[matriculados em Residência em IA]**:

VICTOR LUCAS SOUSA ARANTES

Entrega: **[descrever a ENTREGA - requisitos e produtos gerados: links para textos, códigos, vídeos etc.]**

Área de Interesse: Modelos de Visão-Linguagem Compactos

Ao longo das Semanas da Residência em IA, iniciei meus estudos com artigos basilares da área de modelos multimodais, o que me levou a aprofundar o interesse por modelos de visão-linguagem (VLMs). Dentro desse tema, tenho focado em compreender arquiteturas mais eficientes e compactas, analisando como diferentes abordagens lidam com a redução de custo e preservação de desempenho. Paralelamente, venho estudando técnicas aplicadas a VLMs, como reasoning, knowledge distillation e LoRA, e mais recentemente tenho buscado papers voltados a aplicações práticas, para entender como esses conceitos são utilizados em contextos reais.

Nessa Semana, assim como no planejamento passado, eu continuei tentando implementar a técnica de resumir os tokens de visão em tokens de registro nos primeiros blocos do language model. Tentei diferentes abordagens:

- Backbones diferentes
- Diferentes quantidades de tokens de registro
- Alterar em qual bloco os tokens de registro devem ser retirados

Infelizmente ainda não consegui fazer o modelo convergir com essas abordagens. Report
Também tive muita dificuldade em adaptar o código para o repositório do NanoVLM, principalmente por ele ser todo em Pytorch e bem minimalista.

Como essa técnica de resumir em tokens de registro não possui o código divulgado pelo paper, estou reconsiderando a minha ideia de replicar esse paper e partir para tentar replicar a técnica do paper Vision as LoRA, que é uma maneira "encoder-free" de treinar um modelo multimodal (não apenas para VLMs). Eles têm o código e datasets divulgados, e mesmo que replicar o paper completamente seja muito custoso computacionalmente, acredito que seria possível mostrar ganhos em modelos ainda menores do que o tratado no paper, reduzindo um pouco desse custo.

Em paralelo a isso, li o paper Qwen-GUI-3B: A Lightweight Vision-Language Model for Cross-Resolution GUI Grounding, que foca mais na aplicação prática dos VLMs compactos. Eles trabalham na tarefa de GUI Grouding, que busca localizar elementos na interface (mobile, web, desktop) a partir de comandos em linguagem natural. Eles construíram um dataset e também disponibilizaram o código para replicar o paper em uma GPU RTX 4090. Notas

Planejamento: [descrever o que pretende fazer para realizar a próxima ENTREGA]

Para a próxima Semana planejo:
Começar minha implementação do Vision as LoRA
Ler SmolDocling: An ultra-compact vision-language model for end-to-end multi-modal document conversion
Buscar mais artigos que tratam de aplicação prática e tentar replicar algum desses artigos

Observação: [caso precise fazer alguma observação, de qualquer "natureza"]

Gostaria de agradecer ao Daniel Fazzioni, por me ajudar a entender como funciona o paper do Vision as Lora

ACEITE DA ENTREGA:

CEDRIC LUIZ DE CARVALHO: Go!

Implementação de Tokens de Registro e Primeiros Estudos em GUI Grounding

Dei prosseguimento no estudo conceitual de técnicas de compactação para uma tentativa prática de aplicá-las em um modelo pequeno. O objetivo principal foi adaptar a ideia de Visual Compact Token Registers (Victor), que resume um grande número de tokens de visão em poucos *register tokens* nas primeiras camadas do modelo de linguagem, ao NanoVLM, um VLM compacto.

Para isso, modifiquei o código do NanoVLM inserindo tokens de registro logo após os tokens visuais e implementando blocos iniciais responsáveis por condensar a informação visual nesses registros, descartando os demais tokens de visão nas camadas subsequentes. Testei variações no número de tokens de registro e em quais camadas essa condensação ocorreria, ajustando hiperparâmetros de treinamento (taxa de aprendizado, *batch size* e número de *steps*). Contudo, os treinos não convergiram de forma satisfatória: a perda estabilizava em valores altos e o desempenho qualitativo em exemplos de teste permanecia fraco.

Diante dessas dificuldades, optei por deslocar o foco para entender melhor uma aplicação concreta para VLMs compactos, escolhendo a tarefa de GUI Grounding. Nessa tarefa, o modelo recebe uma captura de tela e uma instrução em linguagem natural (por exemplo, “clique no botão Enviar”) e deve produzir coordenadas ou caixas delimitadoras que identifiquem o elemento correto da interface.

Como referência, estudei o paper ZonUI-3B: A Lightweight Vision-Language Model for Cross-Resolution GUI Grounding, que utiliza o modelo Qwen2.5-VL-3B como backbone e realiza um *fine-tuning* específico para GUI Grounding. Os autores constroem um corpus de cerca de 24k exemplos de interfaces móveis, web e desktop e adotam uma estratégia de treinamento em duas etapas:

- na primeira etapa, o modelo é ajustado em dados multi-plataforma para aprender de forma geral a relacionar linguagem e elementos de interface;

- na segunda etapa, é feito um fine-tuning especializado em capturas de tela de maior resolução, com foco em cenários de desktop e interfaces mais densas.

Todo o processo é implementado via LoRA sobre o backbone de 3B parâmetros e pode ser reproduzido em uma única GPU RTX 4090, o que torna o método particularmente interessante para estudos acadêmicos e para comparação com modelos menores. Nos benchmarks ScreenSpot, ScreenSpot-v2 e ScreenSpot-Pro, o ZonUI-3B atinge acurácias na faixa de 85% em ScreenSpot e 86% em ScreenSpot-v2, superando outros modelos abaixo de 4B parâmetros e aproximando-se do desempenho de modelos bem maiores.

Referências

HSIEH, ZongHan; WEI, Tzer-Jen; YANG, ShengJing. **ZonUI-3B: A Lightweight Vision-Language Model for Cross-Resolution GUI Grounding**. Disponível em: <https://arxiv.org/abs/2506.23491v3>. Acesso em: 3 dez. 2025.

APÊNDICE 7

Termo de Aceite de Entrega

Objetivo deste documento

Este documento faz parte do Processo da disciplina Residência em IA e tem como objetivo formalizar o aceite da entrega considerando o planejado e o realizado para o período.

Data da Reunião ("Gate") de aprovação: 6 de nov. de 2025

Participantes da Entrega [matriculados em Residência em IA]:

VICTOR LUCAS SOUSA ARANTES

Entrega: [descrever a ENTREGA - requisitos e produtos gerados: links para textos, códigos, vídeos etc.]

Área de Interesse: Modelos de Visão-Linguagem Compactos

Durante as Semanas anteriores foram realizadas as seguintes atividades:

- Leitura de artigos e Surveys de modelos multimodais.
- Estudo para relembrar conceitos importantes de visão computacional
- Leitura de artigos e Surveys para a área de modelos de Visão-Linguagem Compactos focando nas arquiteturas
- Foco em papers que tratavam de técnicas de destilação, LoRA, compactação de tokens e reasoning multimodal
- Treinamento de um NanoVLM
- Testes para adaptar técnica de resumo de tokens no NanoVLM
- Pesquisa com foco nas aplicações de VLMs compactos

Nessa Semana, foquei em replicar o paper ZonUI-3B: A Lightweight Vision-Language Model for Cross-Resolution GUI Grounding, que faz um fine tuning do modelo Qwen2.5-VL-3B para a tarefa de GUI Grounding com um dataset pequeno (24K amostras) em 48 horas numa GPU RTX 4090 24GB. Os autores disponibilizam o código e os datasets. Fiz a avaliação nos benchmarks e consegui reproduzir os resultados do paper. Report

Como o modelo base que eles utilizaram neste paper foi pré-treinado para ter capacidades de Grounding, o dataset pequeno, adaptado para interfaces gráficas, melhorou consideravelmente a performance. Porém, ao pensar em formas de acrescentar ao paper, eu percebi que os modelos menores não são treinados para esse tipo de tarefa, e que para tentar reproduzir, teria que ou **buscar um modelo maior**, que fosse treinado para esse tipo de tarefa, ou **treinar um modelo pequeno para aprender a gerar as coordenadas.**

Pesquisando melhor, encontrei esse repositório que ensina a fazer um Fine Tuning de um modelo Gemma para detecção de objetos. Tive muitos erros para rodar esse treinamento e por ter achado um outro repositório mais adaptado para o que eu queria, resolvi deixar de lado.

Me deparei com o Smol2Operator: Post-Training GUI Agents for Computer Use, que treina um SmolVLM2 2.2B para aprender a gerar coordenadas e conseguir ter capacidades de raciocínio para realizar tarefas de

mais de uma etapa. Estou replicando a primeira parte da pipeline, que é de treinar o modelo para gerar coordenadas. Fiz alguns treinamentos com modelos de 256 e 500 milhões de parâmetros e enquanto ia rodando o treinamento fiz esse colab para ir testando os checkpoints.

Encontrei o modelo Vocaela de 500 milhões de parâmetros, que parece ter usado a mesma pipeline e teve bons resultados nos benchmarks de Grounding, mas que pela capacidade do VLM para as tarefas de múltiplas etapas, não se saiu bem.

Planejamento: [descrever o que pretende fazer para realizar a próxima ENTREGA]

Realizar mais experimentos

- **Rodar o modelo de 256M nos benchmarks (ScreenSpot e ScreenSpot V2)**
- **Testar modelos maiores com capacidade de Grounding e tentar treinar eles para tarefas de mais de uma etapa.**
- **Testar em dispositivo móvel**

Observação: [caso precise fazer alguma observação, de qualquer "natureza"]

Wandb não funciona na Rússia

ACEITE DA ENTREGA:

CEDRIC LUIZ DE CARVALHO: Go!

Reprodução do ZonUI-3B e Fine-Tuning de VLMs Compactos para GUI Grounding

Com a tarefa de GUI Grounding já estudada conceitualmente, concentrei-me em reproduzir o ZonUI-3B na prática e em investigar se pipelines semelhantes poderiam ser adaptadas para VLMs ainda menores.

O primeiro passo foi a reprodução do artigo ZonUI-3B: A Lightweight Vision-Language Model for Cross-Resolution GUI Grounding. Utilizando o código e os *scripts* disponibilizados pelos autores, executei o *fine-tuning* em duas etapas sobre o Qwen2.5-VL-3B-Instruct, seguindo a mesma divisão entre treino geral multi-plataforma (Stage 1) e adaptação para alta resolução (Stage 2). Em seguida, avaliei o modelo nos benchmarks ScreenSpot e ScreenSpot-v2, replicando o protocolo original. Os resultados foram organizados em uma tabela comparando o modelo base Qwen2.5-VL-3B-Instruct sem *fine-tuning* , o modelo após o Stage 1, o modelo após o Stage 2 e os valores reportados no *paper*.

Run / Modelo	Avg	Mobile/Text	Mobile/Icon	Desktop/Text	Desktop/Icon	Web/Text	Web/Icon
Qwen/Qwen2.5-VL-3B-Instruct	65.34%	82.05%	60.26%	76.29%	45.71%	74.35%	53.40%
Stage 1	81.71%	91.58%	83.41%	90.72%	66.43%	84.35%	73.79%
Stage 2	84.22%	95.97%	80.79%	90.72%	72.86%	88.26%	76.70%
ZonUI-3B (paper)	84.9%	96.3%	81.6%	93.8%	74.2%	89.5%	74.2%
ScreenSpot							
Qwen/Qwen2.5-VL-3B-Instruct	69.80%	86.90%	69.19%	78.35%	48.57%	75.21%	60.59%
Stage 1	83.41%	93.45%	85.31%	91.75%	68.57%	85.04%	76.35%
Stage 2	85.40%	97.59%	81.99%	93.30%	72.86%	89.32%	77.34%
ZonUI-3B (paper)	86.4%	97.9%	84.8%	93.8%	75.0%	91.0%	75.8%
ScreenSpot V2							

Figura 7.1 – Acurácia (%) nos benchmarks ScreenSpot e Screenspot V2 para o modelo base Qwen2.5-VL-3B-Instruct, após as etapas de fine-tuning (Stage 1 e Stage 2) e para o ZonUI-3B reportado no artigo, discriminando o desempenho médio (Avg) e por cenário (Mobile, Desktop, Web) e tipo de alvo (Text/Icon).

De forma geral, o *fine-tuning* em duas etapas elevou a acurácia global de cerca de 65–70% (modelo base) para 84–85% em ScreenSpot e 83–85% em ScreenSpot-v2, ficando muito próximo dos números originais do artigo (84,9% e 86,4%, respectivamente). Além da média geral, foram analisadas as divisões por plataforma e tipo de alvo (Mobile/Desktop/Web × Text/Icon), que mostraram ganhos consistentes em quase todas as categorias.

A partir dessa reprodução bem-sucedida, a questão passou a ser como aproximar esse tipo de pipeline de modelos realmente compactos. Nesse contexto, estudei o trabalho Smol2Operator: Post-Training GUI Agents for Computer Use, que treina um SmolVLM2 de 2,2B parâmetros para receber instruções em linguagem natural e gerar diretamente coordenadas de interação em interfaces, servindo como base para agentes capazes de executar tarefas de múltiplas etapas em ambientes gráficos.

O método ajusta um VLM, que, inicialmente, não tem capacidades de *Grouding* para prever coordenadas de ação (por exemplo, cliques em botões ou campos de texto) a partir de pares imagem–instrução, em etapas posteriores, são incorporadas habilidades de raciocínio e decomposição de tarefas, permitindo que o agente navegue pela interface em sequências mais longas.

Inspirado por essa abordagem, adaptei a primeira etapa da pipeline do Smol2Operator para modelos menores da família SmolVLM, realizando experimentos com SmolVLM-256M e SmolVLM2-500M, em vez do modelo de 2,2B parâmetros utilizado no trabalho original. Os treinos foram monitorados em dashboards do Weights & Biases, permitindo comparar velocidade de treinamento, número de tokens processados, perda de validação e acurácia média por token entre as diferentes configurações.

A figura a seguir ilustra um desses conjuntos de curvas:

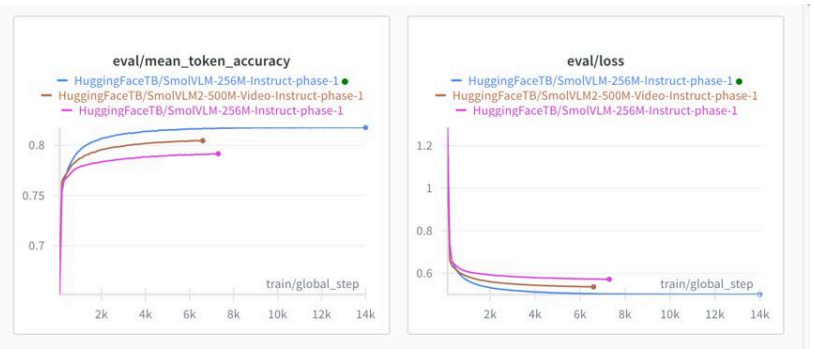


Figura 7.2. Curvas de treinamento de variantes compactas do SmolVLM (256M e 500M de parâmetros) na etapa de instrução multimodal inspirada no Smol2Operator, mostrando, ao longo dos *global steps*, as métricas de acurácia média por token e perda de validação.

Embora os resultados ainda sejam exploratórios e inferiores aos obtidos com o modelo de 2,2B parâmetros, os experimentos indicam que é possível reutilizar parte da pipeline de agentes de GUI para VLMs na faixa de centenas de milhões de parâmetros, desde que haja ajustes na quantidade de dados, nos hiperparâmetros e na forma de avaliar a performance em tarefas multi-etapas.

Foi observado que aumentar a resolução da imagem beneficia o desempenho do modelo mais do que usar o modelo com maior número de parâmetros (linha azul na imagem 7.2 representa o modelo SmolVLM-256M com maior resolução nas imagens).

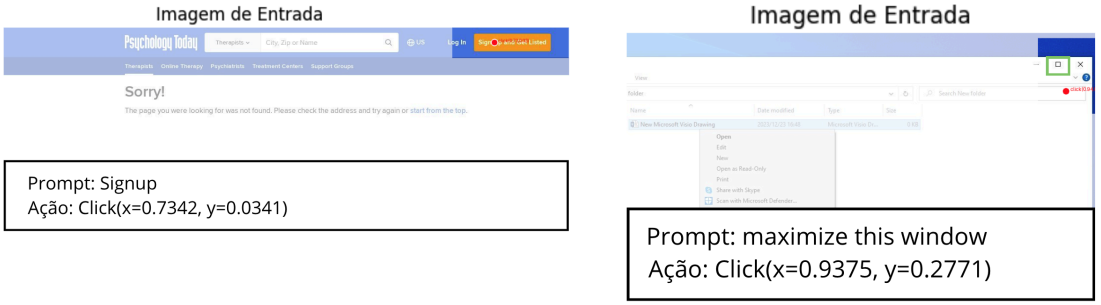


Figura 7.3. Exemplos de acerto (esquerda) e erro (direita) do SmolVLM de 256M. O ponto vermelho significa as coordenadas apontas pelo modelo.

APÊNDICE 8

Termo de Aceite de Entrega

Objetivo deste documento

Este documento faz parte do Processo da disciplina Residência em IA e tem como objetivo formalizar o aceite da entrega considerando o planejado e o realizado para o período.

Data da Reunião ("Gate") de aprovação: 12 de nov. de 2025

Participantes da Entrega [matriculados em Residência em IA]:

VICTOR LUCAS SOUSA ARANTES

Entrega: [descrever a ENTREGA - requisitos e produtos gerados: links para textos, códigos, vídeos etc.]

Área de Interesse: Modelos de Visão-Linguagem Compactos

Durante as Semanas anteriores foram realizadas as seguintes atividades:

- Leitura de artigos basilares e pesquisas (Surveys) sobre modelos multimodais.
- Leitura de artigos e pesquisas (Surveys) para a área de modelos de Visão-Linguagem Compactos analisando as arquiteturas
- Foco em papers que tratavam de técnicas de destilação, LoRA, compactação de tokens e reasoning multimodal
- Pré-treinamento de um VLM (NanoVLM)
- Pesquisa com foco nas aplicações de VLMs compactos
- Fine-tuning supervisionado de VLM

Nessa Semana, dada minha trajetória na Residência até o momento, achei que faria muito sentido que eu aprendesse como transformar VLMs em um formato em que eu possa rodar em um dispositivo de borda, no meu caso, no celular.

Voltei ao paper do BlueLM-V-3B para ver se encontrava algum tipo de código ou diga de como rodar VLMs em um celular, porém só deixaram explícito como se rodava na CPU, com llama.cpp, a implementação na NPU (Neural Processing Unit) ficou em aberto.

Busquei outras formas de implementar pelo celular, mas acabei escolhendo o llama.cpp mesmo por ser o motor de inferência mais leve. Me deparei com a necessidade de transformar meu modelo em um GGUF (GPT-Generated Unified Format). Levei um bom tempo para entender como funciona esse tipo de arquivo e "de brinde" aprendi bastante sobre métodos de Post Training Quantization (PTQ).

Consegui realizar alguns testes de VLMs pequenos (500 milhões de parâmetros) e até relativamente grandes (32 Bilhões de parâmetros).

Consolidei meus estudos aqui: Rodar VLMs no celular

Além disso, por ter conseguido resultados "baixos" no meu modelo de GUI-Grouding. Fiz bastantes testes manuais para ver como ele estava respondendo. Pegando os checkpoints do treinamento, percebi que logo

ele aprendia a formatação das respostas (ex.: click(x=0.45, y=0.12) porém as coordenadas não ficavam certas, ou ele errava por pouco. Pensando nisso, cogitei a possibilidade de fazer algum treinamento onde a recompensa fosse a distância da coordenada certa para a predita.

Comecei a montar meu dataset, como já tinha visto no paper do ZonUI-3B, peguei um dos datasets deles, feitos para SFT, e comecei a transformar para um dataset de GRPO. Tive que fazer algumas transformações nas coordenadas, para adaptar para as quais o meu modelo estava treinado, mudar os conjuntos de instruções para uma linha e sua bounding box correspondente. Salvei ele aqui.

Para o treinamento, comecei testando algumas funções de recompensa mais básicas considerando apenas a distância, mas acabou não dando certo. Fiz uma leitura rápida desse paper UI-R1: Enhancing Action Prediction of GUI Agents by Reinforcement Learning, tirei algumas ideias para melhorar a função de recompensa. Porém ainda não consegui bons resultados. Experimento GRPO GUI Grounding

Planejamento: [descrever o que pretende fazer para realizar a próxima ENTREGA]

Consolidar as entregas e documentos no meu TCC.
Continuar estudando!

Observação: [caso precise fazer alguma observação, de qualquer "natureza"]

Gostaria de agradecer ao Artur e ao Daniel Fazzioni pelas discussões e dicas sobre os experimentos com função de recompensa.

ACEITE DA ENTREGA:

CEDRIC LUIZ DE CARVALHO: Go!

Revisitei o modelo de GUI Grounding treinado nas Semanas anteriores e realizei uma bateria de testes manuais. Esses testes mostraram que o modelo havia aprendido de forma consistente o padrão textual de saída (por exemplo, click(x=0.45, y=0.12)), mas ainda cometia erros relevantes nas coordenadas finais, muitas vezes “quase acertando” o alvo. A partir dessa observação, desenvolvi um conjunto de experimentos exploratórios com Reinforcement Learning via GRPO (Generative Reward Policy Optimization), no qual a recompensa passa a depender explicitamente da distância entre o ponto clicado e o alvo correto, bem como de aspectos de formato e concisão da resposta. Para isso, adaptei um dos datasets de SFT do ZonUI-3B para um formato adequado a GRPO, reescalando bounding boxes e redefinindo instruções. O dataset tratado está disponível no Hugging Face com id *arantesvictorl/ShowUI-web-8k-grounding-grpo.*

Também direcionei o trabalho para a execução desses modelos em dispositivos de borda, em especial no celular. Partindo da trajetória construída ao longo da Residência, retomei a ideia de co-projeto entre algoritmo e sistema discutida em trabalhos como o BlueLM-V-3B e, a partir disso, explorei na prática como converter modelos de Visão-Linguagem para um formato adequado a motores de inferência leves, como o llama.cpp. Esse processo envolveu estudar o formato GGUF e técnicas de Post-Training Quantization (PTQ), entender como diferentes esquemas de quantização impactam memória e tempo de resposta e testar, no próprio aparelho, modelos desde a faixa de centenas de milhões até dezenas de bilhões de parâmetros.

Estudo e Ajustes do Treinamento GRPO para Tarefas de GUI Grounding

Contexto e Objetivo

O experimento teve como objetivo aplicar Reinforcement Learning com o método Generative Reward Policy Optimization (GRPO) para ajustar um modelo de visão-linguagem compacto (SmolVLM-256M-Instruct-GUI) na tarefa de GUI grounding, em que o modelo

precisa identificar, a partir de uma imagem (screenshot) e de uma instrução textual, o ponto de clique correspondente na interface.

O foco foi testar estratégias de modelagem da função de recompensa, mecanismos de controle de divergência (KL), e otimizações de geração, de modo a entender os limites e desafios de aplicar GRPO em modelos multimodais de pequeno porte.

Configuração Experimental

Modelo base: arantesvictorl/SmolVLM-256M-Instruct-GUI

Framework: TRL (Hugging Face)

Dataset: arantesvictorl/ShowUI-web-8k-grounding-grpo

Formato de saída esperado: click(0.XXXX, 0.YYYY), com coordenadas normalizadas entre 0 e 1.

Hardware utilizado: GPU RTX 5090 com memória de 24 GB.

O dataset foi preparado com prompts multimodais, compostos por uma imagem e um texto de instrução, formatados no padrão esperado pelo modelo (system e user roles).

Função de Recompensa e Ajustes Progressivos

A função de recompensa passou por diversas revisões para tentar equilibrar formato correto, distância espacial e compreensão de contexto.

- Recompensa de formato: detecção de saídas que seguem o padrão click(x, y).
- Recompensa de distância: penalização proporcional à distância euclidiana entre ponto previsto e ponto alvo.
- Recompensa por bounding box: bônus quando o ponto previsto está dentro da caixa delimitadora.
- Tolerância variável (τ): introdução de ruído aleatório na escala de normalização da distância.
- Sinal fraco de reforço textual: pequeno bônus se a saída contém a palavra “click”, mesmo fora do formato.

- Bônus de concisão: reforço positivo para respostas curtas e pontuais.
- Penalidade de excesso: redução de recompensa se houver texto além do fechamento “)”.

Apesar dessas modificações, o modelo ainda apresenta baixa consistência nas coordenadas previstas e recompensas médias próximas de zero em várias iterações.

Parser e Extração de Coordenadas

O parser responsável por identificar coordenadas nas respostas foi reformulado para lidar com diferentes variações de formatação (espaçamentos, maiúsculas/minúsculas, vírgulas decimais e variações no formato de saída).

Essa modificação melhorou a taxa de extração de coordenadas válidas, reduzindo a quantidade de exemplos com recompensa nula, mas ainda não foi suficiente para estabilizar o aprendizado.

Controle Dinâmico de Divergência (KL)

Foi implementado um controlador de KL (Kullback-Leibler) para ajustar dinamicamente o parâmetro β ao longo do treinamento. O ajuste automático ajudou a conter o crescimento excessivo do KL, embora o valor ainda oscile acima do ideal (entre 0.8 e 1.5) em várias etapas.

Critérios de Parada e Duração da Geração

Durante as primeiras execuções, o modelo apresentava clipped_ratio = 1.0, indicando que a geração atingia sempre o limite máximo de tokens sem encontrar um token de parada. Foram então testadas duas abordagens:

1. Critério de parada por expressão regular, interrompendo a geração assim que um click(x, y) completo é detectado.
2. Uso de stop_sequences=[")"] como alternativa simples.

Essas mudanças reduziram o tempo de geração e ajudaram a evitar sequências excessivamente longas.

Ajustes de Hiperparâmetros de Geração

- MAX_COMPLETION_LEN: 8 → Limita a resposta a apenas o formato necessário.
- TEMPERATURE: 0.5 → Reduz aleatoriedade e melhora a consistência.
- TOP_P: 0.75 → Evita amostras de baixa probabilidade.
- NUM_GENERATIONS: 3–4 → Mantém diversidade sem custo elevado.
- generation_batch_size: múltiplo de NUM_GENERATIONS → Evita erro de compatibilidade no TRL.

Eficiência Computacional

- Utilização de dataloader_num_workers=8 e pin_memory=True para otimizar I/O.
- Ajuste do tamanho de batch via gradient_accumulation_steps para uso estável de VRAM.
- Configuração de logs reduzida (logging_steps=10, save_steps=1000) para minimizar overhead.

Mesmo com essas otimizações, observou-se baixa utilização de GPU (~20%) e tempo elevado por iteração, o que indica que a etapa de geração (sampling múltiplo em GRPO) ainda é o principal gargalo.

Resultados e Comportamento Observado

- `reward_mean`: de −0.1 → ~0.07 → O modelo passou a gerar coordenadas no formato esperado, mas ainda imprecisas.
- `frac_reward_zero_std`: ↓ de 1.0 → ~0.5 → Indica aprendizado parcial do formato.

- `kl`: ~0.8–1.5 → Acima da meta, modelo ainda distante da política de referência.
- `completions/clipped_ratio`: ↓ < 0.1 → Geração interrompida corretamente.
- `entropy`: 1.0–2.0 → Nível adequado de diversidade de respostas.

Apesar da estabilização, o treinamento ainda não atingiu desempenho satisfatório em precisão espacial ou convergência de política.

Estudo sobre conversão e quantização de modelos em formato GGUF

O GGUF é um formato binário para armazenar e implantar grandes modelos de linguagem de forma eficiente. Ele foi desenvolvido por **Georgi Gerganov** (criador do llama.cpp) e substituiu os formatos GGML / GGMF / GGJT usados anteriormente. Ao contrário de formatos que armazenam apenas tensores, como o safetensors, o GGUF codifica os pesos do modelo e um conjunto padronizado de metadados, tornando-o um arquivo auto-contido e fácil de carregar. A estrutura binária foi projetada para três objetivos: **eficiência** (reduzir requisitos de memória e computação), **escalabilidade** (suportar modelos grandes, acima de 100 GB) e **extensibilidade** (permitir adicionar novos recursos sem quebrar a compatibilidade). Modelos treinados em frameworks como PyTorch podem ser convertidos para GGUF e usados com mecanismos como llama.cpp, GPT4All e Ollama.

Figura 8.1 – Esquema da estrutura interna de um arquivo de pesos no formato GGUF (`model.gguf`). O diagrama destaca o cabeçalho com versão, *magic number*, contagem de tensores e de metadados, a seção de metadados organizada em pares chave–valor (informando arquitetura, nome do modelo, tamanho de contexto, tokenizer etc.) e a lista de tensores, na qual cada tensor possui nome, dimensões, tipo de dado e *offset* que aponta para a região do arquivo onde seus valores estão armazenados.[18]

O formato surgiu para resolver limitações do GGML. Ele utiliza compatibilidade com memory mapping (mmap), o que reduz o tempo de carregamento e melhora a velocidade de inferência. Suporta tokens especiais (delimitadores que indicam o início / fim de prompts), recurso ausente nos modelos GGML. O GGUF também é genérico, compatível com arquiteturas não-Llama, como Falcon, Bloom, Phi e Mistral. Sua estrutura desacopla dados, metadados e hiperparâmetros, tornando o formato mais extensível e evitando rupturas quando há mudanças no modelo. Além disso, carregar ou salvar um arquivo GGUF não requer bibliotecas externas, facilitando o uso. Como reflexo dessas vantagens, o GGUF é hoje utilizado em modelos como LLaMA 3.1, Phi-3 e DeepSeek Coder v2.

Internamente, o formato GGUF é o pilar de um ecossistema de quantização que combina a biblioteca de tensores GGML, o motor de inferência llama.cpp e algoritmos de quantização

[18] Reproduzido de HUGGING FACE. *GGUF*.

pós-treinamento. A ideia é diminuir a largura de bits de cada peso sem re-treinar o modelo, gerando checkpoints menores que podem ser executados em hardware de consumo.

Diferenças entre GGUF e GGML

O GGML foi pioneiro na quantização de modelos Llama-like, mas seu formato tornou-se obsoleto após o lançamento do GGUF em 2023. Os principais benefícios do GGUF em relação ao GGML podem ser resumidos nos tópicos abaixo:

	GGML	GGUF
Velocidade	carregamento e inferência mais lentos	mmap acelera o carregamento/inferência
Tokens especiais	sem tokens especiais	suporta tokens especiais
Arquiteturas suportadas	apenas Llama-like	suporta modelos não-Llama
Extensibilidade	acoplamento rígido e quebrável	estrutura desacoplada, extensível
Facilidade de uso	instalação e configuração complexas	uso simples, sem dependências

Essas melhorias explicam por que o GGUF se tornou o padrão para quantização e inferência de LLMs em hardware de consumidor.

Gerações de quantização no GGUF

O llama.cpp evoluiu por meio de três gerações de algoritmos de quantização. As gerações posteriores reutilizam conceitos das anteriores, mas introduzem otimizações adicionais.

Legacy Quants (primeira geração)

Os métodos legados utilizam quantização afim, mapeando cada peso em ponto flutuante para um inteiro de menor bitagem. Existem dois tipos:

- **Tipo 0 (simétrico)** – assume que os pesos variam de forma simétrica em torno de zero. Os pesos são recortados para um intervalo fixo (α, β), escalados com um fator S e armazenados como inteiros de 4, 5 ou 8 bits (por exemplo, Q4_0). Essa abordagem desperdiça bins quando a distribuição dos pesos não é simétrica

- **Tipo 1 (assimétrico)** – usa um ponto-zero Z para ajustar o intervalo quando os pesos não são simétricos. A quantização realiza um deslocamento (por Z) e depois a escala S. Essa variante requer armazenar S e o deslocamento α, aumentando um pouco o tamanho do arquivo, mas melhora a precisão.

Para reduzir a sobrecarga, os pesos são agrupados em blocos de 16 ou 32 elementos (block quantization). Cada bloco compartilha as constantes de quantização S (e α no tipo 1). Blocos maiores significam menos constantes e menor precisão; blocos menores aumentam a precisão ao custo de espaço.

K-Quants (segunda geração)

Os K-quants introduzem a ideia de quantizar também as próprias constantes de quantização (double quantization), inspirada pelo QLoRA. Em vez de armazenar S e α em precisão completa, eles são quantizados, criando uma hierarquia de dois níveis que reduz o overhead de armazenamento.

Para economizar espaço e melhorar o acesso à memória, os K-quants agrupam oito blocos regulares (de 32 pesos) em um super-bloco. Cada super-bloco possui:

1. **Blocos regulares** – contêm 32 pesos quantizados
2. **Constantes dos blocos** – escala e deslocamento de cada bloco, agora quantizados como INT8

3. **Constantes do super-bloco** – um par de FP16 que "dequantiza" as constantes dos blocos.

Essa hierarquia reduz o overhead de um modelo de 16 B parâmetros de ~2 GB para ~1 GB. Além do ganho de armazenamento, a leitura sequencial de super-blocos melhora o uso de cache e a largura de banda de memória.

Outra inovação é a **estratégia de precisão mista**: nem todos os pesos são quantizados com a mesma bitagem. Camadas críticas, como normalizações de camada e pesos de atenção, podem usar maior precisão (Q5_0, Q6_K, etc.). O sufixo de tamanho (S, M ou L) indica quanto das camadas intermediárias recebe mais bits – S produz arquivos menores, M balanceia tamanho e qualidade, e L prioriza qualidade.

I-Quants (terceira geração)

Os I-quants representam uma mudança conceitual: substituem a quantização escalar por quantização vetorial. Em vez de quantizar cada peso isoladamente, agrupam vetores de 8 pesos (*w_vec*) e usam um codebook com vetores de referência (*r_vec*). O algoritmo simplificado funciona assim:

1. Encontrar o vetor de referência mais próximo no codebook
2. Calcular uma escala S = |*w_vec*| / |*r_vec*|
3. Armazenar o código do vetor de referência e a escala S, que é compartilhada por 256 vetores.

Para reduzir o tamanho do codebook, utiliza-se o truque do sinal: o codebook contém apenas vetores positivos; o algoritmo procura o vizinho mais próximo do vetor em valor absoluto e armazena, além do código e da escala, um byte que codifica os sinais originais dos pesos. Mesmo com esse byte extra, a compressão é superior à das gerações anteriores, por exemplo, IQ2 atinge cerca de 2 bits por peso.

A letra “I” provavelmente significa importance, pois esses quantizadores suportam a matriz de importância descrita a seguir.

Matriz de importância (imatrix)

A importance matrix é um aprimoramento de qualidade que pode ser usado com qualquer método de quantização. Embora as versões de baixa bitagem (IQ2) necessitem dela para obter boa qualidade, a imatrix é ortogonal aos algoritmos e também pode ser aplicada aos métodos Legacy e K. A ideia central é que nem todos os pesos têm a mesma importância: pesos que impactam mais a saída do modelo devem receber maior precisão.

Para cada matriz de pesos W, a imatrix I tem as mesmas dimensões e atribui um escore real a cada peso. O cálculo é feito com base em um conjunto de calibração – algumas centenas de amostras de texto (geralmente extraídas do corpus Wikitext). Durante a inferência nesse conjunto, obtém-se as ativações de saída y; o quadrado dessas ativações fornece um escore de importância por linha $I_i = y_i^2$. Esse escore é combinado com a magnitude do peso para produzir um valor por peso $I_{ij} = y_i^2 + \sqrt{(\sigma^2 + w_{ij}^2)}$.

Em vez de atribuir mais bits, a imatrix ajusta as constantes de quantização S e Z para reduzir o erro de reconstrução dos pesos importantes. O processo calcula um conjunto separado de constantes (S', Z') para a dequantização e pode até realizar uma pequena busca em torno de S para melhorar a precisão. O custo é apenas no momento da quantização: há um pequeno aumento no tempo de conversão (cerca de 10–20%) e nenhum overhead durante a inferência.

Convenção de nomes dos arquivos GGUF

Os nomes dos modelos quantizados seguem um padrão que indica a geração do algoritmo, a largura de bits e a estratégia de precisão:

Componente	Significado (exemplos)
Versão	• _0/_1 = Legacy • K = K-quants • I = I-quants
Bitagem (bpw)	Q4=4 bits, Q5=5 bits, Q6=6 bits, Q8=8 bits; nos I-quants, a taxa é aproximada: IQ1≈1 bpw, IQ2≈2 bpw etc.
Modificador de tamanho	• S (Small) produz arquivos menores com mais agressividade • M (Medium) equilibra tamanho e qualidade • L (Large) ou XL prioriza qualidade.

Por exemplo, Q4_K_S indica um modelo quantizado com 4 bits por peso, usando o algoritmo K-quants na versão “small”.

Motores para uso

Na hora de executar o modelo, diversos motores de inferência open-source suportam GGUF de forma nativa. Entre eles estão:

- **llama.cpp** – um motor em C/C++ que oferece inferência rápida em CPUs e GPUs. Foi a base para a criação do GGUF
- **GPT4All e Ollama** – plataformas que permitem rodar modelos localmente via linha de comando ou API
- **vLLM** – biblioteca de alto throughput que usa técnicas como Paged Attention; ela converte modelos PyTorch para GGUF e pode alcançar throughput até 24× maior que frameworks tradicionais

Referências

HUGGING FACE. *GGUF*. **Documentação oficial do Hugging Face Hub**. Disponível em: https://huggingface.co/docs/hub/gguf. Acesso em: 11 nov. 2025.

IBM. *GGUF versus GGML*. **IBM Think Topics**. Disponível em: https://www.ibm.com/think/topics/gguf-versus-ggml. Acesso em: 11 nov. 2025.

TURC, Iulia. *GGUF Quantization Docs (Unofficial)*. **GitHub repository**, 2024. Disponível em: https://github.com/iuliaturc/gguf-docs. Acesso em: 11 nov. 2025.

Converter modelos para GGUF

Para realizar inferência com llama.cpp, precisei converter meus modelos para o formato GGUF. No repositório deles, já tem disponibilizado o script para realizar essa conversão.

Clone o repositório e instale os requirements

```
git clone https://github.com/ggml-org/llama.cpp.git
cd llama.cpp
pip install -r requirements.txt
```

Da maneira como o llama.cpp foi construído, é necessário separar o projetor do modelo. Então, primeiramente, é necessário converter apenas o projetor. Que como é pequeno e funciona como o mapa da nossa imagem, normalmente se mantém em F16.

```
# ID do seu modelo no hugging face
MODEL_ID = "arantesvictorl/SmolVLM-500M-Instruct-Agentic-GUI"
OUTPUT_MMPROJ = "SmolVLM-500M-SFT-mmproj-f16.gguf"

python convert_hf_to_gguf.py \
 --remote \
 --mmproj \
 --outtype f16 \
 --outfile {OUTPUT_MMPROJ} \
```

```
 {MODEL_ID}
```

Para converter o modelo

```
# ID do seu modelo no hugging face
MODEL_ID = "arantesvictorl/SmolVLM-500M-Instruct-Agentic-GUI"
TIPO_DE_QUANTIZACAO = "q8_0"
OUTPUT_LLM = f"SmolVLM-500M-SFT-{TIPO_DE_QUANTIZACAO}.gguf"

python convert_hf_to_gguf.py \
 --remote \
 --outtype {TIPO_DE_QUANTIZACAO} \
 --outfile {OUTPUT_LLM} \
 {MODEL_ID}
```

Rodar VLMs no celular

Termux

Fazendo uma pesquisa de como poderia implementar isso, re-descobri o aplicativo termux, um emulador de terminal e ambiente linux para Android. Durante a semana 4, por fazer alguns experimentos, já tinha conhecimento dele.

A instalação é muito simples, apenas baixando um arquivo .apk no github deles e instalando no celular.

SSH (opcional)

Para facilitar os comandos e seguir as configurações, procurei se tinha uma forma de conectar no terminal por SSH, encontrei esse tutorial que me serviu muito bem. Seguindo essa lista de comandos:

Instalar os pacotes:

```
pkg update && pkg upgrade
pkg install openssh
```

Definir a senha

```
passwd
```

Descobrir o id de usuário

```
~ $ whoami
u0_a557
```

Descobrir o IP

```
~ $ ifconfig|grep inet
Warning: cannot open /proc/net/dev (Permission denied). Limited output.
        inet 127.0.0.1  netmask 255.0.0.0
        inet 100.72.131.217  netmask 255.255.255.252
        inet 192.168.0.3  netmask 255.255.255.0  broadcast 192.168.0.255
```

Pelo terminal do computador

```
ssh -p 8022 u0_a557@192.168.0.3
```

Com isso já foi possível fazer todos os comandos pelo meu computador.

Motor de Inferência

Para fazer a inferência de modelos encontrei alguns motores que funcionam para Android.

- llama.cpp
- Ollama
- ExecuTorch

Além de alguns aplicativos:

- ChatterUI
- Google AI Edge Gallery
- Pocket Pal

Por ser o mais leve e por ser algo que já havia utilizado, decidi seguir com o llama.cpp.
Buscando pelo Google encontrei um repositório que ensinava a habilitar o uso da GPU do celular para fazer inferência com esses modelos.
Infelizmente não consegui obter os mesmos resultados seguindo à risca o tutorial.
Portanto, decidi optar por rodar em cpu e seguindo o tutorial para buildar o llama.cpp no meu celular.

Instalação do llama.cpp

No terminal do termux, clone o repositório:

```
git clone https://github.com/ggml-org/llama.cpp
cd llama.cpp
```

Instale o pacote make:

```
apt install git cmake
```

Faça o build:

```
cmake -B build
cmake --build build --config Release
```

Fazer inferência com o modelo

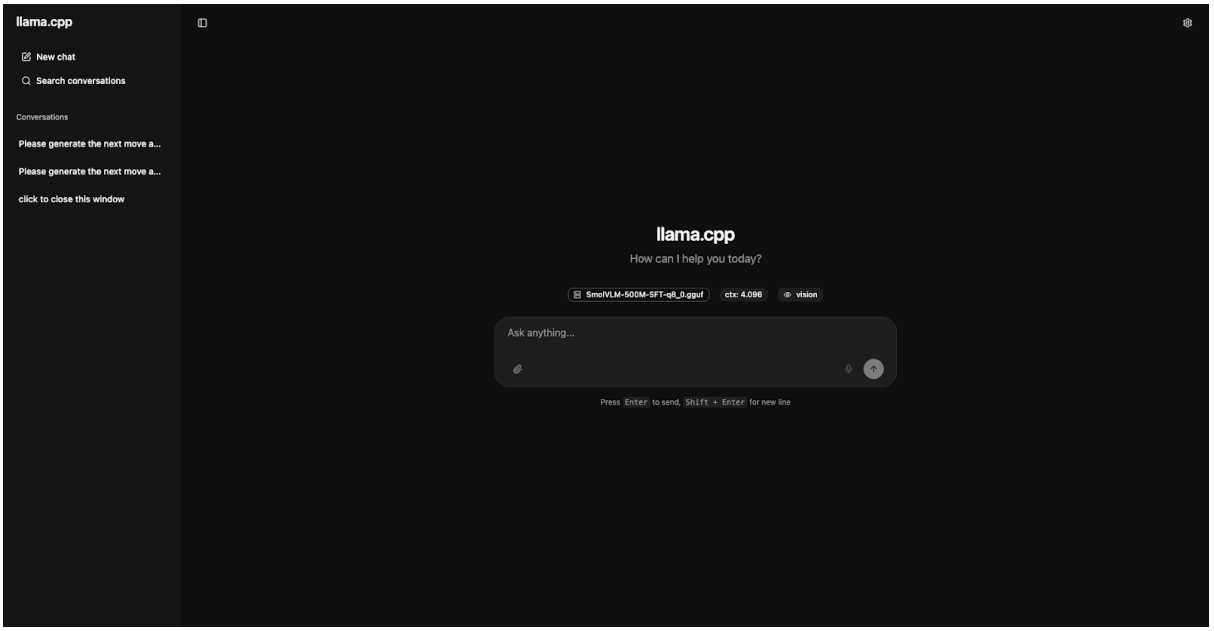
Para realizar a inferência é necessário um modelo no formato GGUF, converti um dos meus modelos com esse tutorial e baixei ele e o projetor dele do hugging face com o comando `wget`.

Web GUI

É possível obter uma interface "GPT-like” por meio do comando:

```
./build/bin/llama-server -m [caminho_para_o_modelo] --mmproj
[caminho_para_o_projetor] --host 0.0.0.0
```

Acessando 0.0.0.0:8080 pelo celular ou <IP_DO_CELULAR>:8080 pelo computador na mesma rede



Multimodal-CLI

```
./build/bin/llama-mtmd-cli -m [caminho_para_o_modelo] --mmproj
[caminho_para_o_projetor] --image [caminho_para_uma_imagem] -p
"[um_prompt_de_texto]" --chat-template [nome_do_template] -n
[numero_de_tokens]
```