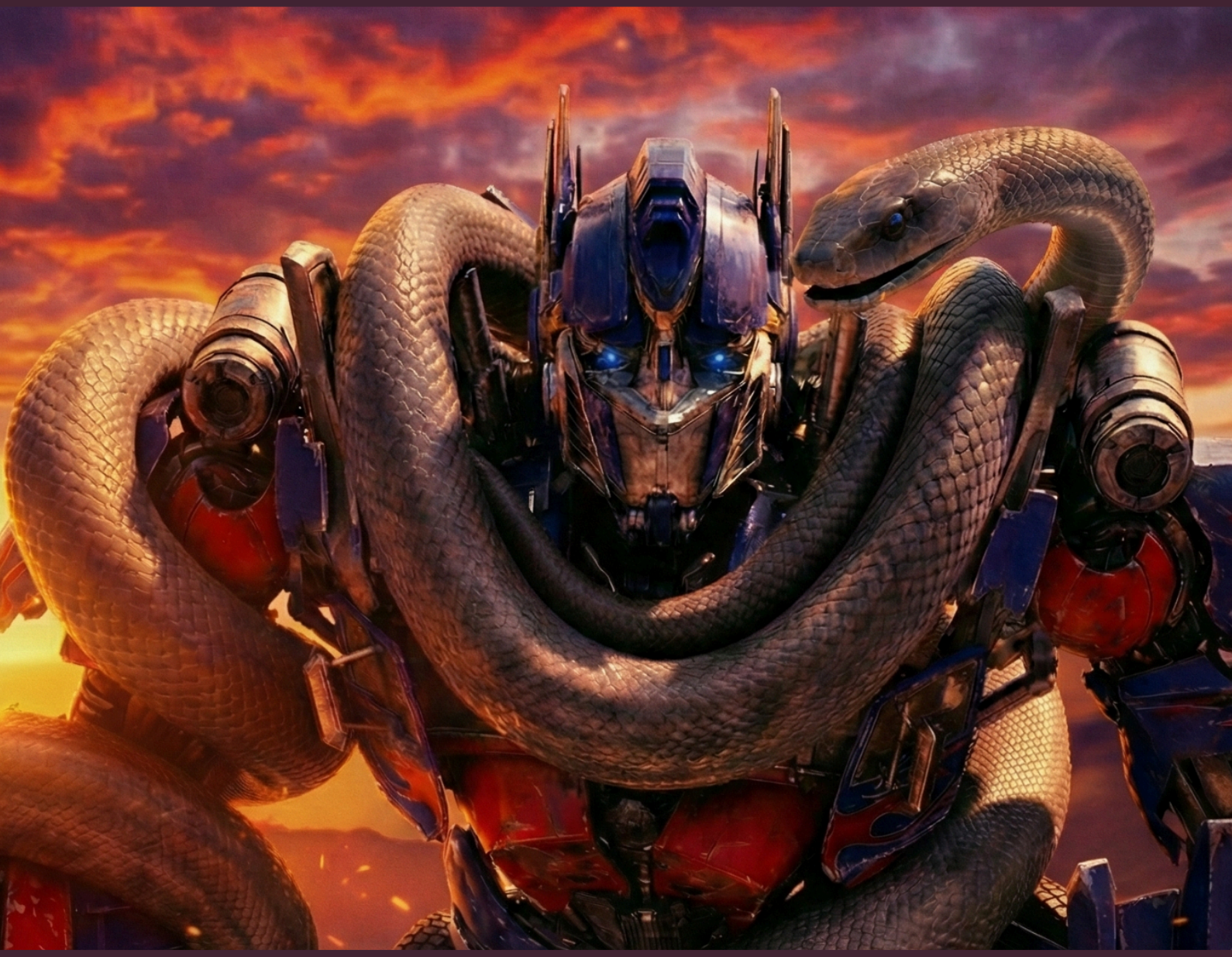
UNIVERSIDADE FEDERAL DE GOIÁS / INSTITUTO DE INFORMÁTICA

Evolução Arquitetural em Modelos de Linguagem

Análise de Otimizações e Avaliação Experimental de Arquiteturas Híbridas

Gustavo Luiz Bueno Pereira



UFG

UNIVERSIDADE
FEDERAL DE GOIÁS

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

GUSTAVO LUIZ BUENO PEREIRA

Evolução Arquitetural em Modelos de Linguagem

Análise de Otimizações e Avaliação Experimental de Arquiteturas Híbridas

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): GUSTAVO LUIZ BUENO PEREIRA

Título do trabalho: Evolução Arquitetural em Modelos de Linguagem

Análise de Otimizações e Avaliação Experimental de Arquiteturas Híbridas

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 **Gustavo Luiz Bueno Pereira**, **Discente**, em 04/02/2026, às 22:47, 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 11:31, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do Decreto nº 10.543, de 13 de novembro de 2020 .

A autenticidade deste documento pode ser conferida no site https://sei.ufg.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0, informando o código verificador **5956495** e o código CRC **FCEA55B8**.

Referência: Processo nº 23070.005498/2026-16 SEI nº 5956495

GUSTAVO LUIZ BUENO PEREIRA

Evolução Arquitetural em Modelos de Linguagem

Análise de Otimizações e Avaliação Experimental de Arquiteturas Híbridas

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.

PEREIRA, GUSTAVO LUIZ BUENO
Evolução Arquitetural em Modelos de Linguagem [manuscrito]: Análise de Otimizações e Avaliação Experimental de Arquiteturas Híbridas / GUSTAVO LUIZ BUENO PEREIRA. - 2025.
103 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 Linguagem. 3. Arquiteturas Híbridas.

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

CDU 004

GUSTAVO LUIZ BUENO PEREIRA

Evolução Arquitetural em Modelos de Linguagem

Análise de Otimizações e Avaliação Experimental de Arquiteturas Híbridas

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)

Daniel Fazzioni
(INF-UFG)

GUSTAVO LUIZ BUENO PEREIRA

Evolução Arquitetural em Modelos de Linguagem

Análise de Otimizações e Avaliação Experimental de Arquiteturas Híbridas

RESUMO

Este Relatório de Conclusão de Curso tem como objetivo reunir os resultados da minha jornada para me tornar um especialista em **Arquiteturas de LLMs**. 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 linguagem; Arquiteturas híbridas.

ABSTRACT

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

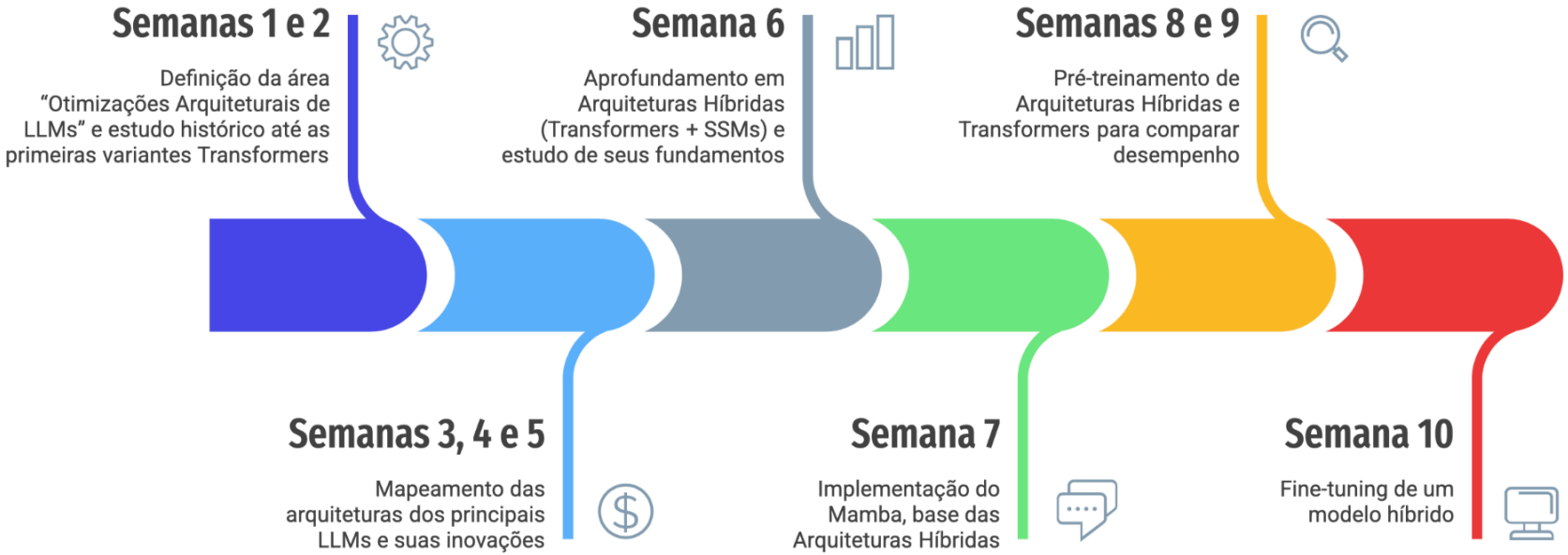
Keywords: Artificial intelligence; Language models; Hybrid architectures.

Goiânia

2025

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DE GOIÁS
INSTITUTO DE INFORMÁTICA

Minha Jornada



Gustavo Luiz Bueno Pereira

Especialista em: Arquiteturas de LLMs

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

MINHA JORNADA

Nome: Gustavo Luiz Bueno Pereira

Especialidade: Arquiteturas de LLMs

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 na **Semana 1**, após perceber que, com minhas atenções voltadas principalmente para a produtização usando Inteligência Artificial e, portanto, mais centradas no uso de APIs de Large Language Models (LLMs) e de frameworks, eu havia reduzido o contato com um aspecto que sempre me instigou na área: compreender por que as arquiteturas neurais são estruturadas como são e como cada solução evolui a partir das limitações da anterior. Essa constatação motivou a definição do tema central em "**Arquiteturas de LLMs**", com o objetivo de estudar a evolução das otimizações arquiteturais presentes nos principais modelos. Para desenvolver uma base sólida para essa compreensão, iniciei pelo estudo dos fundamentos que sustentam os modelos atuais, o que me levou às primeiras leituras sobre a evolução da Modelagem de Linguagem, desde os modelos estatísticos de n-gramas, passando pelas arquiteturas neurais iniciais e mecanismos de recorrência, até os mecanismos de atenção combinados ao paradigma Sequence-to-Sequence que antecederam e motivaram a arquitetura Transformers. Na **Semana 2**, aprofundei esse processo com a leitura do artigo que fundamenta os

[1] Dez Semanas, entre setembro de 2025 e dezembro de 2025.

Transformers; também iniciei uma análise da era pós Transformers, distinguindo as gerações de Pre Trained Language Models e Large Language Models e investigando o papel de modelos como BERT e GPT-3 nessas transições. Ao fim da **Semana**, surgiu a ideia de conduzir o estudo das propostas de otimizações arquiteturais com base na categorização dos fatores que explicam sua adoção ou não em LLMs de fronteira, permitindo uma análise crítica do processo de consolidação dessas inovações. Todos os materiais e referências utilizados nesse período encontram-se organizados no **Apêndice 1**.

Na **Semana 3**, complementei a base construída anteriormente revisitando os artigos que introduziram as variantes *encoder only* e *decoder only* para compreender de forma mais técnica o papel que desempenharam na consolidação dos modelos modernos, identificando fatores que ajudam a explicar a predominância atual do *decoder only* nos LLMs. Em paralelo, estruturei um método para mapear otimizações arquiteturais, adotando um esquema de estudo baseado em seis componentes fundamentais da arquitetura Transformer: (1) Positional Embeddings, (2) Atenção, (3) FeedForward, (4) Normalização, (5) Função de Ativação e (6) Macro Arquitetura. Organizei uma planilha destinada a mapear essas inovações, tomando a arquitetura original Transformers como referência e registrando motivação, origem, local de aplicação, histórico de uso e critérios iniciais de consolidação. Esta planilha pode ser encontrada no **Apêndice 2.** A **Semana 4** foi dedicada à aplicação do meu planejamento em escala mais ampla: pesquisei linhas do tempo de LLMs, identifiquei levantamentos que variavam de pouco mais de 70 a mais de 600 modelos e, a partir deles, selecionei 102 LLMs de destaque para serem centralizados na planilha e iniciei a investigação de suas arquiteturas. Já na **Semana 5**, refinei a priorização dos modelos a serem analisados para concentrar o esforço nos considerados mais relevantes, o que resultou na análise de 30 LLMs, totalizando 34 otimizações mapeadas. Ao fim da **Semana**, em função dos estudos, percebi que gostaria de focar na investigação do modelo Phi-4 mini-flash-reasoning, que me chamou a atenção por se tratar de uma arquitetura híbrida de Transformers e Mamba, sendo um modelo recente (julho de 2025) e lançado por uma Big Tech como parte da sua principal família de modelos. A surpresa veio pelo pensamento de que esse tipo de abordagem híbrida parecia já ter perdido espaço no desenvolvimento de LLMs, o que tornou o achado especialmente inesperado.

O primeiro passo da **Semana 6** foi a leitura do artigo do Phi-4-mini-flash-reasoning, que evidenciou resultados expressivos em benchmarks e maior eficiência de treinamento quando comparado aos demais modelos da mesma família, baseados apenas em Transformer, o que reforçou a justificativa para investigá-lo com mais profundidade. Como o bloco Mamba é essencial para compreender sua estrutura interna, iniciei também o estudo desde as bases dos *State Space Models* até o artigo original do Mamba, buscando entender sua evolução até se tornar viável em modelos desse tipo. Os registros completos dessa etapa, incluindo anotações e documentos de apoio, estão disponíveis nos anexos correspondentes ao **Apêndice 3**.

O início da **Semana 7** foi dedicado a concluir a configuração do ambiente de inferência do Phi-4-mini-flash-reasoning, etapa iniciada na **Semana 6** que exigiu ajustes adicionais de dependências inerentes à arquitetura, cujos detalhes estão documentados no **Apêndice 4**. A continuidade das atividades voltou-se para a implementação do Mamba em PyTorch, com o objetivo de consolidar a compreensão adquirida. Comecei rascunhando a arquitetura a partir do artigo original, destacando dimensionalidades e relações entre módulos; e examinei a organização geral do código oficial, que prioriza otimização e inclui trechos em CUDA. Busquei então tutoriais que apresentassem uma versão essencial do modelo e adotei como base um material que oferecia um equilíbrio adequado entre fidelidade conceitual e acessibilidade. A partir dele, reconstruí o modelo linha a linha, registrando dúvidas e decisões de implementação; e consultando o código oficial sempre que necessário para validar detalhes específicos. A implementação completa desenvolvida nesta etapa também está disponível no **Apêndice 4**.

Nas **Semanas 8 e 9**, dei continuidade ao estudo explorando como as arquiteturas híbridas se comportam em relação às variantes Transformer convencionais durante o pré-treino. A partir do que foi observado no artigo do Phi-4-mini-flash-reasoning, propus uma reprodução, em escala reduzida, dos experimentos conduzidos pelos autores, com a intenção de verificar se a tendência relatada no artigo também surgiria em um cenário computacional mais limitado, ou seja, arquiteturas híbridas convergindo mais rapidamente, isto é, apresentando valores menores de loss e perplexity na mesma quantidade de passos de treinamento. Para isso, utilizei duas arquiteturas disponibilizadas no repositório ArchScale

da Microsoft: uma híbrida e uma baseada exclusivamente em Transformer. As duas arquiteturas foram ajustadas para aproximadamente 150 milhões de parâmetros. Como se tratam de blocos estruturais diferentes, o próprio artigo indica que trataram de aplicar reduções “proporcionais” em componentes específicos de cada arquitetura, garantindo uma comparação justa, segundo os autores. Nesse período, gastei uma boa parte do tempo resolvendo problemas com o ambiente de execução e realizando ajustes necessários no framework, além de organizar um conjunto de treinamento de cerca de 2.2 bilhões de tokens, seguindo as diretrizes das *Chinchilla Scaling Laws* para manter uma proporção mínima entre tamanho do modelo e quantidade de dados. Com o ambiente estabilizado, executei os experimentos em duas configurações distintas de janela de contexto e *batch size*. Os resultados revelaram o mesmo comportamento geral, com a arquitetura híbrida apresentando métricas “melhores” e um treinamento mais estável do que a arquitetura Transformer convencional. As configurações adotadas, as adaptações realizadas e os resultados completos encontram-se detalhados no **Apêndice 5**.

Na **Semana 10**, avancei para a etapa de *Supervised Fine-Tuning*. Meu objetivo era comparar o ajuste supervisionado de uma arquitetura híbrida com o de um modelo Transformer convencional utilizando, em especial, dados em português. Para isso, selecionei o Falcon-H1-0.5B-Base como representante híbrido - por disponibilizar uma versão base pequena e somente pré-treinada, o que tornava o experimento mais viável - e escolhi o Qwen2.5-0.5B como Transformer. Para compor o conjunto de treinamento, utilizei um corpus em português de diálogos multi-turno no formato *human–assistant*, adequado para *instruct-tuning*. Ao longo do processo, boa parte do aprendizado esteve na análise dos hiperparâmetros do SFTTrainer da biblioteca trl (Transformer Reinforcement Learning) e em como cada ajuste influenciava o comportamento do treinamento. Todas as configurações adotadas, scripts utilizados e anotações referentes à preparação do ambiente e à execução do SFT foram reunidas no **Apêndice 6**. As limitações de dependências para modelos híbridos voltaram a aparecer, e, mesmo utilizando um framework consolidado, não foi possível habilitar as implementações otimizadas do bloco Mamba. Por isso, o treinamento do Falcon-H1-0.5B-Base utilizou versões padrão, consideravelmente mais lentas. A intenção inicial era utilizar cerca de 100 mil amostras, mas utilizamos 10 mil amostras, ainda assim exigindo mais de quatro horas de processamento. Em comparação, o Qwen2.5-0.5B

completou o treinamento em cerca de trinta minutos. Apesar dessa diferença, o Falcon-H1-0.5B-Base apresentou um ajuste mais estável e, para a mesma quantidade de *steps*, alcançou valores menores de *loss* em treino e validação, além de maior acurácia média de predição por amostra. A ausência de outros *benchmarks* não permite uma avaliação mais ampla, mas os resultados sugerem que a arquitetura híbrida demonstrou melhor capacidade de adaptação mesmo sob restrições de execução.

Finalizo esta Jornada com um sentimento genuíno de orgulho pelo quanto aprendi ao longo das dez **Semanas**, sobretudo por ter retomado o entusiasmo em estudar arquiteturas a fundo e compreender melhor a evolução dos LLMs que utilizamos diariamente. Também marcou essa trajetória o fato de, pela primeira vez, ter realizado tanto o pré-treino quanto o supervised fine-tuning de um modelo de linguagem, além de reproduzir, mesmo em hardware limitado, experimentos de pré-treino com resultados coerentes ao artigo de referência. Essa experiência reforçou meu interesse pelas Arquiteturas Híbridas, que vêm ganhando espaço e mostram sinais concretos de eficiência, inclusive nos testes que eu mesmo conduzi, indicando um caminho promissor ainda em expansão. Sou grato por ter tido a oportunidade de viver por si só, o Processo da Residência em IA, onde vivi uma metodologia de aprendizado que irei carregar para a vida, e será generalizada para qualquer outra área de conhecimento que decidir estudar. Por fim, agradeço aos meus amigos, tanto pelas trocas de conhecimento constantes, quanto pelos momentos essenciais de descontração; aos professores da Residência que me ajudaram a atravessar momentos decisórios dessa trajetória e a todos os outros professores que me capacitaram para chegar até aqui, bem como à minha família, cuja presença e suporte tornaram possível toda essa trajetória.

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]:

Gustavo Luiz Bueno Pereira

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

Durante a primeira Semana da Residência, foram realizados as seguintes atividades:

- Definição do tema central da Residência: **Otimizações dos LLMs a nível de Arquitetura**;
- Reflexão e descrição da motivação em relação ao tema;
- Seleção de tópicos relacionados ao tema com base nas conferências sugeridas:
  - Natural Language Processing (NLP) - (ICAI 2025 e ACC'25)
  - Neural Networks - (ICAI 2025)
  - Artificial Neural Networks / Deep Learning / Large Language Models - (ICDATA'25)
- Revisão histórica e conceitual de Language Modeling até chegar no paper que inicia o foco do tema, "Attention Is All You Need", pois é nele que é introduzido a primeira versão da arquitetura utilizada nos LLMs atuais. Considerando que Language Modeling é área de atuação fundamental dos LLMs, o seu estudo mostrou-se essencial para o entendimento das motivações que levaram até o estágio atual, incluindo estudos:
  - Da Definição;
  - Das primeiras abordagens com modelos estatísticos de n-gramas;
  - Da primeira abordagem neural de language modeling: Neural Probabilistic Language Model
  - Das primeiras abordagens envolvendo Recurrent Neural Network Language Models, e a sua evolução para as Long Short-Term Memory Network Language Models
  - Da modelagem de problemas Seq2Seq com arquiteturas RNN Encoder-Decoder e a evolução das mesmas para o uso de mecanismos de atenção
- Para cada evolução cronológica da área, buscou-se entender essencialmente o problema alvo da solução proposta, como esse problema é solucionado ou mitigado, e quais problemas ainda emergem dessa solução, permitindo o entendimento da motivação de cada uma delas.
- Ferramentas de apoio à pesquisa:
  - "DeepResearch" do ChatGPT e do Gemini para iniciar a revisão literária, visto que a ferramenta cria um relatório com base em diversos artigos, os quais também serviram de apoio por si só.
  - "Google Scholar" para levantamento de artigos "surveys" de LLMs.
  - "ResearchRabbit" para agregação de todos os artigos que chamaram atenção até aqui.
- A pesquisa pode ser encontrada na íntegra no link a seguir: Pesquisa Semana 1.

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

- Revisar de forma minuciosa a arquitetura Transformers, e o paper “Attention is All You Need” como um todo. Perguntas devem ser formuladas para entendimento mais profundo dos problemas que vêm antes, e daqueles que vêm depois dessa arquitetura, como por exemplo, “quais foram as primeiras expoentes dessa arquitetura”, e assim por diante.
- Busca horizontal das inovações pós Transformers, buscando mapeá-las cronologicamente para **iniciar** um roadmap de entendimento das evoluções chaves e tentativa de evoluções, com foco na arquitetura.

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

ACEITE DA ENTREGA:

CEDRIC LUIZ DE CARVALHO: Go!

Documento de Apoio - Semana 1

*Referente ao documento “Pesquisa Semana 1 ”

Motivação e Definição do Tema

Ao investigar os temas das conferências do Congresso CSCE 2025, refletir sobre minha área de atuação atual e revisitar conteúdos do Bacharelado em Inteligência Artificial da UFG, retomei o interesse em compreender por que determinadas arquiteturas neurais são estruturadas como são, por que funcionam e como são adaptadas para diferentes problemas. Esse interesse sempre esteve ligado ao estudo de arquiteturas de Redes Neurais.

Um marco nesse processo foi perceber que existe uma base conceitual comum, a partir da qual novas arquiteturas surgem como modificações para contornar limitações anteriores. Um exemplo clássico é o das Redes Neurais Convolucionais em Visão Computacional. Embora pareçam muito diferentes de uma MLP, podem ser vistas como uma regularização da MLP, restringindo conexões a janelas locais das features de entrada em vez de usar todas as entradas. A motivação está no fato de que, para imagens, padrões locais podem ser mais relevantes do que padrões globais. A MLP tenta capturar padrões globais, enquanto a convolução limita cada neurônio a uma janela de pixels.

Esse exemplo ilustra o fascínio pelo estudo das arquiteturas e de sua evolução para resolver problemas específicos. No entanto, minha atuação profissional recente ficou centrada no uso de APIs de Large Language Models, modelos fechados baseados em Transformers decoder only, treinados com grandes volumes de dados e alto poder computacional, acessível principalmente a grandes empresas. Esses modelos se tornaram a principal base para criação de produtos, o que direcionou meu foco para o uso de frameworks e APIs, e não para o entendimento profundo das arquiteturas e de sua evolução. O exercício de estudar fundamentos e detalhes internos dos modelos acabou diminuindo bastante.

Diante disso, decidi que a Residência em IA seria uma oportunidade para retomar esse interesse, direcionando o tema ao estudo das otimizações arquiteturais em Large Language Models. A ideia é revisar papers que sustentam a área desde a ascensão dos Transformers até os modelos abertos mais recentes, como Qwen3, DeepSeek 3.1, Gemma 3 e GPT-oss. O gatilho para essa decisão foi justamente o lançamento do GPT-oss, que utiliza Mixture of Experts dentro da arquitetura principal. Eu já havia estudado MoE de forma superficial, mas ao reler esse trecho do relatório técnico percebi que não dominava a arquitetura como gostaria, o que reacendeu a vontade de estudá-la e, por consequência, entender melhor a evolução arquitetural dos LLMs.

Em relação às conferências ICDATA'25, ACC'25 e ICAI 2025, do Congresso CSCE 2025, identifiquei tópicos alinhados a essa direção temática: *Natural Language Processing* (NLP), *Neural Networks*, *Artificial Neural Networks* / *Deep Learning* / *Large Language Models* e Modelagem de Linguagem (*Language Modeling*).

Modelagem de Linguagem (Language Modeling)

Antes de investigar as otimizações arquiteturais dos LLMs ao longo do tempo, torna-se necessário revisitar conceitos fundamentais: a própria tarefa de modelagem de linguagem, a evolução dos modelos até os Transformers e as principais limitações que cada etapa procurou resolver.

Para obter uma visão geral inicial, utilizei a funcionalidade de pesquisa aprofundada tanto no Gemini quanto no ChatGPT, com o mesmo prompt, buscando uma visão histórica e técnica da modelagem de linguagem até o paper *Attention is All You Need*, destacando modelos, arquiteturas anteriores e seus problemas, bem como a forma como as evoluções tentaram corrigi-los. Em paralelo, realizei buscas no Google Scholar por surveys sobre Large Language Models e Natural Language Processing. Os artigos que se mostraram mais promissores foram adicionados a uma coleção no ResearchRabbit para consulta ao longo da Residência, em especial *Large Language Models: A Survey* e *A Survey of Large Language Models*.

O que é Language Modeling?

Modelagem de linguagem é a tarefa de prever a próxima palavra ou token em uma sequência, dada a sequência anterior. Em termos probabilísticos, um modelo de linguagem define uma distribuição sobre sequências e atribui probabilidades aos possíveis próximos termos. Essa tarefa é a base tanto para geração de sentenças coerentes quanto para tarefas derivadas, como instruction following, em que o modelo aprende a seguir comandos em linguagem natural sem depender de muitos exemplos explícitos, característica central de modelos usados em chatbots como ChatGPT e Gemini.

Primeiras abordagens: modelos estatísticos de n-gramas

As primeiras abordagens para modelagem de linguagem foram estatísticas, baseadas na hipótese de Markov. Desde 1948, com as ideias de Claude Shannon, modelos de n-gramas passaram a ser usados para estimar probabilidades de continuação a partir de contagens de frequência de sequências de n palavras. Em um modelo de trigramas, por exemplo, a próxima palavra depende apenas das duas anteriores.

Esses modelos apresentam limitações importantes. Não capturam significado ou similaridade semântica: “rei” e “rainha” são apenas símbolos distintos. Combinações raras ou nunca vistas recebem probabilidades inadequadas, e a janela de contexto fixa cria um dilema. Janelas pequenas reduzem esparsidade, mas não capturam dependências de longo alcance. Janelas grandes capturam mais contexto, mas sofrem com explosão combinatória de estados e custo exponencial.

Modelagem com redes neurais

O *Neural Probabilistic Language Model* (NNLM) de Bengio et al. (2003) foi o primeiro a usar redes neurais para modelagem de linguagem em nível de palavras, atacando diretamente o problema de esparsidade dos n-gramas. Cada palavra passa a ser representada por um vetor denso aprendido junto com a tarefa, permitindo que palavras com uso semelhante fiquem próximas no espaço vetorial. O modelo recebe as últimas $n-1$

palavras, projeta-as nesses vetores, concatena e passa tudo por uma MLP, cuja saída é uma distribuição de probabilidade (softmax) sobre o vocabulário.

Essa arquitetura consegue generalizar para sequências nunca vistas, pois embeddings próximos permitem atribuir probabilidades razoáveis a contextos novos. Mesmo assim, permanece a limitação de uma janela curta de contexto, já que aumentar muito essa janela implica custo elevado. Ainda assim, o trabalho é fundamental por introduzir representações que antecipam a noção de embeddings e abrir caminho para modelos neurais modernos.

As *Recurrent Neural Network Language Models* (RNNLMs), introduzidas por Mikolov em 2010, avançam em relação ao NNLM ao abandonar a janela fixa. Em vez disso, cada palavra atualiza um estado oculto que resume o histórico. A partir desse estado, o modelo gera a probabilidade da próxima palavra. O compartilhamento de pesos ao longo do tempo reduz o número de parâmetros e permite, em teoria, capturar dependências de comprimento variável.

Na prática, porém, as RNNs sofrem com desaparecimento e explosão de gradientes devido ao algoritmo de backpropagation através do tempo (BPTT), que envolve sucessivas multiplicações pela matriz de pesos recorrentes. Gradientes podem diminuir ou crescer exponencialmente, prejudicando o aprendizado de dependências de longo prazo e dificultando a convergência. Além disso, o processamento sequencial impede paralelismo eficiente, tornando o treinamento mais lento.

A *Long Short-Term Memory* (LSTM), proposta em 1997 e aplicada com força em modelagem de linguagem a partir de 2013–2014, foi a principal resposta a esses problemas. A LSTM introduz portas de esquecimento, entrada e saída para controlar o fluxo de informações no estado de célula, criando uma memória interna mais estável ao longo do tempo e mitigando vanishing e exploding gradients por meio de um caminho mais aditivo para os gradientes. Na prática, isso permite lembrar informações por dezenas de passos e esquecer seletivamente quando necessário, o que torna a arquitetura muito mais adequada a dependências de longo alcance. Ainda assim, limitações permanecem, como o treinamento lento e dificuldades com dependências muito distantes. As GRUs surgem como

uma versão simplificada das LSTMs, seguindo a mesma lógica de controle de fluxo de informação, mas sem o mesmo impacto que as LSTMs tiveram.

Modelagem Seq2Seq com arquitetura encoder-decoder

Aplicações como tradução automática, sumarização e resposta a perguntas exigem mapear sequências de entrada em sequências de saída, e não apenas continuar um texto. Arquiteturas puramente LSTMLM não eram adequadas para isso, pois não distinguiam onde termina a entrada e começa a saída. Para resolver esse problema, surge a arquitetura encoder-decoder, ou sequence to sequence (Seq2Seq), inicialmente aplicada à tradução.

O encoder lê a sequência de entrada e a comprime em um vetor de contexto de tamanho fixo, resultante do estado oculto final. O decoder, inicializado com esse vetor, gera a sequência de saída palavra a palavra. Essa abordagem permitiu atacar tarefas de mapeamento de sequência para sequência, mas introduziu um gargalo: quanto maior a sequência de entrada, mais informação precisa ser comprimida em um único vetor, o que prejudica a qualidade em sentenças longas.

Bahdanau et al. (2014) propuseram então o mecanismo de atenção no contexto seq2seq. Em vez de usar um único vetor fixo, o decoder passa a consultar todos os estados ocultos do encoder a cada passo. Uma pequena rede calcula pontuações de relevância entre o estado atual do decoder e cada estado do encoder, que são normalizadas por softmax para gerar pesos de atenção. O vetor de contexto passa a ser uma soma ponderada dos estados do encoder, recalculada a cada passo. Assim, o modelo tem acesso dinâmico à sequência de entrada e foca nas partes mais relevantes em cada momento, o que reduz o gargalo de informação e melhora o tratamento de dependências de longo alcance.

Diversas outras arquiteturas surgiram nesse período, mas as descritas acima são as mais importantes para compreender o caminho até os Transformers.

Referências

MINAEE, Shervin et al. Large Language Models: A Survey. arXiv, , 23 mar. 2025. Disponível em: <http://arxiv.org/abs/2402.06196>. Acesso em: 3 set. 2025

ZHAO, Wayne Xin et al. A Survey of Large Language Models. arXiv, , 11 mar. 2025. Disponível em: <http://arxiv.org/abs/2303.18223>. Acesso em: 2 dez. 2025

SINHA, Akanksha. From N-Grams to Transformers: Tracing the Evolution of Language Models. Medium, 17 maio 2025. Disponível em: <https://medium.com/@akankshasinha247/from-n-grams-to-transformers-tracing-the-evolution-of-language-models-101f10e86eba>. Acesso em: 2 dez. 2025

BOLL, Heloisa Oss. The neural probabilistic language model. Medium, 11 out. 2024. Disponível em: <https://hossboll.medium.com/the-neural-probabilistic-language-model-ddbffa0bae34>. Acesso em: 2 dez. 2025

Chapter 8 Attention and Self-Attention for NLP | Modern Approaches in Natural Language Processing. Seminar on Natural Language Processing, LMU Munich, 2020. Disponível em: https://slds-lmu.github.io/seminar_nlp_ss20/attention-and-self-attention-for-nlp.html. Acesso em: 2 dez. 2025.

MIKOLOV, Tomáš et al. Recurrent neural network based language model. In: INTERSPEECH 2010. Interspeech 2010. ISCA, 26 set. 2010. Disponível em: <https://www.isca-archive.org/interspeech_2010/mikolov10_interspeech.html>. Acesso em: 2 dez. 2025

SUTSKEVER, Ilya; VINYALS, Oriol; LE, Quoc V. Sequence to Sequence Learning with Neural Networks. In: Curran Associates, Inc., 2014. Disponível em: <https://proceedings.neurips.cc/paper_files/paper/2014/hash/5a18e133cbf9f257297f410bb7eca942-Abstract.html>. Acesso em: 2 dez. 2025

BAHDANAU, Dzmitry; CHO, Kyunghyun; BENGIO, Yoshua. Neural Machine Translation by Jointly Learning to Align and Translate. arXiv, , 19 maio 2016. Disponível em: <http://arxiv.org/abs/1409.0473>. Acesso em: 2 dez. 2025

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]:

Gustavo Luiz Bueno Pereira

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

Durante a segunda Semana da Residência:

- Reescrita do tema central para "**Large Language Models e suas Otimizações Arquiteturais"**, apenas para deixar mais claro que não tomarei o rumo de propor uma nova arquitetura ou otimização, além de deixar novas possíveis direções em aberto.
- Foi realizada a leitura em detalhes do paper "Attention Is All You Need", com o intuito de consolidar os conhecimentos acerca da arquitetura "Transformers". Um detalhe é que, por mais que eu já conhecesse relativamente bem a arquitetura, nunca havia lido o paper original, então ainda sim foi uma atividade enriquecedora.
- Busca horizontal inicial dos modelos lançados após esse paper, com apoio dos dois surveys coletados na Semana anterior, onde:
    - Descobri que são considerados duas grandes gerações de modelo pós Transformers, sendo elas a Pre-Trained Language Models (PLM) e Large Language Models (LLM);
    - Busquei entender, então, como essas duas gerações são diferenciadas, quais as características gerais dos modelos presentes nela, que tipo de tarefas são voltadas a solucionar, e o que marca o início de cada uma;
    - Ao constatar que os modelos que principalmente marcam o início de cada geração são o BERT e o GPT-3, respectivamente, procurei diferenciá-las baseando-se principalmente nas características deles, como se fossem "representantes";
    - Busquei entender, também, a "Árvore Evolucionária dos Language Models", no sentido de: após o Transformers, visualizar quais modelos foram lançados ao longo do tempo, mas considerando as ramificações dessa arquitetura-mãe: Encoder-Only, Decoder-Only e Encoder-Decoder. Foi uma análise interessante para compreender o porquê cada uma dessas duas gerações tem predominância de diferentes ramificações.
- Ao fim, mapeei os principais LLMs lançados ao longo do tempo, mas não são necessariamente eles que irei selecionar para estudar nas próximas Semanas. Conforme o planejamento dessa Semana, os planos da criação do "roadmap", o qual havia citado na última Semana, mudaram.
- Refletindo sobre o tema proposto, constatei que as mudanças arquiteturais em modelos de fronteira têm ocorrido de forma mais lenta do que os avanços em dados e treinamento. Um exemplo é o "Diff Transformer", proposto pela Microsoft Research: apesar de apresentar ganhos em prova de conceito, não se consolidou nos LLMs de fronteira, nem mesmo na própria família Phi, da Microsoft. Esse caso ilustra a dificuldade de identificar claramente os motivos de "adoção" ou "não adoção" de novas propostas arquiteturais ou de otimizações. Parte dos avanços em

inteligência artificial ocorre de forma essencialmente empírica - hipóteses que se confirmam na prática sem explicação teórica imediata - enquanto outras, mesmo bem fundamentadas, não resultam em ganhos concretos. O Batch Normalization é um exemplo desse fenômeno: sua hipótese inicial foi posteriormente refutada, mas a técnica permaneceu como um marco no treinamento de redes profundas. Diante disso, considerei que um caminho relevante e desafiador seria direcionar o trabalho para um estudo crítico dessas otimizações, mas propondo uma categorização dos fatores que explicam sua adoção ou não em LLMs de fronteira, com o objetivo de permitir uma compreensão pessoal mais estruturada acerca desse processo de consolidação de inovações. Evidentemente, há uma quantidade muito grande de propostas de arquiteturas e otimizações - e é desse fato que nasce o planejamento da próxima Semana.

- A pesquisa pode ser encontrada no link: Semana2_pesquisa

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

- Definir o método de busca e critérios de seleção de artigos de propostas de otimização das arquiteturas dos “Large Language Models”, visto que existem muitos tipos de otimizações possíveis, e muitas publicações nessa área. Portanto, para não correr o risco de tentar “abraçar” o mundo em um tempo tão limitado, é necessário ter critérios bem definidos para escolher bem as principais propostas que serão estudadas.
- Explorar possíveis categorias de classificação das otimizações, já no âmbito proposto de motivos de “adoção” ou “não adoção”. Neste primeiro momento, a proposta é apenas explorar diferentes possibilidades de categorização nesse aspecto, e verificar se realmente faz sentido por si só. A definição final das categorias ocorreria apenas após o mapeamento completo das otimizações, com base em uma visão mais abrangente que possibilite propor uma estrutura coerente. Além disso, estamos considerando ”adoção” como presença da otimização em LLMs abertos notáveis das Big Techs.

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

Agradecimento ao Fazzioni por me relembrar a arquitetura que estavam estudando na época, “Diff Transformer”, visto que contribuiu diretamente para a reflexão sobre o tema e do planejamento da próxima Semana.

ACEITE DA ENTREGA:

CEDRIC LUIZ DE CARVALHO: Go!

Documento de Apoio - Semana 2

*Referente ao documento “ Semana2_pesquisa ”

Tema

Na Semana 1, o tema foi proposto como “Otimizações dos LLMs a nível de Arquitetura”. Agora, reescrito como “‘Large Language Models’ e suas Otimizações Arquiteturais”, busca-se deixar claro que o objetivo não é propor uma nova arquitetura ou otimização, mas estudar e analisar as já existentes, mantendo em aberto possíveis direções futuras.

Motivação da Leitura de “Attention is All You Need”

Na pesquisa da Semana 1 foram investigadas as principais arquiteturas propostas para a tarefa de Language Modeling que antecederam os Transformers, arquitetura central dos LLMs atuais. Considerando o tema da Residência, este artigo é tratado como ponto de partida, pois a partir dele surgem as otimizações que levaram aos modelos mais recentes.

Apesar de já possuir conhecimento consolidado sobre Transformers pela formação, eu nunca havia lido o paper original com a devida atenção, recorrendo a materiais didáticos secundários. Por isso, a leitura integral e detalhada do artigo é fundamental para dar continuidade à Residência com base direta na fonte original.

Leitura de “Attention is All You Need”

Artigo lido: *Attenton_is_All_You_Need_Annotated.pdf*

As anotações foram feitas por meio de marca-texto e comentários no próprio PDF, seguindo o padrão: em vermelho, aspectos, problemas ou soluções em comparação a arquiteturas anteriores; em amarelo, conceitos e definições fundamentais; em verde, anotações sobre o funcionamento da arquitetura.

Pesquisa Horizontal da Era Pós-Transformers

Retomando os surveys de Large Language Models definidos na primeira semana [1] [2], observa-se que a era pós “Attention is All You Need”, no contexto de Language Modeling, é dividida em duas grandes gerações: *Pre-Trained Language Models (PLM)* e *Large Language Models (LLM)*. Até aqui, já se havia passado pelas gerações Statistical LM (modelos de n-gramas) e Neural LM (MLPs, arquiteturas recorrentes e convolucionais).

A geração PLM é marcada principalmente pelo BERT, que utiliza apenas a parte Encoder da arquitetura Transformers (Encoder-only) e propõe um pré-treinamento em larga escala com dados não rotulados (texto puro). O objetivo é produzir embeddings fortemente contextualizados, semântica e sintaticamente, e generalistas. A partir desse modelo pré-treinado, realiza-se fine-tuning para tarefas específicas, congelando ou não parte dos parâmetros. No caso de classificação de sentimentos, por exemplo, adiciona-se uma camada linear sobre o vetor final do token especial [CLS], seguido de softmax, transformando o modelo em um classificador em múltiplas classes. O BERT superou o estado da arte em diversas tarefas, inclusive modelos projetados especificamente para elas.

Essa geração consolidou o paradigma de pré-treinar modelos grandes em dados massivos e objetivos genéricos, para depois adaptá-los em downstream tasks por meio de fine-tuning. Outras arquiteturas PLM importantes são variações e otimizações do BERT, como RoBERTa, ALBERT e DistilBERT. Até aproximadamente 2020, Encoder-only dominava o cenário: pela quantidade de modelos desse tipo (como mostra a Figura 1) e pelo fato de os modelos Decoder-only (basicamente GPTs iniciais) ainda apresentarem desempenho inferior em downstream tasks, enquanto as arquiteturas Encoder-Decoder tinham impacto mais limitado.

A arquitetura e o treinamento propostos pelo GPT-1, embora inicialmente pouco expressivos, evoluíram até o GPT-3, que marca o início da era dos Decoder-only como LLMs e o principal divisor de águas da geração Large Language Models. Essa geração se caracteriza por ampliar de forma significativa o tamanho dos modelos e dos dados. BERT-Large possui cerca de 340 milhões de parâmetros, enquanto GPT-3 tem cerca de 175

bilhões. O dataset do BERT tem cerca de 3,3 milhões de tokens; o do GPT-3, cerca de 500 bilhões de tokens, correspondendo a aproximadamente 570 GB de texto.

Esses fatores resultaram em características marcantes:

- Modelos de propósito geral com habilidades emergentes, isto é, capacidades de resolver tarefas para as quais não foram explicitamente treinados, recebendo apenas instruções e exemplos no próprio prompt (In-Context Learning, que não é um treinamento adicional).
- Mudança na forma de desenvolvimento e uso de IA, com forte centralidade em prompts em linguagem natural, descrevendo tarefas, objetivos e exemplos.
- Consolidação da hipótese de que a arquitetura Decoder-only, autoregressiva, seria suficiente para resolver tarefas complexas de geração de texto, levando à predominância desse tipo de arquitetura na maioria dos LLMs posteriores.

Após o lançamento do GPT-3 em 2020, houve uma forte aceleração no lançamento de LLMs (como ilustrado na Figura 1) e o início de uma era em que muitos modelos, especialmente os de fronteira, são closed-source, dado o caráter competitivo do mercado de serviços baseados em LLMs. Entre os modelos impactantes desse período, de forma não exaustiva, destacam-se Gopher (280B, DeepMind), Megatron-Turing NLG (530B, Nvidia/Microsoft), LaMDA (137B, Google) e Chinchilla (70B, DeepMind). Passa-se também a falar em famílias de modelos, como PaLM (Google), LLaMA (Meta), Claude (Anthropic), Gemma e Gemini (Google), Mistral, DeepSeek, Qwen, entre outras.

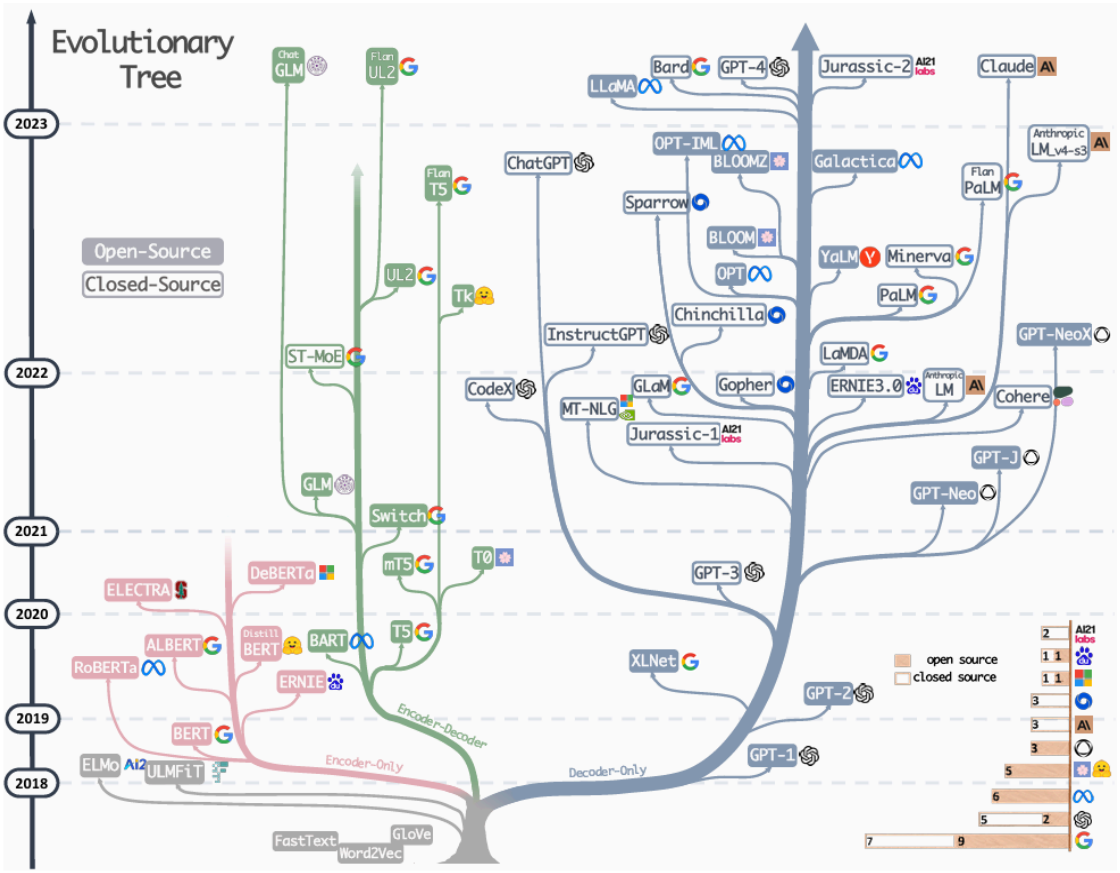


Figura 1: Árvore evolucionária dos Language Models [5]. Representa os modelos lançados mais impactantes até 2023 e a que categoria pertencem quanto ao tipo de arquitetura. Em marrom claro, no início, estão as arquiteturas não baseadas em Transformers. As demais são baseadas em Transformers, em rosa as variações Encoder-only, em verde as Encoder-Decoder e, em cinza, as Decoder-only.

Visão e Planejamento

Refletindo sobre o tema e antecipando alguns passos, analisei o paper do GPT-oss e observei que ele utiliza essencialmente a mesma arquitetura e otimizações já presentes há mais de um ano, com a incorporação de alguns mecanismos que também não são recentes, como Mixture of Experts, conforme verificado em outros modelos abertos de fronteira lidos ao longo do curso. Isso sugere que as otimizações arquiteturais em modelos de fronteira parecem evoluir mais lentamente do que aspectos como estratégias de uso de dados e métodos de treinamento.

Nessa linha, recordo o estudo de uma otimização arquitetural proposta pela Microsoft Research, o Diff Transformer, que foi apresentado no grupo de estudos. Apesar de mostrar

melhor desempenho, em pequena escala, em relação à arquitetura vigente, essa proposta não foi incorporada a modelos de fronteira, nem mesmo na família Phi da própria Microsoft. Em muitos casos, não é trivial identificar claramente os motivos de adoção ou não adoção de uma otimização, e nem sempre há uma explicação única ou definitiva. Parte dos avanços em inteligência artificial ocorre de forma essencialmente empírica: formula-se uma hipótese, realiza-se a experimentação e, mesmo sem uma justificativa teórica consolidada, obtêm-se resultados práticos relevantes. O inverso também acontece, quando propostas teoricamente elegantes não se traduzem em ganhos concretos. Um exemplo frequentemente discutido é o Batch Normalization, inicialmente motivado pela ideia de reduzir o *internal covariate shift*, explicação que mais tarde foi questionada, sem que isso diminuísse o impacto prático da técnica.

Diante disso, considerei que um caminho interessante e desafiador para o trabalho é realizar um estudo crítico das otimizações arquiteturais propostas na literatura, analisando e categorizando os motivos de adoção e não adoção. O objetivo é entender por que algumas propostas se consolidaram em modelos de linguagem de grande escala, enquanto outras permaneceram restritas a ambientes experimentais ou foram abandonadas.

Referências

[1] ZHAO, Wayne Xin et al. A Survey of Large Language Models. arXiv, 11 mar. 2025. Disponível em: http://arxiv.org/abs/2303.18223. Acesso em: 11 set. 2025.

[2] MINAEE, Shervin et al. Large Language Models: A Survey. arXiv, 23 mar. 2025. Disponível em: http://arxiv.org/abs/2402.06196. Acesso em: 11 set. 2025.

[3] How positional encoding works in transformers? 16 dez. 2023. Disponível em: https://www.youtube.com/watch?v=T3OT8kqoqjc. Acesso em: 11 set. 2025.

[4] VASWANI, Ashish et al. Attention Is All You Need. arXiv, 2 ago. 2023. Disponível em: http://arxiv.org/abs/1706.03762. Acesso em: 11 set. 2025.

[5] YANG, Jingfeng et al. Harnessing the Power of LLMs in Practice: A Survey on ChatGPT and Beyond. arXiv, 27 abr. 2023. Disponível em: http://arxiv.org/abs/2304.13712. Acesso em: 11 set. 2025.

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: 18 de set. de 2025

Participantes da Entrega [matriculados em Residência em IA]:

GUSTAVO LUIZ BUENO PEREIRA

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

Tema central: " 'Large Language Models' e suas Otimizações Arquiteturais"

Durante a terceira Semana da Residência:

- Identificou-se uma lacuna comprometedora nos estudos anteriores: mesmo após estudar toda a evolução do Language Modeling até chegar no Transformers, e estudar suas variações e entender que os LLMs seguiram com a variação Transformers Decoder, percebi que não houve um estudo em relação à compreensão da motivação que levou ao surgimento das demais variantes em relação à original Encoder-Decoder. O interessante de realizar esse estudo é que permite analisar em que medida essas variantes podem ser vistas como otimização da arquitetura-mãe voltadas a diferentes finalidades, sendo diretamente relacionado ao tema central. Para isso, as leituras dos Papers que deixaram essas variantes populares, tornaram-se obrigatórias para essa Semana: "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding", do BERT, referente à variação Encoder, e "Improving Language Understanding by Generative Pre-Training", do GPT-1, referente à variação Decoder.
- Apesar dos aprendizados com a leitura do GPT-1, o artigo não deixa claro o porquê da escolha e nem sequer debate a comparação entre Decoder e Encoder-Decoder para geração de texto - no entanto, ele referencia o Paper que de fato propôs essa variante: "Generating Wikipedia by Summarizing Long Sequences". Portanto, também foi realizada a leitura dele.
- Assim, somando as conclusões dos experimentos do GPT-1 e do Paper que propôs o Transformers Decoder, foi possível chegar em seis pontos que começam a justificar a predominância dessa variante no backbone dos LLMs: "Redução no número de parâmetros", "Mais sinais de aprendizado (gradiente)", "Menos redundância nas próprias representações", "Maior escalabilidade em contexto longo", "Remoção da necessidade de Dados Paralelos" e "'Desbloqueio' da generalização com pouco dado supervisionado".
- Para avançar na tarefa do estabelecimento de um método de busca e critérios de seleção de Papers relacionados à otimização de LLMs, tomei como inspiração a seção de "Arquitetura" do Survey coletado na primeira Semana "A Survey of Large Language Models (2025)". Em suma, ela ajudou a planejar meu estudo subdividindo os artigos de otimização com base na área que atuam nas seguintes partes principais da arquitetura Transformers: Positional Embeddings, Atenção, FeedForward, Normalização, Função de Ativação e Macro Arquitetura.
- Além disso, decidi que cada estudo deverá ser mapeado em uma planilha com as seguintes colunas: Paper, Otimização, Onde atua, O que busca otimizar, Ano, Modelo(s) que usaram, Status, Fatores de Consolidação, Fatores de Não Consolidação e "Status de Estudo". A explicação de

cada coluna pode ser encontrada no documento da pesquisa.

- Essa planilha buscará mapear as inovações, logo, o ponto de partida para comparação será a arquitetura original Transformers.
- Por, a princípio, chamar uma proposta de “consolidado” ou “não consolidado” não é algo trivial, irei considerar, para esse trabalho em específico, que: propostas consolidadas são aquelas que foram incorporadas por pelo menos mais de um modelo de fronteira de grupo de pesquisas diferentes. Em caso de já ter caído em desuso, ainda será considerada como consolidada.
- Por critérios **iniciais** de fatores de consolidação, temos: “Melhoria na generalização”, “Mantém viabilidade ao escalar modelo” e “Melhoria na eficiência computacional sem degradar”. Já como critérios iniciais de fatores de não consolidação, temos: “Escalabilidade limitada", “Aumento de custo e complexidade Computacional sem ganho justificável”, “Melhoria restrita a benchmarks específicos” e “Ofuscado por propostas de ganho semelhante, mas que foram adotadas por modelos influentes”.
- Quanto a estratégia de busca, para evitar o risco de um início de busca lento e sem muita direção em relação às propostas de otimização relevantes, planejei guiar a busca da seguinte forma:
    - Levantar as melhorias do Transformer até o GPT-3 (definido como ponto inicial dos LLMs na Semana 2), seguindo a linha da família GPT (precursora dos LLMs).
    - Prosseguir do GPT-3 em diante, considerando principalmente os LLMs e famílias listados na Semana 2.
    - Anotar informações arquiteturais de cada modelo.
    - Explorar e analisar as melhorias aplicadas em cada um, mapeando na planilha.
    - Identificar outras propostas correlatas a partir dessas melhorias e analisá-las.
    - Mapear gradualmente os fatores de sucesso e de não adoção, ajustando as categorias propostas ao longo do estudo.
- Restrição de busca definidas inicialmente: Somente propostas que mantém o backbone “Transformer” como base, evitando as que buscam alternativas (Ex: Mamba); propostas focadas em otimização a nível de arquitetura, evitando as de nível de hardware (Ex: FlashAttention). Além disso, ao encontrar propostas classificadas como “não consolidadas”, será priorizado o estudo daquelas provenientes de centros de pesquisa reconhecidos (Big Techs ou laboratórios reconhecidos, como Mistral, DeepSeek, etc) ou que tenham sido explorados por no máximo um modelo/família de fronteira, visto que ainda deve se encaixar dentro da classificação de “Não Consolidado”.
- A pesquisa encontra-se na íntegra em: Semana3_pesquisa

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

Para a próxima Semana, planeja-se:

- Seguir os passos de busca e critérios de seleção definidos nesta Semana para fazer um levantamento das otimizações consolidadas feitas nos principais LLMs até o momento (definidos na Semana anterior). A partir desse mapeamento, deverá haver uma exploração inicial dentro de cada proposta consolidada para, ao fim, encontrar no mínimo uma proposta não consolidada para cada subdivisão de “inovação” definida nesta Semana.
- Realizar a criação da planilha sugerida, e colocar todos os levantamentos nela.
- Além de mapear, estudar a fundo, no mínimo, três propostas levantadas.

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

Documento de Apoio - Semana 3

*Referente ao documento “ Semana3_pesquisa ”

Identificação de Lacuna na Semana 2

Até aqui, para chegar ao estudo dos LLMs propriamente ditos, foi realizada uma revisão da evolução até a arquitetura-mãe “Transformer” [2] e uma visão geral da era pós-Transformer, destacando as duas gerações de modelos (PLM e LLM), conforme categorização apresentada nos dois surveys de referência [1] [6]. Utilizou-se ainda a classificação fundamental das variantes arquiteturais do Transformer - Encoder-only, Encoder-Decoder e Decoder-only - para compreender as principais características dos modelos que predominaram na geração PLM e que hoje predominam na geração LLM.

No entanto, considerando que a arquitetura original já pertence à categoria Encoder-Decoder, identificou-se uma lacuna: a compreensão da motivação que levou ao surgimento das demais variantes. Esse aspecto, não aprofundado na semana anterior, é relevante ao tema central deste trabalho, pois permite analisar em que medida essas variantes podem ser vistas como otimizações da arquitetura-mãe voltadas a diferentes finalidades.

Portanto, será necessário revisar as publicações originais que introduziram essas variantes: *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding* [3], para Encoder, e *Improving Language Understanding by Generative Pre-Training* [4], para Decoder. No entanto, será uma revisão muito mais direcionada para o entendimento da motivação e das modificações propostas, visto que é necessário prosseguir para o planejamento efetivo da Semana.

Anotações do artigo do GPT-1

As anotações da leitura do paper foram realizadas através de marca-texto e anotações por meio de comentários em determinadas partes. Ao baixar o documento, será

possível visualizar com essas informações. Tentei seguir o seguinte padrão de grife: em vermelho, aspectos, problemas ou soluções descritas de forma comparativa a outras arquiteturas anteriores; em amarelo, conceitos e definições diversas, mas fundamentais para o entendimento; em verde, anotações de explicações mais diretamente ligadas à arquitetura e como ela funciona.
“Improving Language Understanding by Generative Pre-Training” (GPT-1) grifado e com anotações: GPT1.pdf

Além das informações gerais aprendidas sobre as premissas, hipóteses, arquitetura e treinamento do modelo, considerando que li também pensando em tentar encontrar uma justificativa para entender o porquê usar Transformers Decoder-Only, fui frustrado ao perceber que na verdade o artigo nem chega a falar sobre isso. Na prática, a ideia de retirar a parte Encoder da rede original, e manter somente a parte Decoder, vem do artigo citado *Generating Wikipedia by Summarizing Long Sequences*.
“Generating Wikipedia by Summarizing Long Sequences” grifado e com anotações: decoder_only_generating_wikipedia.pdf

Relacionando com a discussão principais motivações por trás do surgimento da variante Decoder-Only, temos:

- Tarefa objetiva: aprendizado supervisionado para geração de sumarização. Usaram como input referências e resultados de pesquisa web em relação a um determinado artigo da Wikipedia, e colocaram esse próprio artigo como output desejado, simulando a ideia de sumarização humana: seleciona várias fontes principais, e depois agrega todas elas em um único texto coerente.
- A princípio, é uma tarefa Seq2Seq, onde é necessário mapear a sequência de tokens do input, para a sequência de tokens do output.
- Assim, para essa tarefa, testaram uma LSTM Seq2Seq com atenção, o Transformers-base Encoder-Decoder (T-ED) e, mesmos sendo uma tarefa Seq2Seq, testaram com uma variante Transformers Decoder (T-D), que consiste na mesma arquitetura, mas adaptada para não ter a parte encoder e nem outros módulos que dependiam dela, como Cross-Attention.

- Para adaptar o problema a esse modelo proposto, concatenaram a sequência de entrada (m) e saída (y) em uma única sequência, colocando apenas um token especial para separar ambas sequências. Com isso, o modelo não aprende só a prever o próximo token do resumo, mas também os próximos tokens do próprio input. Isso gera mais gradientes durante o treino, tanto na parte do input quanto da saída - isto é, o modelo “treina mais” a cada exemplo, o que ajuda a aprender melhor (no T-ED, o gradiente vem somente da sequência de output, enquanto no T-D, vem de ambas sequências, visto que ambas estão sendo reconstruídas). Outra vantagem é que o modelo fica “mais leve”, visto que seus parâmetros reduziram praticamente pela metade.
- Também levantaram a hipótese de que em tarefas monolíngues (texto em inglês -> texto em inglês), tanto o encoder quanto o decoder acabam reaprendendo representações de linguagem parecidas (gramática, estilo, coesão), tornando o processo mais redundante e a otimização mais difícil, e essa parte da otimizações mais difícil foi vista empiricamente com os experimentos.
- Empiricamente, viram que o T-ED melhora até ~500-1000 tokens, mas não consegue treinar de forma estável a partir de 2000 tokens. Já o Transformers-Decoder conseguiu boa performance e convergiu bem até 4.000 tokens. Além disso, propuseram variações dessa arquitetura Decoder-Only, onde o foco está em métodos de compressão da Atenção, principalmente por ela ter custo O(n^2), visando principalmente alcançar boa performance com sequências ainda mais longas - fato que conseguiram demonstrar, visto que conseguiu uma performance ainda melhor do que as outras arquiteturas, mesmo utilizando uma sequência de até 11.000 tokens.

Com todos esse fatores e teste empíricos levantados por este Paper e o do GPT-1, cheguei a conclusão que a variação Decoder-Only foi proposta, e vingou, porque:

1. Redução no número de parâmetros, mais eficiente - ao remover o encoder e o cross-attention, a arquitetura fica mais leve (menos parâmetros, menos custo computacional) sem perda de desempenho nos artigos que li, pelo menos em tasks da mesma língua.
2. Mais sinais de aprendizado (gradiente) - como o modelo prevê tanto tokens de entrada quanto de saída, os gradientes se propagam por mais posições, acelerando o aprendizado.

3. Menos redundância nas próprias representações - em tarefas monolíngues, encoder e decoder aprendem representações linguísticas muito semelhantes; o decoder-only elimina essa duplicação, facilitando a otimização.
4. Maior escalabilidade em contexto longo - seguindo o paper da geração da wikipedia, o T-D demonstrou bem mais estabilidade e qualidade em sequências longas, em detrimento do T-ED.
5. Remoção da necessidade de Dados Paralelos - com exploração do T-D pelo GPT-1 em Language Modeling, ou seja, com uma performance promissora em uma tarefa não supervisionada, é possível concluir que torna o modelo ainda mais escalável, pois é muito mais fácil e abrangente conseguir dados não paralelos.
6. “Desbloqueio” da generalização com pouco dado supervisionado - como o GPT-1 demonstrou, Language Modeling é uma excelente tarefa para aumentar a capacidade de generalização do modelo, permitindo se ajustar em outras tarefas com relativamente poucos dados supervisionados

Todos esses fatores começam a justificar a predominância do Transformers Decoder-Only em detrimento do Transformers Encoder-Decoder para tarefas de geração de texto (fora os fatores comparativos que vieram em Papers posteriores, como o do GPT-2 e GPT-3) mas não necessariamente justificam a falta de exploração desses modelos (pouquíssimos foram experimentados como um LLM em si). O próprio Survey aponta que há pouca investigação sistemática em larga escala sobre encoder-decoders, e que mais pesquisas seriam necessárias para compreender como diferentes arquiteturas e objetivos de pré-treino impactam a capacidade de LLMs.

Anotações do artigo do BERT

“BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding” grifado e com anotações: bert.pdf

Quanto ao BERT, por mais que não tenha comparado explicitamente com Encoder-Decoder, fica explícito que a escolha por não usar a parte Decoder do Transformer decorre do fato de que, pela natureza da tarefa de language modeling auto-regressiva, não é viável treinar um modelo bidirecional sem gerar data leakage, já que cada token teria acesso

a si mesmo e à informação futura, tornando a predição trivial - e a natureza bidirecional é justamente a hipótese que o Paper deseja testar para as diversas tarefas de NLP propostas. Além disso, mesmo que fosse possível, não faria sentido na prática, pois em um cenário real de geração o modelo nunca teria acesso a tokens futuros. Por isso, o BERT descarta o Decoder e adota apenas a pilha de Encoders, introduzindo o Masked Language Model (MLM) como alternativa para aprender representações bidirecionais profundas de forma controlada e úteis para tarefas de compreensão de linguagem.

Processo de elaboração dos critérios de pesquisa e seleção das otimizações

Para se ter um norte em como encontrar esses artigos de propostas de melhorias na arquitetura dos LLMs, e começar a pensar em como selecionar, bassei-me na seção “Architecture” do survey *A Survey of Large Language Models* [1], com o objetivo de ter uma visão geral das arquiteturas e melhorias principais (o próprio paper chama de *mainstream*) até a última revisão do paper, 11 de Março de 2025.

De forma geral, ele divide os LLMs quanto a arquitetura, no que chamou de arquiteturas típicas, subdivididas em Encoder-Decoder, Causal Decoder, Prefix Decoder, Mixtral of Experts e Emergent Architectures. Em suma, é reforçado a predominância do Causal Decoder como backbone dos LLMs em relação ao Encoder-Decoder e Prefix Decoder, e é citado o *Mixtral of Experts* (MoE) como uma possibilidade de modificação nessas arquiteturas com o objetivo de permitir dar escala ao modelo e manter um custo computacional constante. Quanto às *Emergent Architectures*, são citadas as propostas de backbones de LLMs que buscam uma alternativa ao Transformers como arquitetura-mãe, como o Mamba. Só discordo de colocarem a implementação do MoE nessa mesma prateleira de diferenciação, visto que é apenas uma implementação a nível de camada que pode ser usada em qualquer uma dessas arquiteturas.

Em seguida, expõe melhorias realizadas na arquitetura Transformer categorizadas em relação a em que parte elas estão sendo aplicadas, subdividindo em: posição da normalização, métodos de normalização, funções de ativação, position embeddings e atenção. Quanto a cada uma, são citados:

- Métodos de Normalização: LayerNorm, RMSNorm e DeepNorm.
- Posição da Normalização: Post-LN, Pre-LN e Sandwich-LN.
- Funções de Ativação: GeLU, GLU, SwiGLU e GeGLU.
- Position Embeddings: Absolute position embedding, Relative position embedding, Rotary position embedding e ALiBi.
- Atenção: Full Attention, Sparse Attention, Multi-query/Grouped-query attention, FlashAttention e PagedAttention. Nota: Não gostei dessas variações citadas, visto que está sendo misturado melhorias de arquitetura e melhorias de implementação a nível de hardware, que é o caso da FlashAttention. Ou seja, uma Grouped-query Attention pode ser implementada usando FlashAttention, por isso não gostei de deixar na mesma prateleira, ainda mais na seção de arquitetura

O restante da seção refere-se a “pre-training tasks” e “estratégias de decoding”, o que foge do objeto de interesse no momento. Essa breve leitura foi interessante principalmente por conta dessa subdivisão realizada para expor diferentes otimizações propostas ao Transformer, o que me ajudou a pensar em possibilidades de subdivisão do meu estudo.

Prosseguindo com o objetivo da Semana, para conduzir de forma estruturada meu estudo acerca das otimizações nas arquiteturas dos LLMs, propus uma planilha para ir mapeando tudo que for encontrando com base nas leituras. Ela seguirá um formato de mapeamento que segue as colunas:

- Paper: Artigo da otimização.
- Otimização: Nome da otimização proposta.
- Onde atua: Parte da arquitetura onde a otimização é aplicada. Opções definidas: Positional Embeddings, Atenção, FeedForward, Normalização, Função de Ativação e Macro Arquitetura. Macro Arquitetura refere-se em otimizações onde há uma mudança na arquitetura como um todo, como a própria ideia dos “Decoder-Only”.
- O que busca otimizar: Foco principal da otimização. Por exemplo: uma proposta de normalização normalmente buscará otimizar a estabilidade das ativações.
- Ano: ano de introdução da otimização no contexto de LLMs.
- Modelo(s) que usaram: Modelos de fronteira que implementaram a otimização em sua arquitetura.

- Status: Consolidado ou Não Consolidado. É uma palavra forte de ser utilizada, então será empregada com base na seguinte delimitação simplificada: se foi incorporada em pelo menos mais de um modelo de fronteira, de grupos de pesquisa diferentes, é uma otimização consolidada. Em caso de já ter caído em desuso, ainda será considerada como consolidada.
- Fatores de Consolidação: Campo ainda em definição. Propostas iniciais: “Melhoria na generalização”, “Mantém viabilidade ao escalar modelo”, “Melhoria na eficiência computacional sem degradar”
- Fatores de Não Consolidação: Campo ainda em definição. Propostas iniciais: “Escalabilidade limitada", “Aumento de custo e complexidade Computacional sem ganho justificável”, “Melhoria restrita a benchmarks específicos”, “Ofuscado por propostas de ganho semelhante, mas que foram adotadas por modelos influentes”.
- Status de Estudo: Concluído ou Não Concluído. Planejo mapear conforme for encontrando, mas é inviável estudar todos.

Neste trabalho, consideramos como modelo de fronteira todo modelo de linguagem que, no momento do seu lançamento, representou avanço significativo em escala, desempenho ou proposta arquitetural, sendo reconhecido na comunidade (acadêmica ou open-source) como referência técnica. Isso inclui modelos lançados por Big Techs e também por laboratórios independentes que tiveram forte impacto no ecossistema (ex: Mistral, DeepSeek, Falcon). Como normalmente são os modelos abertos que falam sobre a arquitetura, serão eles os principais considerados, salvo modelos fechados com reporte técnico que fala sobre a arquitetura.

Para estabelecer um ponto de partida, considerei como referência inicial o próprio Transformer original [2] Nesse sentido, o GPT-1 já pode ser interpretado como uma primeira otimização aplicada aos modelos de linguagem, por inaugurar a linha precursora dos LLMs. Assim, o baseline adotado é:

- Positional Embeddings: codificações senoidais fixas, adicionadas aos embeddings de tokens, não treináveis.
- Atenção: Scaled Dot-Product Multi-Head Attention.
- Feedforward: MLP position-wise de 2 camadas totalmente conectadas.

- Função de Ativação: ReLU.
- Normalização: Post-LayerNorm.
- Macro Arquitetura: Transformer Encoder-Decoder.

A partir desse ponto, a estratégia de mapeamento das melhorias arquiteturais consiste em partir dos principais LLMs lançados ao longo do tempo. Isso oferece uma base mais sólida e direcionada do que buscar otimizações de forma dispersa, permitindo identificar quais modificações de fato se consolidaram. A compreensão de por que determinadas melhorias foram aceitas em modelos de expressão é um passo essencial para, em seguida, ter a capacidade de analisar por que outras propostas não tiveram adoção, tornando o contraste mais claro.

Ao detalhar as melhorias empregadas nos principais LLMs, muitas vezes os artigos não explicam em profundidade a motivação ou o funcionamento das técnicas escolhidas, limitando-se a citar os trabalhos de origem. Assim, o caminho natural é destrinchar essas referências, voltando aos artigos originais em que a proposta foi apresentada. Neles, em geral, há comparações com alternativas contemporâneas, que podem incluir propostas de relevância teórica ou experimental que, na prática, não se consolidaram. Esse processo cria uma via mais sólida para identificar tanto as inovações adotadas quanto as tentativas que ficaram pelo caminho. Assim, um guia de ação seria:

- Levantar as melhorias do Transformer até o GPT-3 (definido como ponto inicial dos LLMs na Semana 2), seguindo a linha da família GPT (precursora dos LLMs).
- Prosseguir do GPT-3 em diante, considerando principalmente os LLMs e famílias listados na Semana 2.
- Anotar informações arquiteturais de cada modelo.
- Explorar e analisar as melhorias aplicadas em cada um.
- Identificar outras propostas correlatas a partir dessas melhorias e analisá-las.
- Mapear gradualmente os fatores de sucesso e de não adoção, ajustando as categorias propostas ao longo do estudo.

Optei por restringir, neste momento, o escopo da pesquisa a propostas que mantêm o backbone Transformer como base. Assim, artigos que introduzem arquiteturas alternativas,

como o Mamba ou outros State Space Models, não serão considerados. A justificativa é que os LLMs atuais ainda têm os Transformers como núcleo predominante, enquanto essas propostas representam linhas paralelas de investigação. Embora sejam relevantes e promissoras, não se enquadram como objeto de estudo neste recorte, especialmente considerando as limitações de tempo disponíveis. Além disso, a análise será focada em otimizações arquiteturais. Portanto, não serão incluídas melhorias de nível de implementação (ex.: FlashAttention) nem avanços centrados em dados de pré-treino ou métodos de treinamento, que configuram outras frentes de pesquisa igualmente importantes, mas externas ao objetivo imediato deste trabalho.

Além desses critérios de exclusão, entre as propostas classificadas como “não consolidadas”, será priorizado o estudo daquelas provenientes de centros de pesquisa reconhecidos (Big Techs ou laboratórios independentes de forte impacto, como Mistral, DeepSeek) ou que tenham sido exploradas em um frontier model ou dentro de uma família somente (critério explicitado pelo motivo de que se for empregado em mais de um frontier model de grupos de pesquisa diferentes, entra no critério de consolidado), ou em modelos subsequentes menores. Essa priorização visa garantir um nível mínimo de relevância para análise, evitando dedicar esforços a propostas sem validação prática ou reconhecimento significativo. O número final máximo de papers analisado será dinâmico, de acordo com o nível de evolução.

Referências

[1] ZHAO, Wayne Xin et al. A Survey of Large Language Models. arXiv, , 11 mar. 2025. Disponível em: <http://arxiv.org/abs/2303.18223>. Acesso em: 11 set. 2025

[2] VASWANI, Ashish et al. Attention Is All You Need. arXiv, , 2 ago. 2023. Disponível em: <http://arxiv.org/abs/1706.03762>. Acesso em: 11 set. 2025

[3] DEVLIN, Jacob et al. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In: BURSTEIN, Jill; DORAN, Christy; SOLORIO, Thamar (orgs.). NAACL-HLT 2019. Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1

(Long and Short Papers). Minneapolis, Minnesota: Association for Computational Linguistics, jun. 2019. Disponível em: <https://aclanthology.org/N19-1423/>. Acesso em: 17 set. 2025

[4] RADFORD, Alec et al. Improving Language Understanding by Generative Pre-Training. [S.d.]. Disponível em: <https://www.mikecaptain.com/resources/pdf/GPT-1.pdf>. Acesso em: 17 set. 2025

[5] BROWN, Tom B. et al. Language Models are Few-Shot Learners. arXiv, , 22 jul. 2020. Disponível em: <http://arxiv.org/abs/2005.14165>. Acesso em: 17 set. 2025

[6] INAEE, Shervin et al. Large Language Models: A Survey. arXiv, , 23 mar. 2025. Disponível em: <http://arxiv.org/abs/2402.06196>. Acesso em: 11 set. 2025

[7] LIU, Peter J. et al. Generating Wikipedia by Summarizing Long Sequences. arXiv, , 30 jan. 2018. Disponível em: <http://arxiv.org/abs/1801.10198>. Acesso em: 17 set. 2025

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: 25 de set. de 2025

Participantes da Entrega [matriculados em Residência em IA]:

GUSTAVO LUIZ BUENO PEREIRA

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

Tema central: " 'Large Language Models' e suas Otimizações Arquiteturais"

Na Semana anterior, consolidamos o estudo histórico da base dos LLMs, entendendo a motivação das variações fundamentais da arquitetura Transformer. Também definimos critérios de busca e seleção de papers de otimizações arquiteturais e estruturamos uma planilha para mapear as propostas encontradas.

Durante a quarta Semana da Residência:

- Pesquisei linhas do tempo de LLMs para complementar a investigação iniciada na segunda Semana. Encontrei um trabalho que listava uma linha do tempo "principal" com pouco mais de 70 modelos e, em outra tabela, mais de 600 LLMs. A partir dessa lista, selecionei 102 modelos de destaque, cujo objetivo é extrair a arquitetura para dar continuidade ao estudo. Modelos sem qualquer informação pública sobre arquitetura (paper ou blog técnico) serão excluídos conforme pesquisados.
- Planilhei esses modelos selecionados, com as colunas: "Modelo", "Centro de Pesquisa", "Paper", "Data" (MM/YYYY), "Status de Extração da Arquitetura" e "Prioridade".
- Iniciei a exploração da arquitetura dos modelos, seguindo este procedimento:
  - Localizar o paper ou blog e identificar a seção de arquitetura.
  - Copiar todas as informações relevantes para um documento de apoio (ainda a ser formatado para ser inserido em algum Termo de Entrega). Penso que, além de ter o potencial de ser um dicionário de arquitetura, pode contribuir na minha análise de fatores de consolidação ou não consolidação, visto que terei uma visão de qual dos principais modelos utilizam determinadas otimizações.
  - Identificar as otimizações arquiteturais adotadas em relação ao baseline definido (Transformer original) e mapear na planilha, caso ainda não estejam registradas.
  - Efeito "Retropropagação": quando um Paper (seja de otimização ou de um novo LLM) cita explicitamente arquiteturas ou técnicas de outros modelos, esses também são incluídos na lista de análise.
- Até o momento, foram extraídos a arquitetura de 11 modelos, e 16 técnicas de otimização arquitetural foram mapeadas.
- Por uma mudança de enfoque, ainda não estou classificando se cada otimização está consolidada ou não, nem os fatores que explicam esse status. Essa etapa será mais adequada em estágios posteriores. No momento, a prioridade é registrar: "paper de origem", "paper do LLM onde a otimização foi encontrada", "data" (MM/YYYY) e "parte da arquitetura em que atua".

- Também houve a leitura e anotação do paper “GLU Variants Improve Transformer” para o estudo das otimizações “GEGLU Activation” e “SwiGLU Activation”.
- Documento da pesquisa: Semana4_pesquisa

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

Para a próxima Semana, planeja-se:
- Continuar o mapeamento das otimizações arquiteturais com base nos modelos selecionados;
- Iniciar a priorização das otimizações, distinguindo entre:
  - aquelas que terão estudo aprofundado;
  - aquelas que receberão apenas análise superficial, limitada à identificação de local de atuação e objetivo;
- Estruturar um padrão para o documento de anotação das arquiteturas de cada modelo analisado.

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

ACEITE DA ENTREGA:

CEDRIC LUIZ DE CARVALHO: Go!

Documento de Apoio - Semana 4

*Referente ao documento “ Semana4_pesquisa ”

Busca por Otimizações Arquiteturais nos LLMs

Comecei pesquisando por linhas do tempo de LLMs para complementar a pesquisa da Semana 2, e comecei a montar uma planilha de alguns LLMs selecionados, conforme algumas obras que encontrei na internet [1][2]. Os trabalhos destacam pouco mais de 70 modelos principais, mas agrupam mais de 600 LLMs - portanto, importei a maioria dos principais, e adicionei mais alguns que achei que não poderia faltar pelo nome mais popular e pelo centro de pesquisa. Totalizaram 98 modelos inicias, os quais foram planilhados com as colunas: “Modelo”, “Centro de Pesquisa”, “Paper”, “Data” (MM/YYYY), “Status de Extração da Arquitetura” e “Prioridade”.

Modelos sem qualquer informação pública sobre arquitetura (paper ou blog técnico) serão excluídos conforme pesquisados. Além disso, por conta de um efeito de “retropropagação”: quando um Paper (seja de otimização ou de um novo LLM) cita explicitamente arquiteturas ou técnicas de outros modelos, esses também são incluídos na lista de análise. Portanto, a lista não é fixa.

Assim, o procedimento foi:

- Localizar o paper ou blog e identificar a seção de arquitetura.
- Copiar todas as informações relevantes para um documento de apoio (ainda a ser formatado para ser inserido em algum Termo de Entrega). Penso que, além de ter o potencial de ser um dicionário de arquitetura, pode contribuir na minha análise de fatores de consolidação ou não consolidação, visto que terei uma visão de qual dos principais modelos utilizam determinadas otimizações.
- Identificar as otimizações arquiteturais adotadas em relação ao baseline definido (Transformer original) e mapear na planilha, caso ainda não estejam registradas.

A Semana foi finalizada com:

- 102 modelos levantados;

- 11 arquiteturas de modelos anotadas;
- 16 técnicas de otimização mapeadas.

Este documento reúne as informações acerca dos artigos estudados nesta Semana: Otimizações Arquiteturais - Language Modeling with Gated Convolutional Networks e Searching for Activation Functions

Referências

[1] THOMPSON, Alan D. Models Table. Dr Alan D. Thompson – LifeArchitect.ai, 25 fev. 2023. Disponível em: <https://lifearchitect.ai/models-table/>. Acesso em: 24 set. 2025

[2] THOMPSON, Alan D. Timeline of AI and language models. Dr Alan D. Thompson – LifeArchitect.ai, 9 jul. 2021. Disponível em: <https://lifearchitect.ai/timeline/>. Acesso em: 24 set. 2025

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: 2 de set. de 2025

Participantes da Entrega [matriculados em Residência em IA]:

GUSTAVO LUIZ BUENO PEREIRA

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

Tema central: " 'Large Language Models' e suas Otimizações Arquiteturais"

Na **Semana anterior**, foi iniciado o processo de busca e seleção de Papers de LLMs, e o processo de mapeamento das otimizações realizadas em cada um deles, sendo concluída com 102 LLMs selecionados e, dentro deles, foram extraídos as arquiteturas de 11 modelos, resultando em 16 novas otimizações diferentes encontradas. Também foi iniciado o estudo de algumas dessas otimizações.

Durante a quinta Semana da Residência:

- Foi realizado uma repriorização na extração da arquitetura dos modelos, resultando em 30 LLMs priorizados, já excluindo aqueles com arquitetura fechada dessa lista;
- Ao finalizar a análise de cada um deles, tivemos, ao fim desta Semana:
  - 19 modelos analisados, totalizando 30 modelos;
  - 18 novas otimizações mapeadas, totalizando 34.
- Com base na frequência de aparição de cada otimização, foram priorizados com classificação "Alta", 6 otimizações, "Média", 4 otimizações, e o restante como "Baixa".
- Planilha de otimizações arquiteturais e LLMs: Mapeamento de Otimizações

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

Para a próxima Semana, planeja-se:

- A partir desta Semana, preencher a planilha resultante de otimização mapeadas em segundo plano;
- Estudar as otimizações priorizadas em segundo plano, ou seja, em paralelo, mas com prioridade menor do que o foco da Semana;
- Iniciar o estudo da arquitetura do LLM "Phi-4-mini-flash-reasoning" e das otimizações ligadas a ele, além de preparar o ambiente (busca e configuração de frameworks) para realizar testes de inferência com este modelo. Motivação: é uma arquitetura recente (Julho deste ano), relativamente pequena (3.8B parâmetros, potencialmente facilitando e viabilizando testes), lançada pela Microsoft, e que reúne diversas otimizações que criei interesse de estudo - o fato de ser uma arquitetura híbrida (Mamba e Transformers), inclusive, me chamou a atenção. Esta análise de uso

e possibilidades com esse modelo será crucial para determinar possíveis futuras direções dentro da Residência.

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

Durante a quinta Semana, estive na conferência BRACIS, impactando o ritmo de estudo.

ACEITE DA ENTREGA:

CEDRIC LUIZ DE CARVALHO: Go!

Documento de apoio - Semana 5

Neste link, encontramos a planilha de mapeamento tanto dos LLMs selecionados (Página “LLMs”), quanto das otimizações (Página “Otimizações”): Mapeamento de Otimizações. Devido à mudança de foco, foram mantidas somente as colunas: Modelo, Centro de Pesquisa, Paper, Data, Status de Extração da Arquitetura, Prioridade, Notas - e nem todas foram preenchidas. Já na planilha de LLMs, foram mantidas as colunas Modelo, Centro de Pesquisa, Paper, Data, Status de Extração da Arquitetura, Prioridade.

O principal valor foi, além de uma melhor compreensão de como foram evoluindo as arquiteturas, a descoberta do lançamento do “Phi-4-mini-flash-reasoning”, que chamou muita atenção devido à sua arquitetura, tanto que mudou o direcionamento do meu trabalho, e me permitiu o aprofundamento em algo que despertou um interesse que, a princípio, havia descartado justamente por não ter visto a arquitetura híbrida empregadas em modelos de Big Techs, e o Phi mostrou o contrário disso. Portanto, foi uma etapa crucial da minha Jornada, mesmo que não tenha sido completa conforme a intenção inicial.

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: 9 de out. de 2025

Participantes da Entrega [matriculados em Residência em IA]:

GUSTAVO LUIZ BUENO PEREIRA

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

Tema central: " 'Large Language Models' e suas Otimizações Arquiteturais"
Nas cinco primeiras Semanas da Residência, foi conduzido um estudo histórico desde os primeiros modelos de Language Modeling, até chegar no modelo que se tornou a principal base para as duas últimas gerações de LLMs: Transformers. A partir dele, levantou-se o estudo das suas principais variantes, e da linha do tempo dos LLMs que vieram após o mesmo; desta linha, buscou-se entender e mapear as otimizações arquiteturais realizadas por cada um dos modelos, e ficou evidente que existem determinadas limitações/fraquezas conhecidas da arquitetura-mãe, e cada um se propõe a explorar e mitigar algumas delas. Durante o estudo, fiquei especialmente interessado em um dos modelos: Phi-4-mini-flash-reasoning, e o motivo se dá por ser um modelo recente, de Julho deste ano, desenvolvido pela Microsoft, e que, surpreendentemente, implementa uma **arquitetura Híbrida**, alternando entre blocos de Transformer e Mamba. Particularmente, já havia pensando que esse tipo de **Arquitetura híbrida** tinha sido deixado de lado, justamente por não ter visto nenhum grande modelo aberto usá-la, e os menores que usaram não tiveram uma performance que justificasse seu uso. Portanto, chamou a minha atenção ver a Microsoft aplicando-a pela primeira vez na sua família de modelos Phi, e então deixei o estudo das demais otimizações em segundo plano, e direcionei esforços para este modelo.

Portanto, durante a sexta Semana da Residência:

- Realizei a leitura e anotações do Paper que introduz o Phi-4-mini-flash-reasoning (Decoder-Hybrid-Decoder Architecture for Efficient Reasoning with Long Generation, 2025). Dado a diferença de arquitetura para as dos demais que estudei até o momento, havia uma certa lacuna de conhecimento quanto a determinados conceitos, mas a leitura foi finalizada com sucesso. Dentro deste Paper, existem diversos experimentos comparativos, onde os resultados também chamaram minha atenção, por exemplo: em uma série de benchmarks, este modelo superou os anteriores Phi-4-mini e Phi-4-mini-reasoning, os quais estão entre os melhores LLMs "pequenos", também chamados de Small Language Models. Além disso, ainda foi constatado um treinamento mais eficiente, ou seja, menos computação e dados para chegar em determinado ponto de performance.
- Portanto, o Paper como um todo me inspirou a direcionar as atenções da Residência para os modelos de **Arquitetura Híbrida** (quanto ao tipo de bloco principal usado). Ciente de que sempre terá uma infinidade de otimizações a serem estudadas, senti que para permitir o direcionamento a esse subconjunto de modelos, teria que, no mínimo, entender o principal núcleo deles até então: o bloco Mamba, um State Space Model.
- A partir disso, comecei a leitura e anotações do paper que introduz o Mamba (Mamba: Linear-time

sequence modeling selective state spaces, 2023). Lendo as principais seções que explicam as motivações e o bloco por si só, considerei uma leitura extremamente complicada para mim, e percebi que, antes, precisava entender como os State Space Model evoluíram até chegar no Mamba. Para isso, pesquisei vídeos e blogs que explicam essa evolução antes de explicar o Mamba propriamente dito. Ainda achei de difícil compreensão, mas cheguei a um nível suficiente de entendimento do todo para poder prosseguir com o uso prático.

- Na última Semana, havia proposto preparar um ambiente para pelo menos permitir realizar inferências com o Phi-4-mini-flash-reasoning, já imaginando que não seria uma tarefa fácil, justamente por ter uma arquitetura que foge do "convencional", e por ser um modelo relativamente novo, com quase nenhum conteúdo na internet. De fato, testando os principais frameworks de inferência: "transformers", "vLLM" e "Ollama", nenhum tinha um suporte por padrão ao modelo. Percebi então que a Microsoft orienta baixar determinados módulos separados, para determinada versão do transformers e do pytorch. Também vi que ela criou uma branch para permitir suporte do vLLM a esse modelo, e que inclusive foi "mergeada" na principal há poucos meses, mas aparentemente não é uma modificação que já vem na versão estável por padrão, e provavelmente terei que "forçar" o seu uso. Portanto, ainda não tive sucesso em inferência, mas pretendo conseguir. Caso não funcione, durante o estudo dessa Semana, encontrei um outro modelo recente (2025) de **Arquitetura Híbrida**: o Falcon-H1, que também alcançou resultados interessantes - portanto, existe a possibilidade de tentar explorá-lo.
- A pesquisa encontra-se em: Semana 6

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

Para a próxima Semana, planeja-se:

- Finalizar teste de inferência de arquiteturas híbridas: priorizar Phi-4-mini-flash-reasoning, e partir para o Falcon-H1, caso não funcione.
- Implementar a arquitetura Mamba em Python e, em caso de sobrar tempo, já implementar como uma Arquitetura Híbrida, consolidando os conhecimentos acerca desse tema.

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

ACEITE DA ENTREGA:

CEDRIC LUIZ DE CARVALHO: Go!

Documento de apoio - Semana 6

*Referente ao documento “ Semana 6 ”

Anotações “Decoder-Hybrid-Decoder Architecture for Efficient Reasoning with Long Generation” (Phi4-mini-flash-reasoning)

Inicio a leitura conhecendo alguns requisitos para o bom entendimento do Paper: *State Space Models* (SSMs), *Linear Attention* e Modern *Recurrent Neural Networks* - elas demonstram resultados promissores para uma modelagem de sequências mais eficiente do que o Transformers, visto que processam em O(L) - linear no tamanho da sequência, ao passo que o Transformers processa em tamanho quadrático. Apesar disso, ainda mantém uma performance inferior, levando à criação dos modelos híbridos, que buscam combinar a atenção com os SSMs, no objetivo de alcançar a performance Transformers com a eficiência dos SSMs. YOCO[13] é citado como uma dessas arquiteturas.

A arquitetura

Basearam-se no *Gated Linear Units* (o qual já foi estudado anteriormente), *Gated Attention Units*[14] e SSMs (*Mamba*[11], *Gated Linear Attention Transformers with Hardware-Efficient Training*[15], Gated Delta Networks: Improving Mamba2 With Delta Rule[17]), e propuseram o *Gated Memory Unit* (GMU), que toma a representação de entrada da camada atual, o estado da memória da camada anterior como entrada, e produz as representações gated com projeções aprendidas (mecanismo de compartilhamento eficiente de memória entre camadas). A junção das GMUs ao cross-decoder do YOCO, com o uso do Samba[16] como self-decoder, produz a arquitetura proposta, SambaY. Um dos grandes problemas da arquitetura YOCO , mesmo com seu ganho de eficiência em prompts com longas sequências, é o grande custo de entrada/saída da memória de atenção para as cross-attentions durante a geração - para LLMs modernos, que geram cadeias de pensamento extensivamente longas, torna-se uma limitação ainda maior. Assim, ao substituir cerca de 50% das cross-attentions por GMUs, torna o processo de geração mais

eficiente, mantém a complexidade de tempo linear do pre-filling (etapa inicial de inferência em que o modelo lê todo o prompt (os tokens de entrada) e constrói as representações e o cache K/V que serão usados depois na geração token-a-token) e remove a necessidade de um positional encoding.

Na prática, GMU é expressa como:

$$y_l = (m'_l \odot \sigma(W_1 x_l))W_2$$

"Memória" são as hidden representations das camadas anteriores - memory state $m'_l \in \mathbb{R}^{d_h}$, onde l' é a camanda anterior à l. GMU opera sobre esse memory state e o input hidden state atual, $x_l \in \mathbb{R}^{d_m}$. A saída é produzida por esse mecanismo de gating modulado por projeções aprendíveis (garante compatibilidade dimensional; aprende uma combinação linear útil das features resultantes do gating; atua como uma cabeça de saída da unidade GMU).

- σ é a função de ativação SiLU[18].
- d_h é a dimensão interna SSM.
- $\sigma(W_1 x_l)$ funciona como um filtro adaptativo que diz: "Para este token e neste contexto atual, quero reforçar certos canais da memória passada e atenuar outros - e o quanto usar da memória está condicionado à entrada atual". O GMU funciona de maneira análoga à atenção no sentido de que, com base no contexto atual (query), ele recalibra dinamicamente a importância das informações anteriores (memória). No entanto, enquanto a atenção aprende pesos entre tokens (token-to-token), o GMU aprende pesos entre camadas (layer-to-layer), canal por canal, de forma muito mais leve e eficiente.
- Também é possível adicionar RMSNorm antes da projeção final para melhorar a estabilidade do treino.

A pergunta que o GMU tenta responder é "Como posso reutilizar informações já processadas (de camadas anteriores), sem precisar recalcular atenção ou MLPs inteiros, e ainda permitir que o modelo decida quanto dessa memória usar em cada token?" - é como um atalho treinável e contextual entre camadas.

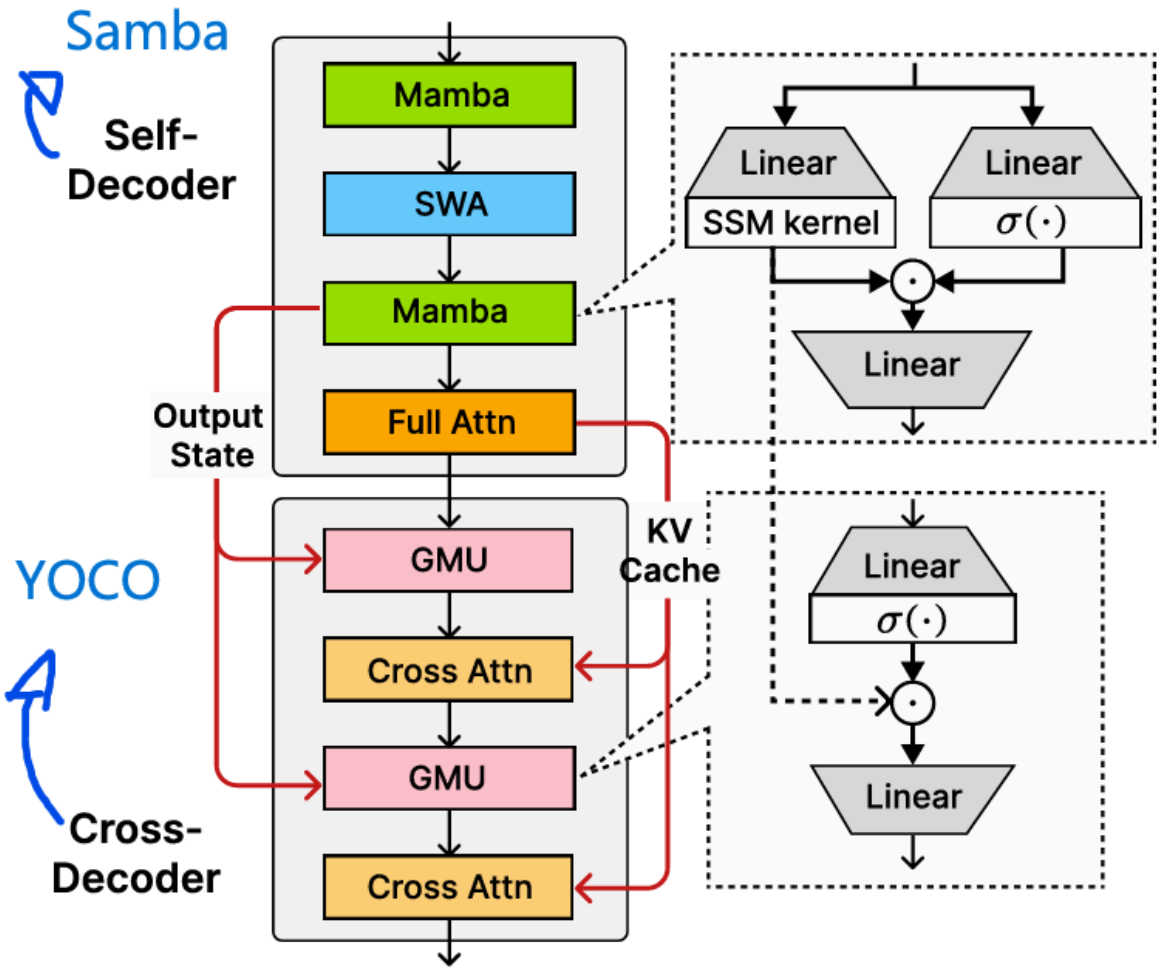


Figura 1: Arquitetura do SambaY [1].

É possível visualizar que agora está meio a meio a quantidade de GMU layers e Cross Attention Layers, onde o ganho está no fato de que GMU possui um custo constante $O(d_h)$ e as Cross Attentions, custo linear à sequência $O(d_{kv}N)$. O ganho do GMU cresce conforme o prompt fica mais longo, visto que a Cross Attention escala linearmente com N, enquanto o GMU é constante - ou seja, para ter ganho, $N >> d_h/d_{kv}$. As camadas de self attentions usam GQA (tamanho de grupo 4). O tamanho da Slide Windows Attention ficou 256, e a Head dimension = 128.

Experimentos

Nos experimentos de scaling (Seção 3.1), os autores concluem que, embora Transformers e modelos híbridos tenham capacidade expressiva semelhante em teoria, o SambaY apresenta melhor eficiência de aprendizado: ele reduz a validation loss mais rapidamente e alcança o mesmo limite teórico com muito menos dados e computação. Essa conclusão vem de curvas de scaling law obtidas em pré-treino sobre dados open-source genéricos, sem avaliar tarefas específicas - ou seja, o foco é puramente no comportamento de convergência e não em métricas de downstream performance.

Seção 3.2: Recuperação Eficiente de Contexto Longo: buscou determinar a capacidade das arquiteturas híbridas de recuperar informações em contextos longos com janelas de atenção deslizantes (SWA) de tamanho mínimo, uma característica crucial para tarefas de raciocínio complexas. O método consistiu em pré-treinar modelos de 1 bilhão de parâmetros em conjuntos de dados de contexto longo (ProLong-64k) com sequências de 32 mil tokens, variando o tamanho da SWA. As arquiteturas SambaY, SambaY+DA, Samba+YOCO e TransformerLS foram comparadas em benchmarks de recuperação de contexto longo, como Phonebook e RULER. A conclusão foi que os modelos híbridos, especialmente as variantes SambaY, superaram consistentemente as arquiteturas puramente baseadas em Transformer, mesmo com janelas de atenção significativamente menores. A grande vantagem do SambaY e SambaY+DA foi alcançar um forte desempenho de recuperação de contexto longo com janelas menores (256 e 512, respectivamente), o que aumenta a eficiência. Os modelos baseados em Transformer indicaram que eles podem exigir volumes de dados substancialmente maiores para treinar eficazmente a capacidade de contexto longo.

Seção 3.3: Pré-treinamento em Larga Escala com Dados de Alta Qualidade: valida a viabilidade da arquitetura proposta em uma escala ainda maior, criando um modelo protótipo chamado Phi4-mini-Flash. Este modelo de 3,8 bilhões de parâmetros, baseado na arquitetura SambaY+DA (SambaY com "Differential Attention"), foi pré-treinado em 5 trilhões de tokens de um corpus de dados de alta qualidade. O desempenho do Phi4-mini-Flash foi

comparado diretamente com o modelo de base Phi4-mini em uma variedade de tarefas, incluindo conhecimento geral (MMLU), codificação (MBPP) e raciocínio (GSM8K). A conclusão foi que o Phi4-mini-Flash superou consistentemente o Phi4-mini na maioria dos benchmarks, obtendo ganhos notáveis em tarefas intensivas de conhecimento e codificação. A principal vantagem demonstrada foi a capacidade de alcançar um desempenho superior mantendo uma eficiência computacional substancialmente maior durante a inferência. Uma desvantagem mencionada foi a ocorrência de instabilidade durante o treinamento (divergência de perda), que precisou ser mitigada com técnicas como suavização de rótulos e dropout de atenção, indicando que o processo de otimização ainda pode ser aprimorado.

Phi-4-Mini consiste em 32 camadas Transformer com tamanho de estado oculto de 3.072, com tied input/output embeddings. Cada bloco Transformer inclui um mecanismo de atenção baseado em Group Query Attention (GQA). O modelo emprega 24 query heads e 8 k/v heads, reduzindo o consumo de cache KV para um terço de seu tamanho padrão. Além disso, na configuração RoPE, uma dimensão RoPE fracionada é usada, garantindo que 25% da dimensão do cabeçalho de atenção permaneça independente da posição. Esse design oferece suporte a um tratamento mais suave de contextos mais longos.

Seção 3.4: Raciocínio Eficiente com Geração Longa: avalia a eficácia do modelo em tarefas de raciocínio complexas que exigem a geração de respostas longas (cadeias de pensamento). Para isso, o modelo Phi4-mini-Flash foi continuamente treinado com dados de destilação para criar o Phi4-mini-Flash-Reasoning. Este modelo foi então comparado com sua contraparte, o Phi4-mini-Reasoning, e outros modelos de código aberto em benchmarks de raciocínio desafiadores como AIME24/25 e Math500. A eficiência (latência vs. vazão) também foi medida em cenários de geração longa. O resultado foi que o Phi4-mini-Flash-Reasoning superou significativamente o Phi4-mini-Reasoning nas tarefas de raciocínio, mesmo sem passar por uma etapa final de treinamento com aprendizado por reforço. A vantagem mais marcante foi sua eficiência: o modelo alcançou uma vazão até 10 vezes maior em cenários de geração longa (prompt de 2k, geração de 32k), mostrando ganhos práticos substanciais. Uma desvantagem implícita é que a implementação atual do "Differential Attention" no framework de inferência vLLM não foi otimizada, deixando espaço para melhorias futuras de velocidade para alcançar a eficiência máxima do SambaY puro.

Observações

Ao finalizar a leitura desse Paper, me interessei ainda mais pela ideia de modelos híbridos quanto ao tipo de bloco utilizado. Além disso, descobri que modelos híbridos podem ser divididos em duas abordagens principais: inter-layer e intra-layer. Na hibridização inter-layer, como ocorre nos modelos Samba e Jamba, os blocos de diferentes tipos (por exemplo, atenção e state space) são organizados em camadas distintas. Assim, uma camada inteira realiza atenção e a seguinte realiza operações baseadas em state space, o que permite que cada tipo de camada seja otimizada separadamente e maximize o uso do hardware. Já na hibridização intra-layer, como no modelo Hymbra, diferentes mecanismos são combinados dentro da mesma camada, geralmente em nível de head, com algumas heads executando atenção e outras operando com state space models. Essa estratégia oferece uma integração mais profunda entre as representações, mas introduz um gargalo de eficiência: como cada head tem custos computacionais distintos, a velocidade total de processamento é limitada pela head mais lenta. Isso reduz a utilização teórica da GPU em comparação à abordagem inter-layer, que mantém blocos homogêneos e mais fáceis de paralelizar.

OBSERVAÇÃO: Nas seções seguintes serão utilizadas imagens do autor Maarten Grootendorst [3][6]

Anotações Mamba: Linear-Time Sequence Modeling with Selective State Spaces

Diversas arquiteturas subquadráticas, como linear attention, gated convolution, recurrent models e structured state space models (SSMs), surgiram para tentar resolver a ineficiência computacional dos transformers em sequências longas. Mesmo assim, esses modelos ainda não atingiram o mesmo desempenho dos transformers em áreas importantes como linguagem natural. O principal motivo apontado é a dificuldade desses modelos em realizar raciocínio baseado em conteúdo (content-based reasoning), algo que a self-attention

faz bem. Além disso, a self-attention tradicional tem duas limitações conhecidas: não consegue modelar dependências além de uma janela finita e apresenta custo quadrático em relação ao tamanho dessa janela, o que torna o processamento caro em contextos longos.

O conceito de state space models (SSMs) vem da década de 1960, com o trabalho *A New Approach to Linear Filtering and Prediction Problems*, de Rudolf E. Kalman, que introduziu o Kalman Filter. Anos depois, esse princípio foi reinterpretado para modelagem de sequências, dando origem aos structured state space sequence models (S4), que unem ideias de RNNs e CNNs dentro do formalismo contínuo dos modelos de espaço de estado. Essa linha de pesquisa é representada por trabalhos como *Combining Recurrent, Convolutional*, and *Continuous-time Models with the Linear State Space Layer* (2021) e *Efficiently Modeling Long Sequences with Structured State Spaces* (2022).

Dentro dessa evolução, o Mamba aparece como o primeiro modelo de tempo linear que realmente alcança desempenho comparável ao dos transformers, tanto em pré-treinamento (perplexity) quanto em avaliações downstream. Ele apresenta três pontos principais: (i) high quality, pois a seletividade (selectivity) garante bom desempenho em modalidades densas como linguagem e genômica; (ii) fast training and inference, já que o custo computacional e de memória cresce de forma linear com o tamanho da sequência no treinamento, e a inferência autoregressiva é constante por passo, sem necessidade de manter cache de tokens anteriores; e (iii) long context, pois a combinação entre qualidade e eficiência permite lidar com contextos muito longos, chegando a cerca de 1 milhão de tokens. O Mamba, portanto, representa um avanço importante por unir eficiência linear e desempenho equivalente ao de transformers, mostrando que é possível manter qualidade em tarefas complexas com custo bem menor. Neste link, encontramos o repositório da implementação original: Github.

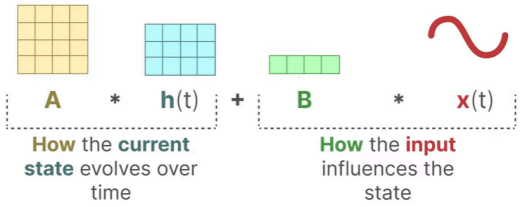
Structured State Space Models (S4)

State Space é uma forma matemática de representar um problema como um sistema composto por possíveis estados. Uma state representation descreve como o sistema se comporta em um determinado instante. Assim, um State Space Model tem como objetivo prever o próximo estado potencial do sistema.

Seja h o hidden state, que representa o conhecimento relevante sobre o mundo. Seja x a sequência contínua de entrada, ou seja, as observações obtidas a cada instante. h' representa a derivada do hidden state, isto é, como o estado está evoluindo ao longo do tempo. O objetivo é prever a sequência contínua de saída y, que corresponde à ação ou observação seguinte ideal.

Um SSM parte da suposição de que sistemas dinâmicos podem ser previstos a partir do seu state no tempo t por meio de duas equações. A dinâmica de um SSM é analiticamente controlada, pois se baseia em equações fechadas que descrevem o comportamento temporal do modelo. O state pode ser representado com a equação:

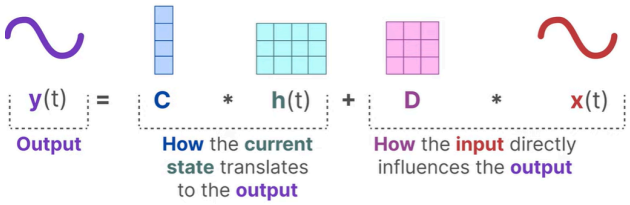
$$h'(t) = Ah(t) + Bx(t)$$



É como se a equação dissesse: "Como o state h(t) mudou através de uma matriz fixada A, baseando-se em como o input x(t) influencia o state através de uma matriz fixada B". Ou seja, as duas partes da equação, juntas, atualizam o state com a nova informação (entrada atual).

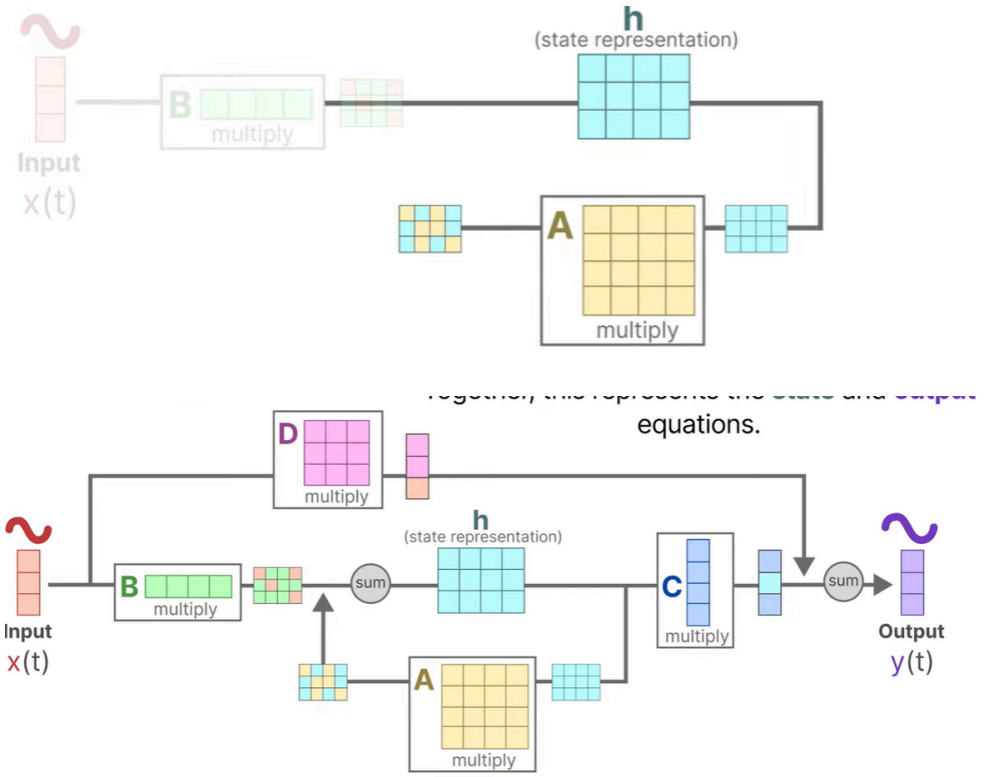
Sabendo que h permite determinar a próxima ação y, a "output equation" é definida por:

$$y'(t) = Ch(t) + Dx(t)$$



É como se a equação dissesse: “Descreve como o state (já atualizado) é transformado no output através de uma matriz fixada C, e como o input influencia diretamente o output, através de uma matriz fixada D.

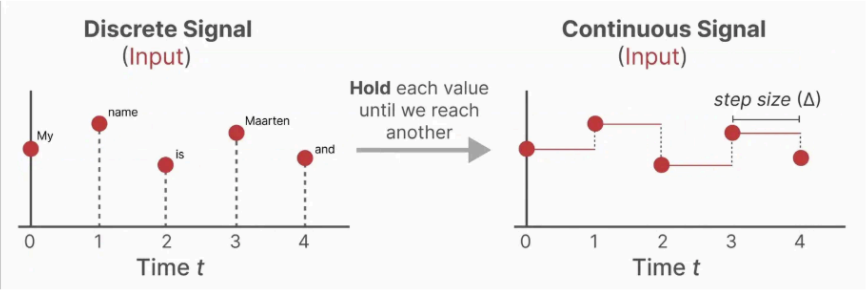
Esse state space model é linear e invariante no tempo. É linear porque as relações nas expressões anteriores são lineares, e é invariante no tempo porque as matrizes de parâmetros A, B, C e D não dependem do tempo, ou seja, permanecem fixas. Para encontrar o sinal de saída y(t), é preciso primeiro determinar uma função h(t) que descreva o estado do sistema em todos os instantes de tempo. Assim, dado um determinado state e uma entrada observada, temos um sistema de natureza autoregressiva que define a próxima ação. Para simplificar, o termo D, que atua como uma skip-connection, pode ser modelado separadamente. Assim, o state space model pode ser pensado apenas em termos de A, B e C.



Uso prático

Para permitir o uso prático do modelo, considerando que é um modelo pensado para function to function, ou seja, completamente contínuo:

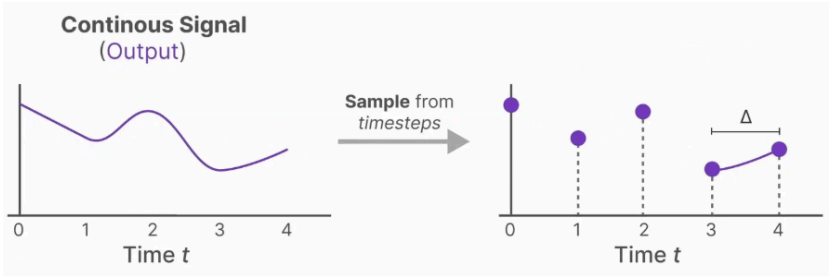
1 - Em modelagem de sequência, considerando cada token como um sinal discreto no tempo, a ideia é segurar o valor do tempo “t” até o tempo “t+1”, criando um sinal contínuo artificial. O espaço entre dois tokens que foi preenchido é chamado de step size (Δ), e é um parâmetro fixado (aprendido).



2 - Esse delta é usado para discretizar as matrizes A e B. Matematicamente:

$$\bar{A} = exp(\Delta A)$$
$$\bar{B} = (\Delta A)^{-1}(exp(\Delta A) - I) \cdot (\Delta B)$$

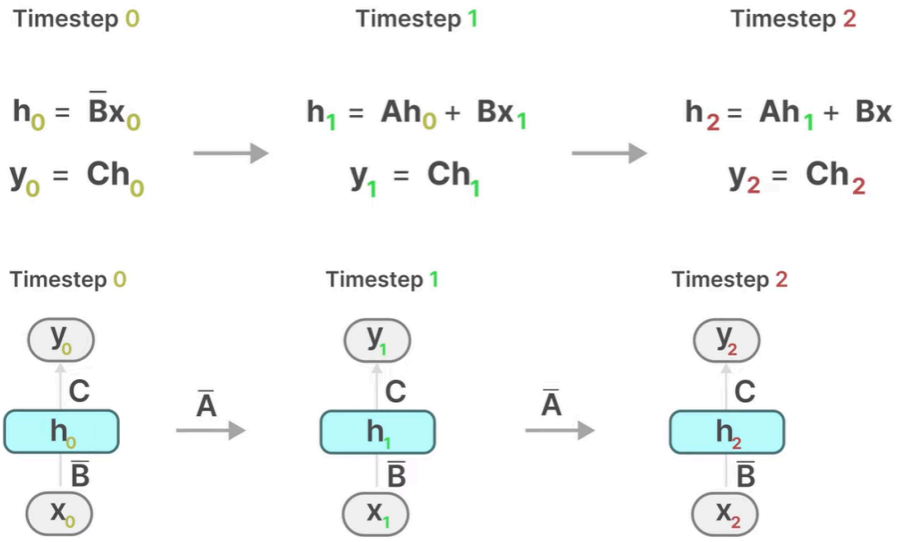
3 - Para transformar o sinal de saída contínuo em discreto, basta reutilizar o step size para “quantizar”.



Assim, transformarmos um modelo function to function para um sequence to sequence.

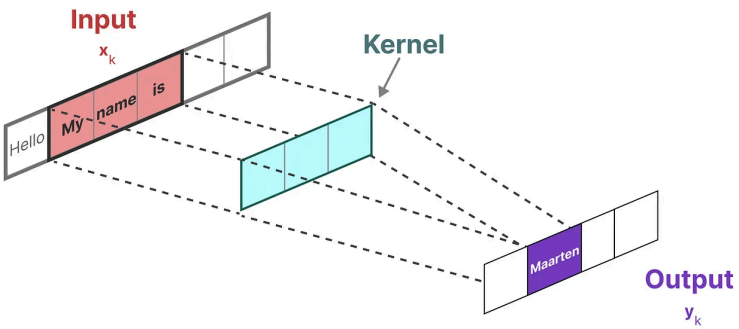
$$h_t = \bar{A}h_{t-1} + \bar{B}x_t$$
$$y_t = Ch_t$$

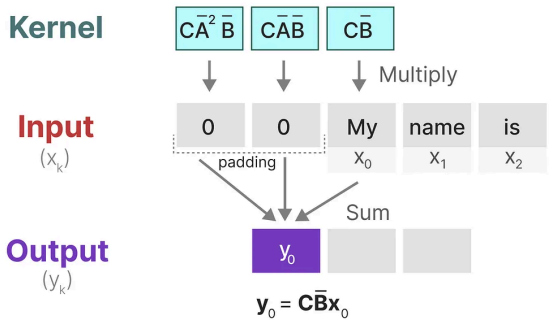
Com isso, podemos enxergar facilmente como uma RNN (linear, ou seja, sem funções não lineares). Até aqui, foi usado o método Zero-Order Hold para a adaptação.



A parte boa de representar uma SSM como uma RNN, é que a inferência é rápida (escala linearmente com o tamanho da sequência de entrada), mas tem um treinamento lento e pouco paralelizável. Mesmo assim, não há o problema de treino dificultado pelo vanishing gradient, visto que não há funções não lineares.

Visto que estamos visualizando as RNNs como um sistema linear, é possível reescrevê-las como uma convolução.





Abrindo e manipulando a “recorrência” da RNN:

$h_t = \bar{A}h_{t-1} + \bar{B}x_t$

Expandindo a recorrencia:

$h_t = \bar{A}(\bar{A}h_{t-2} + \bar{B}x_{t-1}) + \bar{B}x_t$

$h_t = \bar{A}^2h_{t-2} + \bar{A}\bar{B}x_{t-1} + \bar{B}x_t$

$h_t = \bar{A}^3h_{t-3} + \bar{A}^2\bar{B}x_{t-2} + \bar{A}\bar{B}x_{t-1} + \bar{B}x_t$

$\ldots$

$h_t = \bar{A}^t h_0 + \sum_{k=0}^{t-1} \bar{A}^k \bar{B}x_{t-k}$

Substituindo em $y_t = Ch_t$:

$y_t = C\bar{A}^t h_0 + \sum_{k=0}^{t-1} C\bar{A}^k \bar{B}x_{t-k}$

Assumindo $h_0 = 0,$ o termo inicial desaparece:

$y_t = \sum_{k=0}^{t-1} C\bar{A}^k \bar{B}x_{t-k}$

Definindo o kernel (resposta ao impulso): $\bar{K}_k = C\bar{A}^k\bar{B}$

$\Rightarrow; y_t = \sum_{k=0}^{t-1} \bar{K}_k, x_{t-k}$

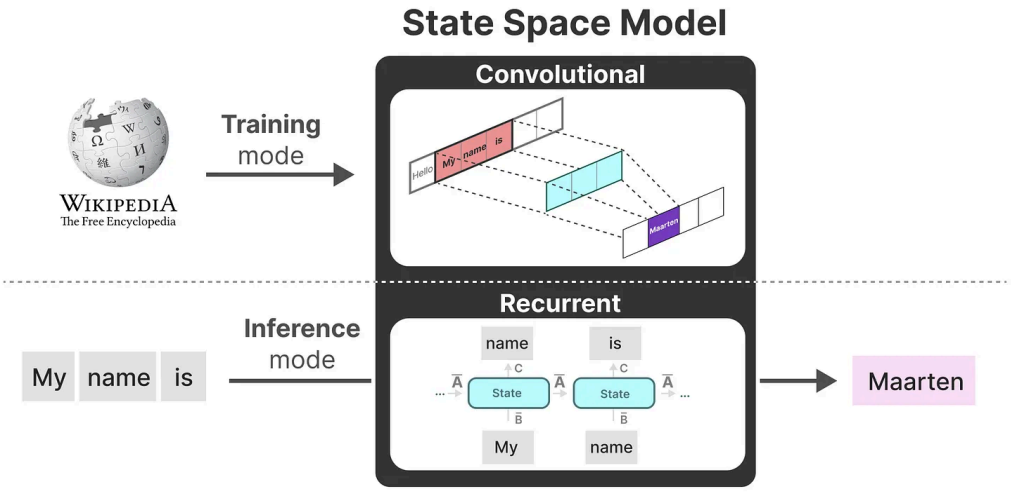
Defini##o geral da convolucaoo discreta: $(f * g) * t = \sum *k = 0^t f_k, g_{t-k}$

$\therefore; y_t = (\bar{K} * x)_t$

$\boxed{y = \bar{K} * x}$

onde $\bar{K} = [C\bar{B}, ; C\bar{A}\bar{B}, ; C\bar{A}^2\bar{B}, ; \ldots]$

Dessas três representações, temos uma combinação ideal possível: a eficiência de treinamento das forma convolucional, juntamente com a eficiência de inferência da forma recorrente.



A matriz (A) é o núcleo do SSM, pois governa a evolução do estado oculto (hidden state) e, portanto, toda a dinâmica temporal do modelo. Inicializá-la a partir da teoria HiPPO (High-order Polynomial Projection Operators) e permitir que ela continue aprendível dentro dessa estrutura mostrou-se altamente eficaz: o modelo parte de um operador de memória contínua - capaz de representar o histórico do sinal de forma comprimida e estável, e aprende apenas pequenas deformações estáveis dessa base. O HiPPO, por sua natureza, privilegia informações mais recentes com maior fidelidade e deixa que o conteúdo mais antigo se degrade gradualmente, o que fornece uma forma matemática de memória com decaimento controlado, garantindo estabilidade e eficiência no aprendizado de longos contextos.

Portanto, combinando tudo isso: State Space Model, HiPPO, RNN na inferência, CNN no treinamento, temos o *Structured State Space for Sequences* (S4).

Limitações

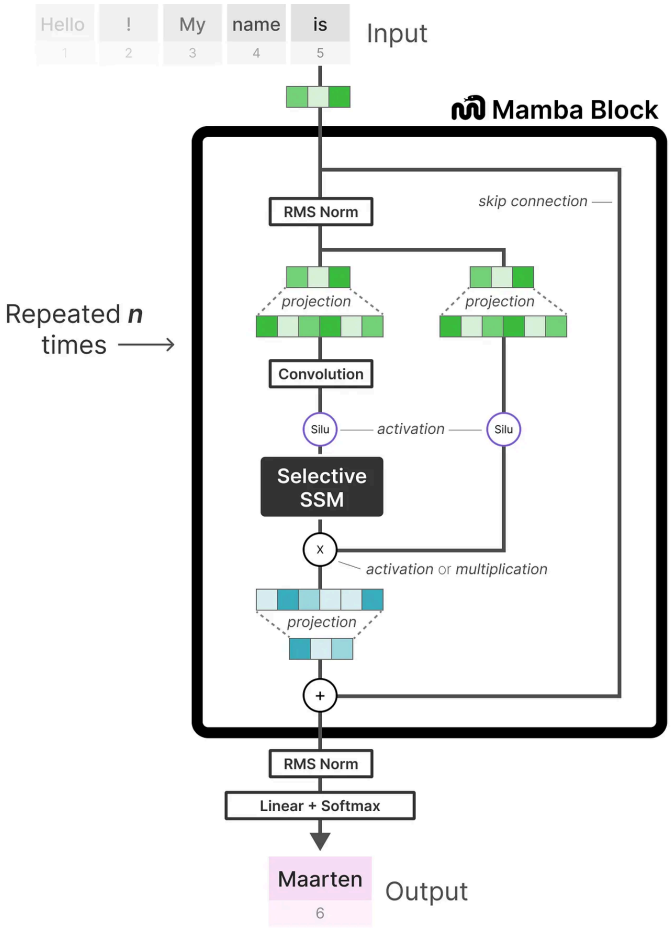
Os state space models ainda apresentam desempenho limitado em Language Modeling, algo que ficou evidente em tarefas simples de reconstrução de sequência, nas quais o modelo precisa receber uma sequência de entrada e retornar a mesma sequência, mas filtrando e ordenando apenas certos tipos de palavras (por exemplo, capturar apenas os substantivos). Esse tipo de tarefa expõe uma limitação estrutural: como as matrizes são fixas, ao contrário dos transformers, os SSMs não conseguem ajustar dinamicamente sua

“atenção” com base no conteúdo do input. Em outras palavras, para diferentes entradas, as matrizes são as mesmas, o que restringe a capacidade do modelo de adaptar seu comportamento ao significado dos tokens.

Essa limitação está diretamente relacionada à ausência do que se chama de "content-aware reasoning". Esse conceito descreve a capacidade de um modelo processar uma sequência de forma que a interação entre dois elementos dependa do próprio conteúdo desses elementos, ou seja, de seu significado ou valor. Os transformers são um exemplo claro de modelos que possuem essa propriedade. O mecanismo de auto-atenção calcula, para cada token, um score de relevância em relação a todos os outros tokens, e esse score depende diretamente do conteúdo. Assim, em uma frase como "O banco financiou a startup", o modelo associa “banco” e “financiou” por estarem semanticamente relacionados, enquanto em "Ele sentou no banco do rio", essas conexões são completamente diferentes. A forma como o modelo processa a sequência, portanto, muda dinamicamente conforme o conteúdo da entrada.

Os SSMs, por outro lado, não possuem essa capacidade até o momento. O S4, por exemplo, é sensível apenas à posição ou ao tempo, e não ao conteúdo. Sua arquitetura se baseia em um filtro de convolução global, que atua de maneira fixa e independente dos dados. Durante o treinamento, o modelo aprende um filtro convolucional K a partir das matrizes A, B e C, mas, uma vez treinado, esse filtro se torna estático: ele é o mesmo para qualquer sequência de entrada. A convolução, por definição, combina informações com base em posição relativa, não em significado. Assim, o filtro K aprende padrões posicionais (como “o que vem 5 passos antes é importante de tal maneira”), mas aplica essa mesma regra em toda a sequência, independentemente do conteúdo dos tokens. Isso faz com que o modelo seja “cego” ao conteúdo e não consiga recalcular dinamicamente as relações entre tokens conforme seus significados, diferentemente dos transformers, cujo mecanismo de atenção é intrinsecamente dependente do conteúdo.

Arquitetura Mamba (S6)



Dois componentes principais: Selective Scan Algorithm para filtrar informações (irrelevantes e relevantes), e o hardware-aware algorithm para armazenar os resultados eficientemente.

O objetivo é obter mais inteligência por FLOPs e dados na modelagem, obtendo mais capacidades de forma mais eficiente.

Compressão

A RNN canônica utiliza uma compressão extremamente forte de state, tendo um vetor de tamanho fixo para comprimir todo o passado, o que permite maior eficiência (treinamento linear, inferência em tempo constante), mas acarreta em performance ruim em tarefas com

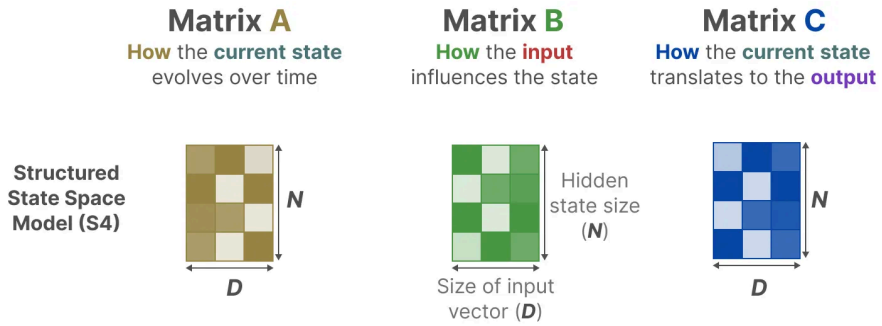
informações densa, como o próprio Language Modeling. Os Transformers, por sua vez, em troca de baixa eficiência, não comprimem o passado, fazendo com que o state seja todos os embeddings dos vetores anteriores, mas possuem uma performance muito superior em tarefas complexas.

1. Premissa do State size: o quão largo o state do modelo autoregressivo deve ser, e quanta informação ele consegue reter? Ao invés de usar um vetor como state, usa uma matriz N-dimensional.
2. Premissa do State update: como manter o controle granular do que é retido no state, e o que é esquecido? Mecanismo de seletividade/gating: input-dependent transitions.
3. Premissa da Eficiência: como manter a eficiência conforme a sequência e o modelo escala? Associative Scan: recorrência linear paralelizável e hardware aware algorithmts.

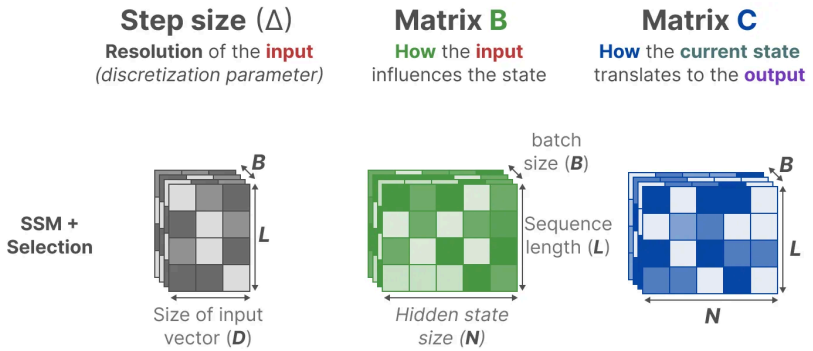
Selective Scan Algorithm

No S4, as matrizes B, C e o step size são independentes da entrada, sendo fixadas com dimensões Hidden State Size (N) x Input Vector (D). No Mamba, essas matrizes passam a depender do input, tornando-se dinâmicas, com dimensões Sequence Length (L) x Hidden State Size (N) x Batch Size (B). Isso significa que, em vez de manter os mesmos parâmetros para toda a sequência, o modelo ajusta suas matrizes conforme o conteúdo e o momento da entrada, tornando-se sensível ao contexto.

Com essas matrizes dinâmicas, as operações não podem mais ser calculadas como uma convolução tradicional. Assim, o Mamba adota uma forma recorrente similar a uma RNN, onde cada estado é a soma do estado anterior (multiplicado por A) mais a entrada atual (multiplicada por B). Essa operação é chamada de scan e normalmente seria executada de forma sequencial, já que cada estado depende do anterior.
Antes:

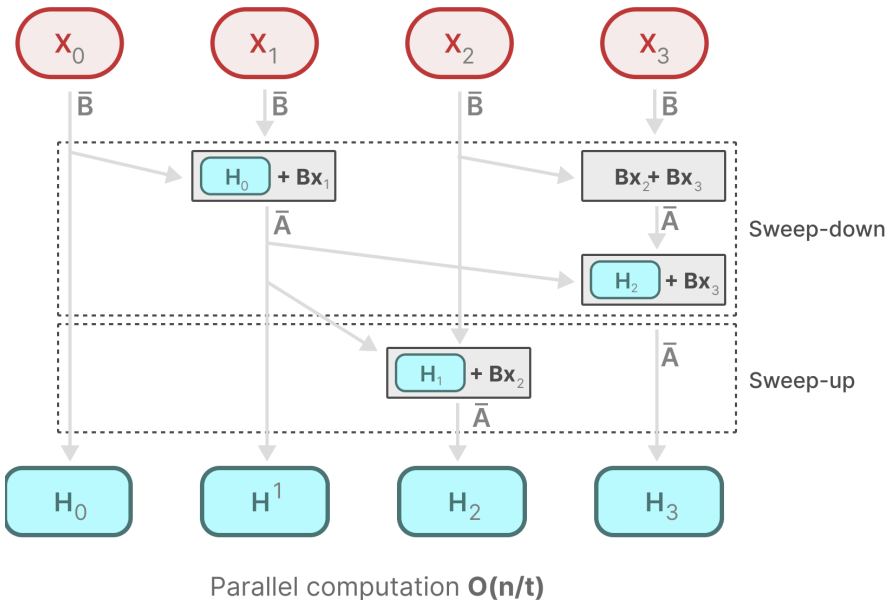


Depois:



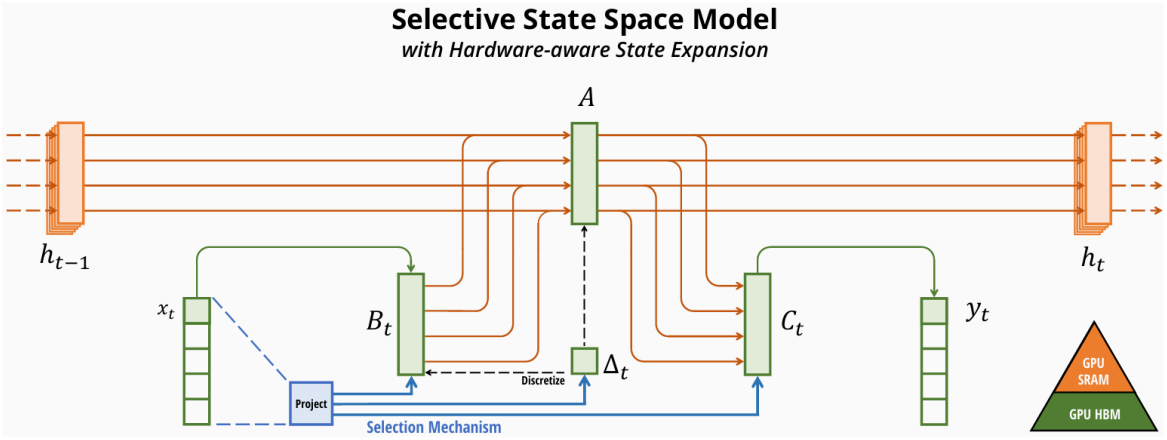
A inovação do Mamba está em tornar essa operação paralelizável por meio do *Parallel Scan Algorithm*. A ideia é assumir que a ordem das operações não altera o resultado, explorando a propriedade associativa. Dessa forma, a sequência pode ser dividida em partes menores, as recorrências internas de cada parte são calculadas em paralelo e, por fim, os resultados são combinados de maneira eficiente.

A combinação das matrizes dinâmicas B e C com o parallel scan resulta no *Selective Scan Algorithm*, que permite ao Mamba representar o comportamento recorrente de forma dinâmica e rápida. Essencialmente, a sequência é fragmentada em blocos, as dependências locais são resolvidas em paralelo dentro de cada bloco e os resultados são fundidos, garantindo desempenho linear e alta eficiência no treinamento e na inferência.



Parallel computation **O(n/t)**

Hardware-aware algorithm



- GPU High Bandwidth Memory (HBM) and - memória maior, mas mais lenta, enquanto GPU Static Random-Access Memory (SRAM) é uma memória bem menor, mas bem mais rápida.
- Ideia principal: embora o hidden state seja grande, não é necessário armazená-lo na memória HBM - deve ser armazenado na SRAM, para cada layer. Mecanismo parecido com Flash Attention.

Referências

[1] REN, Liliang *et al.* Decoder-Hybrid-Decoder Architecture for Efficient Reasoning with Long Generation. arXiv, , 16 jul. 2025. Disponível em: <http://arxiv.org/abs/2507.06607>. Acesso em: 9 out. 2025

[2] MICROSOFT *et al.* Phi-4-Mini Technical Report: Compact yet Powerful Multimodal Language Models via Mixture-of-LoRAs. arXiv, , 7 mar. 2025. Disponível em: <http://arxiv.org/abs/2503.01743>. Acesso em: 9 out. 2025

[3] Intuition behind Mamba and State Space Models | Enhancing LLMs! , 20 mar. 2025. Disponível em: <https://www.youtube.com/watch?v=BDTVVlUU1Ck>. Acesso em: 9 out. 2025

[4] Mamba architecture intuition | Shawn's ML Notes. , 4 set. 2024. Disponível em: <https://www.youtube.com/watch?v=LefUVgPbnNg>. Acesso em: 9 out. 2025

[5] Mamba Explained. Disponível em: <https://thegradient.pub/mamba-explained/>. Acesso em: 9 out. 2025.

[6] GROOTENDORST, Maarten. A Visual Guide to Mamba and State Space Models. Disponível em: <https://newsletter.maartengrootendorst.com/p/a-visual-guide-to-mamba-and-state>. Acesso em: 9 out. 2025.

[7] IITM, AI Club. Zero to Mamba: An intuitive explanation to the Mamba Architecture. Medium, 6 jan. 2025. Disponível em: <https://medium.com/@aiclub.iitm/zero-to-mamba-an-intuitive-explanation-to-the-mamba-architecture-d52265b771ab>. Acesso em: 9 out. 2025

[8] GU, Albert; DAO, Tri. Mamba: Linear-Time Sequence Modeling with Selective State Spaces. arXiv, , 31 maio 2024. Disponível em: <http://arxiv.org/abs/2312.00752>. Acesso em: 9 out. 2025

[9] GU, Albert *et al.* Combining Recurrent, Convolutional, and Continuous-time Models with Linear State Space Layers. *In*: Curran Associates, Inc., 2021. Disponível em: <https://proceedings.neurips.cc/paper_files/paper/2021/hash/05546b0e38ab9175cd905eebcc6ebb76-Abstract.html>. Acesso em: 9 out. 2025

[10] GU, Albert; GOEL, Karan; RÉ, Christopher. Efficiently Modeling Long Sequences with Structured State Spaces. arXiv, , 5 ago. 2022. Disponível em: <http://arxiv.org/abs/2111.00396>. Acesso em: 9 out. 2025

[11] Mamba: Linear-Time Sequence Modeling with Selective State Spaces (COLM Oral 2024). , 16 out. 2024. Disponível em: <https://www.youtube.com/watch?v=X-7rgesJaGM>. Acesso em: 12 out. 2025

[12] Mamba: The Hard Way. Disponível em: <https://srush.github.io/annotated-mamba/hard.html>. Acesso em: 15 out. 2025.

[13] SUN, Yutao *et al.* You Only Cache Once: Decoder-Decoder Architectures for Language Models. arXiv, , 9 maio 2024. Disponível em: <http://arxiv.org/abs/2405.05254>. Acesso em: 9 out. 2025

[14] HUA, Weizhe *et al.* Transformer Quality in Linear Time. arXiv, , 27 jun. 2022. Disponível em: <http://arxiv.org/abs/2202.10447>. Acesso em: 9 out. 2025

[15] YANG, Songlin *et al.* Gated Linear Attention Transformers with Hardware-Efficient Training. arXiv, , 27 ago. 2024. Disponível em: <http://arxiv.org/abs/2312.06635>. Acesso em: 9 out. 2025

[16] REN, Liliang *et al.* Samba: Simple Hybrid State Space Models for Efficient Unlimited Context Language Modeling. arXiv, , 28 fev. 2025. Disponível em: <http://arxiv.org/abs/2406.07522>. Acesso em: 9 out. 2025

[17] YANG, Songlin; KAUTZ, Jan; HATAMIZADEH, Ali. Gated Delta Networks: Improving Mamba2 with Delta Rule. arXiv, , 6 mar. 2025. Disponível em: <http://arxiv.org/abs/2412.06464>. Acesso em: 9 out. 2025

[18] ELFWING, Stefan; UCHIBE, Eiji; DOYA, Kenji. Sigmoid-Weighted Linear Units for Neural Network Function Approximation in Reinforcement Learning. arXiv, , 2 nov. 2017. Disponível em: <http://arxiv.org/abs/1702.03118>. Acesso em: 9 out. 2025

APÊNDICE 4

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]:**

Gustavo Luiz Bueno Pereira

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

Tema central: " 'Large Language Models' e suas Otimizações Arquiteturais"
Enfoque: "Arquiteturas Híbridas baseadas em State Space Models e Transformers".
Nas primeiras semanas da Residência, foi realizado um estudo evolutivo dos modelos de Language Modeling, desde suas origens até os Transformers, base das últimas gerações de LLMs. A partir daí, analisaram-se suas variantes e otimizações arquiteturais ao longo do tempo, e ficou evidente que existem determinadas limitações/fraquezas conhecidas da arquitetura original, e cada otimização é uma tentativa de mitigá-las.
Durante o estudo, chamou atenção o modelo Phi-4-mini-flash-reasoning (Microsoft, 2025), por ser recente e adotar uma arquitetura híbrida que alterna blocos de Transformers e Mamba. A leitura do paper motivou o aprofundamento no funcionamento dos State Space Models e na evolução dessa abordagem, fundamental para compreender os modelos híbridos contemporâneos.

Durante a sétima Semana da Residência:

- Houve a implementação do Mamba em Pytorch como objeto de estudo central, onde:
  - Iniciei rascunhando em desenho o que havia entendido da arquitetura a partir do Paper, já destacando dimensionalidades;
  - Realizei uma leitura do Código Original, sem se prender ao que não havia entendido, apenas para entender a disposição do código. Percebi que, enquanto o Paper tenta passar o essencial, as motivações e intuições para entendimento da arquitetura, o código foca em otimizar ao máximo todos os módulos, tornando-se bem menos intuitivo, e de difícil reprodução em pouco tempo (havia código em CUDA, por exemplo);
  - Por conta disso, busquei tutoriais centrados na implementação essencial do modelo, ainda com algum nível de fidelidade à implementação original, mas sem se importar tanto com a otimização. Nisso, encontrei tutoriais com justamente esse objetivo, mas que acabaram se distanciando mais do que gostaria do original, então me baseei principalmente em um que manteve um bom equilíbrio: Tutorial (2023, desenvolvido pelo Rudy Pei, PhD, Engenheiro Sênior de Deep Learning da NVIDIA). Enquanto isso, mantive o original aberto, checando frequentemente como determinado módulo era realmente implementado;
  - Após rascunhar uma estrutura de código no Google Colab, comecei a acompanhar linha por linha o tutorial citado, tirando todas as dúvidas antes de reescrevê-la, não deixando de escrever textos e comentários das dúvidas tiradas, além de manter o meu próprio estilo em determinados trechos.
  - Seguindo essa metodologia, considerei a implementação como uma atividade que me

ajudou muito a compreender melhor determinadas partes da arquitetura Mamba, além de ter servido como revisão de determinados conceitos de Deep Learning e do próprio Pytorch.
    - Implementação: implementacao_mamba.ipynb
- Além disso, também consegui preparar o ambiente corretamente para fazer inferência com o modelo “Phi-4-mini-flash-reasoning”. Documentação: Inferencia Phi-4-mini-flash

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

Para a próxima Semana, planeja-se:
- Iniciar o estudo e implementação de como fazer um fine-tuning de uma arquitetura híbrida, utilizando somente dados em português.

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

ACEITE DA ENTREGA:

CEDRIC LUIZ DE CARVALHO: Go!

Documento de Apoio - Semana 7

No link “ implementacao_mamba.ipynb ”, temos a implementação simplificada do Mamba, realizada nesta Semana. Neste Colab, a primeira parte é referente a anotações realizadas informalmente ao longo do entendimento alcançado durante a codificação. Em seguida, há a implementação comentada do módulo “State Space Model” e do “CausalConv1d” utilizados no “MambaBlock”, os quais são instanciados na arquitetura final, “Mamba”.

No link “ Inferencia Phi-4-mini-flash ”, estão detalhes de criação do ambiente de inferência, erros corrigidos, e resultados retornados.

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: 22 de out. de 2025

Participantes da Entrega [matriculados em Residência em IA]:

GUSTAVO LUIZ BUENO PEREIRA

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

Tema central: " 'Large Language Models' e suas Otimizações Arquiteturais"
Enfoque: "Arquiteturas Híbridas baseadas em State Space Models e Transformers".

Nas primeiras Semanas, iniciamos um estudo evolutivo dos modelos de Language Modeling, até os Transformers; em seguida estudamos suas variantes e otimizações arquiteturais ao longo do tempo. Durante o estudo, chamou atenção o modelo híbrido Phi-4-mini-flash-reasoning, onde a leitura do Paper motivou o aprofundamento no funcionamento dos State Space Models e na evolução dessa abordagem até o Mamba, o qual foi implementado na Semana anterior, visando melhor compreensão de um dos fundamentos das arquiteturas híbridas.

Durante a oitava Semana:

- Decidi que prosseguiria com uma tentativa de comparação entre o pré-treino de uma arquitetura híbrida, e o de uma arquitetura transformers comum, ao invés de um fine-tuning.
- De certa forma, o Paper do Phi-4-mini-flash-reasoning (Decoder-Hybrid-Decoder Architecture for Efficient Reasoning with Long Generation, 07/2025) realiza experimentos como esse, comparando a arquitetura proposta com outras "baselines", e em diferentes etapas de treinamento.
- Uma diferença é que a minha proposta é compará-los somente na etapa de pré-treino, olhando diretamente para as métricas de "loss" e até de "perplexidade", e verificar se a arquitetura híbrida apresenta uma convergência mais rápida, ou seja, melhor "métrica", em menos "tempo", que é uma das constatações do Paper.
- Outra diferença crucial, é que não tenho milhares de GPUs como a Microsoft, logo, será uma comparação em uma escala ainda menor do que qualquer uma proposta no Paper: Arquitetura Híbrida (SambaY) vs Transformers++, ambos próximos de 110M de parâmetros. Para se ter uma ideia, para realizar esse experimento que constatou a convergência mais rápida dos modelos híbridos, no Paper, fixaram diversos modelos em 1B de parâmetros, e treinaram em escalas de 100B até 600B de tokens.
- Baseando-se nas "Chinchilla Scaling Laws" (Training Compute-Optimal Large Language Models, 03/2022), seguirei a ideia de manter uma proporção de número de tokens de treinamento "N" vezes maior do que o número de parâmetros do modelo. Neste Paper, uma das proporções utilizadas, e com excelente resultado, foi a de "20 tokens para cada 1 parâmetro". Nessa ideia, minha meta de treinamento seria a utilização de algo próximo de 2.2B de tokens, mas, a princípio, iniciarei somente com aproximadamente 1.1B de tokens, ou seja, metade da escala utilizada no Paper, no intuito de começar ainda menor para verificar se o treinamento irá pelo menos começar a rodar

com sucesso. O Dataset será composto por amostras do mesmo dataset utilizado no teste de escala do Paper, “Slimpajama”. Por ser considerado de alta qualidade, está alinhado à premissa da família Phi, em que poucos dados de alta qualidade são mais valiosos do que muitos de baixa qualidade, permitindo uma experimentação mais eficiente.

- Atualmente, o hardware que tenho acesso para o treinamento é uma GPU RTX 4090, com 24GB VRAM. Se minhas estimativas estiverem corretas, considerando as condições listadas nos tópicos acima, ela será suficiente para rodar o experimento.
- Pesquisei diversos frameworks para realizar o pré-treino de uma arquitetura híbrida, e o “ArchScale”, o mais recente da Microsoft, pareceu o mais adequado. Então o defini como framework de treinamento, e iniciei a preparação de dados e ambiente:
    - Foram encontrados uma infinidade de problemas para a execução do framework, e as principais estavam ligadas a dependências, visto que ele não possui uma boa documentação, e o Dockerfile disponibilizado não estava funcionando por estar desatualizado. Ou seja, a adaptação do ambiente realizada na Semana anterior para inferência foi somente a “ponta do iceberg”, se comparada à preparação para pré-treino.
    - Por conta dos problemas, realizei um fork do framework, onde comecei a modificar o Dockerfile, a atualizar alguns imports, e adicionei um script para baixar somente algumas amostras do dataset proposto, visto que era inviável baixar 627B de tokens para só então definir a porcentagem desejada para treinamento. O fork pode ser encontrado em: https://github.com/Gustavoaluiz/ArchScale.
    - O ambiente (Kubeflow) em que a GPU RTX 4090 está disponibilizada, por si só, já é uma imagem Docker, então não era possível executar um outro container dentro dela, o que dificultou muito o prosseguimento, visto que a imagem base era essencial. Por isso, gastei um tempo considerável criando uma imagem customizada para disponibilizar a GPU, a qual pode ser encontrada em: gustavoluizbp/pytorch-kf - Docker Image | Docker Hub.
    - Apesar de tudo isso, não consegui iniciar o treinamento, pois ainda estou enfrentando diversos problemas de dependências e código desatualizado.

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

Para a nona Semana da Residência, planeja-se:

- Resolver os problema de ambiente;
- Finalizar preparação dos dados;
- Realizar o treinamento da arquitetura híbrida SambaY, e da arquitetura Transformers++, nas condições listadas durante esta Semana.

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: 6 de nov. de 2025

Participantes da Entrega [matriculados em Residência em IA]:

GUSTAVO LUIZ BUENO PEREIRA

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

Tema central: " 'Large Language Models' e suas Otimizações Arquiteturais"
Enfoque: "Arquiteturas Híbridas baseadas em State Space Models e Transformers".

Nas primeiras Semanas, iniciamos um estudo evolutivo dos modelos de Language Modeling, até os Transformers; em seguida estudamos suas variantes e otimizações arquiteturais ao longo do tempo. Durante o estudo, chamou atenção o modelo híbrido Phi-4-mini-flash-reasoning, onde a leitura do Paper motivou o aprofundamento no funcionamento dos State Space Models e na evolução dessa abordagem até o Mamba, o qual foi implementado para melhor compreensão.

Na Semana anterior, foi iniciada a comparação entre a arquitetura híbrida SambaY, proposta no paper citado anteriormente, e arquiteturas Transformers convencionais, por meio do pré-treinamento em menor escala de ambas. O objetivo foi reproduzir, em menor proporção, o resultado apresentado no Paper: arquiteturas híbridas tendem a convergir de forma mais eficiente do que arquiteturas Transformers convencionais. Em outras palavras, esperava-se que as métricas de loss e perplexity, tanto de treino quanto de validação, fossem menores para a SambaY na mesma quantidade de Steps. Além disso, iniciou-se o fork do repositório original da Microsoft utilizado no estudo, com o intuito de atualizá-lo e adaptá-lo ao ambiente local, utilizando uma GPU RTX 4090 com 24 GB de VRAM.

Portanto, durante a nona Semana:

- Decidiu-se pela comparação do SambaY de ~150M com o TransformerLs (arquitetura parecida com Llama 2, mas com Sliding Window Attention) de ~150M de parâmetros. Uma comparação justa de arquiteturas que possuem blocos diferentes não é fácil, mas o Paper tratou disso, e disponibilizou no repositório - na prática, apenas escolhi que seria utilizado a versão de profundidade "8" de ambas arquiteturas, e automaticamente fizeram o "resize" de partes diferentes da arquitetura para que torne justa a comparação.
- Seriam comparadas duas versões diferentes de cada uma:
  - Janela de contexto de 4096 tokens, e batch size efetivo de ~1M de tokens;
  - Janela de contexto de 2048 tokens, e batch size efetivo de ~2M de tokens.
- Finalizou-se a preparação dos dados:
  - 2.2B de tokens para treinamento e 2M de validação, do dataset de alta qualidade SlimPajama.
- A primeira "semana" ficou praticamente reservada para ajuste do ambiente de execução. Principais modificações:

- Tive que encontrar combinações de dependências que funcionassem no meu ambiente, já que o repositório não deixou documentado. As bibliotecas que encontrei mais dificuldade foram “flash-attn”, “causal-conv1d” e “mamba-ssm”;
- O repositório foi mais pensado para o uso de múltiplas GPUs, então realizei adaptações para minimizar overhead;
- O código utilizava módulos descontinuados do flash-attn para executar os “Rotary Positional Embeddings” e as “Layer Normalizations”, então realizei adaptações para utilizar as versões atualizadas, conforme orientado pelo próprio repositório do flash-attn, e foi uma adaptação que ocasionou muitos erros;
- Ao fim, foram 29 commits e 11 arquivos modificados no Fork.

- Nas comparações entre as arquiteturas SambaY e TransformerLs, considerando respectivamente a combinação de janela de contexto 2048 com batch size de 2M e de janela 4096 com batch size de 1M, observou-se que, após cerca de 7 horas de treinamento em cada um dos quatro modelos, o SambaY apresentou valores de Loss e Perplexity, tanto de treino quanto de validação, menores que os do TransformerLs, para a mesma quantidade de Steps. Assim, ainda que em menor escala, os resultados reproduzem o que foi relatado no paper, indicando que arquiteturas híbridas tendem a convergir mais rapidamente do que arquiteturas Transformer convencionais.
- As comparações e modificações realizadas estão documentadas em mais detalhes: Semana 9 - Pré Treino

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

- Para a próxima Semana, planeja-se:
  - Analisar a viabilidade de realizar um Supervised Fine Tuning (SFT) com a biblioteca “transformers” de algum modelo pequeno híbrido, tal como o Falcon-H1 0.5B. Caso seja viável no tempo restante, será realizado um fine-tuning “instruct” com dados em português, com o objetivo de ficar minimamente melhor do que o modelo utilizado como base.
  - Caso não seja viável, penso em realizar um estudo mais aprofundado da arquitetura SambaY, pois li somente o Paper do Phi4-mini-flash-reasoning, que na verdade junta outras duas arquiteturas: Yoco e Samba, e a inovação está na forma que são combinadas. Portanto, não conheço de forma aprofundada essas duas arquiteturas, e esse estudo seria interessante para justificar com mais propriedade os resultados comparativos encontrados nesta Semana. No entanto, a prioridade é o SFT.

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

ACEITE DA ENTREGA:

CEDRIC LUIZ DE CARVALHO: Go!

Documento de Apoio - Semanas 8 e 9

*Referente ao documento “Semana 9 - Pré Treino”, que engloba os links citados no Termo da Semana 8

Repositório selecionado para o Pré-Treinamento

- ArchScale, Microsoft - microsoft/ArchScale: Simple & Scalable Pretraining for Neural Architecture Research
- Fork realizado para modificar o repositório e viabilizar os experimentos: https://github.com/Gustavoaluiz/ArchScale
    - Foram realizados 29 commits até chegar na versão final, compatível com o meu pré-treinamento. Foi necessário a alteração de 11 arquivos, onde 3 foram criados para facilitar determinados processos.

Principais Modificações

- O módulo de implementação de rotary embeddings foi atualizado porque a biblioteca utilizada anteriormente deixou de estar disponível no repositório do flash-attn. Mesmo realizando git checkout, a instalação não funcionava. Seguindo a recomendação do próprio flash-attn, passou-se a utilizar um módulo interno da biblioteca triton (“from flash_attn.ops.triton.rotary import apply_rotary”). A partir disso, a implementação precisou ser adaptada, gerando vários erros até chegar a uma versão funcional.
- O código de pré-treino não havia sido escrito com foco em reprodutibilidade, pois muitos hiperparâmetros estavam hard-coded. Por isso, foram tornados configuráveis via CLI: total_evals, save_step_interval, log_step_interval, micro_batch_size e num_extrapol. O script de logging também foi aprimorado, pois não estava obedecendo seus próprios parâmetros (imprimia a cada iteração, ignorando o log_step_interval).
- O código de pré-treino também assumia o uso de múltiplas GPUs, então foram aplicadas pequenas otimizações para evitar overhead desnecessário. Foi necessário desativar torch.compile devido a problemas no ambiente de execução, o que pode

ter deixado o treinamento mais lento. Além disso, imports e dependências voltados ao treinamento multi-GPU foram removidos para facilitar o andamento e reduzir fontes de erro.

- O script original de download do dataset baixava o conjunto completo para, só então, selecionar a porcentagem a ser utilizada no treinamento e validação. Como baixar todo o dataset (aproximadamente 1 TB, com 6×27B tokens) era inviável, foi criado um script que baixa apenas N amostras para treino e validação. No total, foram baixados 2.301.699.072 tokens de treinamento e 27.490.880 tokens de validação.
- Foi criado também um script para contar os tokens de cada split, já que a tokenização gera diversos arquivos .bin preparados para o treinamento.
- As bibliotecas utilizadas apresentavam problemas de compatibilidade, exigindo testes de diferentes combinações. No estado original do código, as instalações mais problemáticas, como causal-conv1d, mamba-ssm e flash-attn, levavam até cerca de 15 minutos. Após pesquisa, identifiquei que é possível acessar as releases oficiais em seus repositórios e baixar diretamente versões já compiladas para combinações específicas de PyTorch, CUDA e Python. Assim, o motivo da lentidão era que o pip recompilava as bibliotecas localmente a cada instalação.

Ambiente

Foi utilizado o Kubeflow como ambiente de treinamento, com a utilização de 1 RTX 4090 24GB VRAM. Para que o ambiente funcionasse corretamente, foi necessário a criação de uma imagem personalizada de execução do Kubeflow (uma imagem comum da nvidia não funcionaria sem os pré-requisitos do Kubeflow, visto que o ambiente de acesso à GPU é montado a partir dessa imagem). Ela pode ser encontrada em: https://hub.docker.com/r/gustavoluizbp/pytorch-kf

Dataset

O dataset utilizado foi o SlimPajama, encontrado no link “cerebras/SlimPajama-627B · Datasets at Hugging Face”. Dos 627B de tokens disponíveis, somente 2.2B foram baixados e utilizados.

Detalhes de Implementação

Os Hiperparâmetros utilizados, as arquiteturas detalhadas dos modelos, bem como cada um foi rodado, estão descritos em detalhes no documento “ Semana 9 - Pré Treino ”.

Resultados Finais

Comparações configuradas de tal forma que cada experimento (modelo treinado) durava de 6-8 horas.

Comparação 1

Experimento “SambaY 4096 ctx_len” x Experimento “TransformerLs 4096 ctx_len”, ambos com Batch Size de 1M tokens:

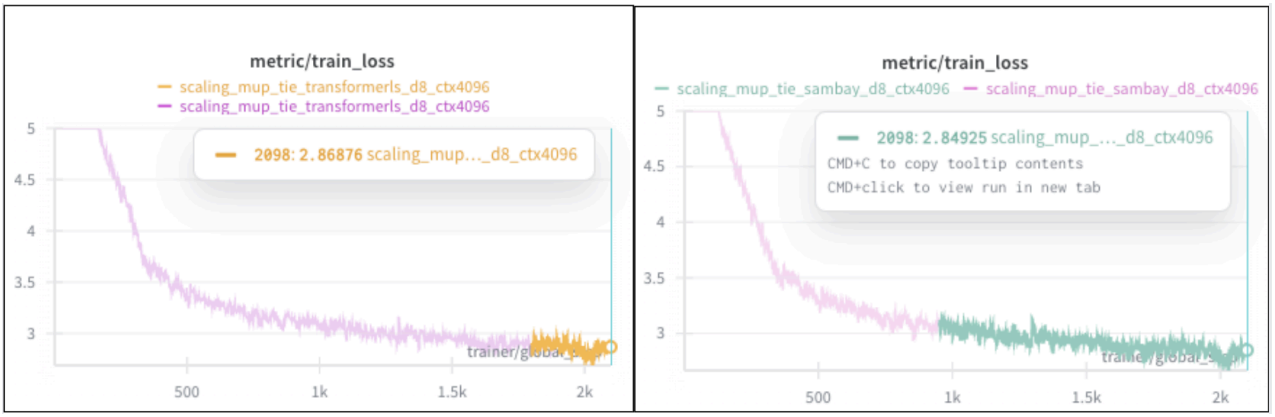


Figura 1: Métrica “Loss de treino” - “SambaY 4096 ctx_len” (À direita) X “TransformerLs 4096 ctx_len” (À esquerda).

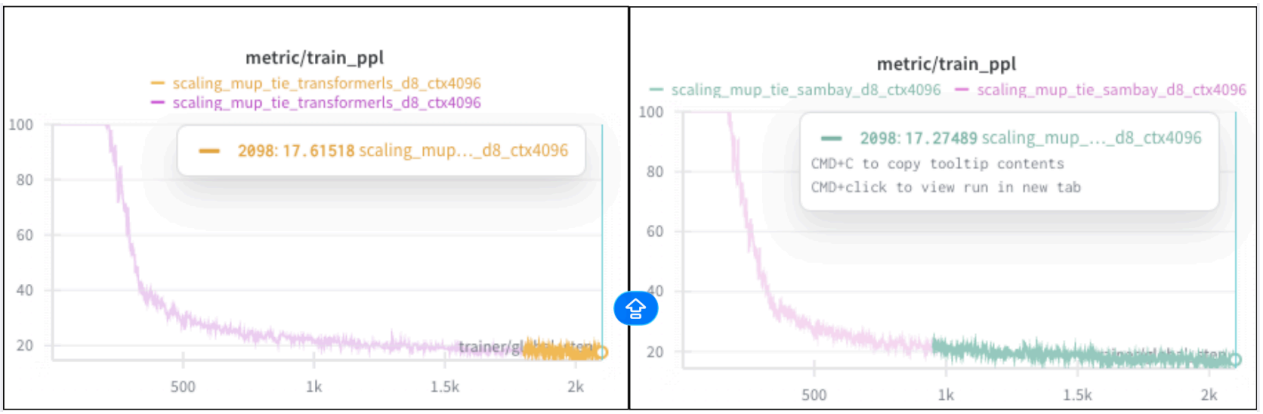


Figura 2: Métrica “Perplexidade de treino” - “SambaY 4096 ctx_len” (À direita) X “TransformerLs 4096 ctx_len” (À esquerda).

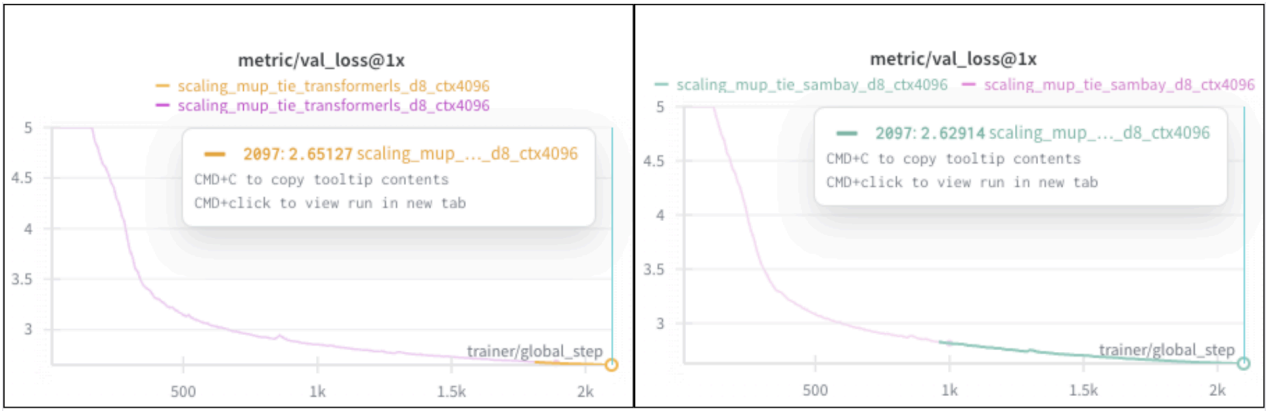


Figura 3: Métrica “Loss de validação” - “SambaY 4096 ctx_len” (À direita) X “TransformerLs 4096 ctx_len” (À esquerda).

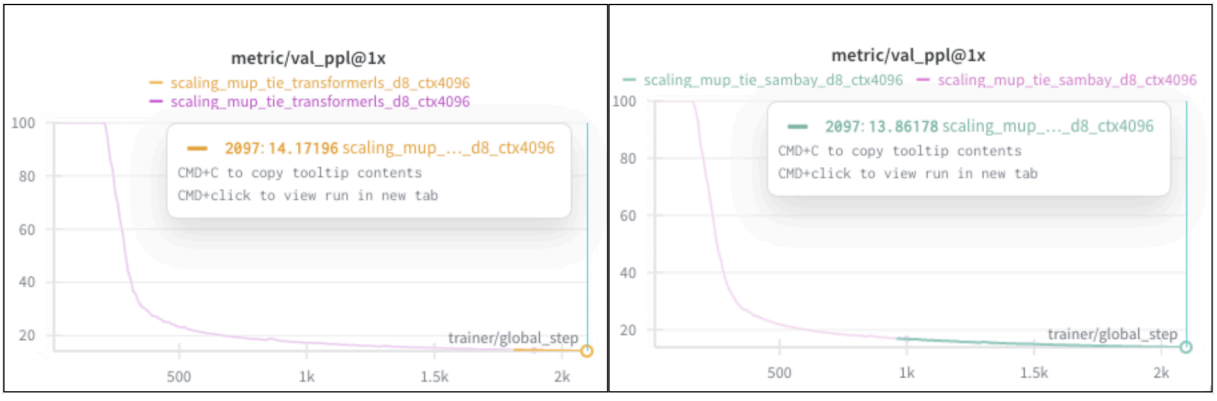


Figura 4: Métrica “Perplexidade de validação” - “SambaY 4096 ctx_len” (À direita) X “TransformerLs 4096 ctx_len” (À esquerda).

Comparação 2

Experimento "SambaY 2048 ctx_len" x Experimento "TransformerLs 2048 ctx_len", ambos com Batch Size de 2M tokens:

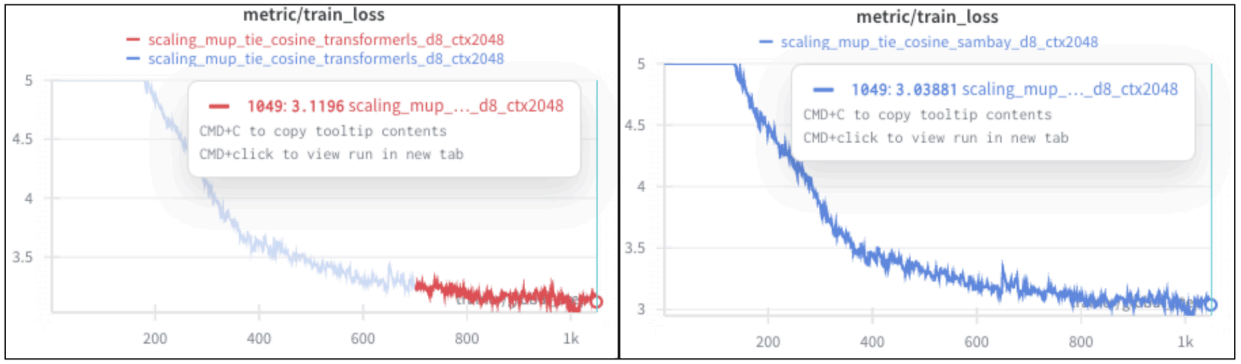


Figura 5: Métrica "Loss de treino". "SambaY 2048 ctx_len" (À direita) X "TransformerLs 2048 ctx_len" (À esquerda).

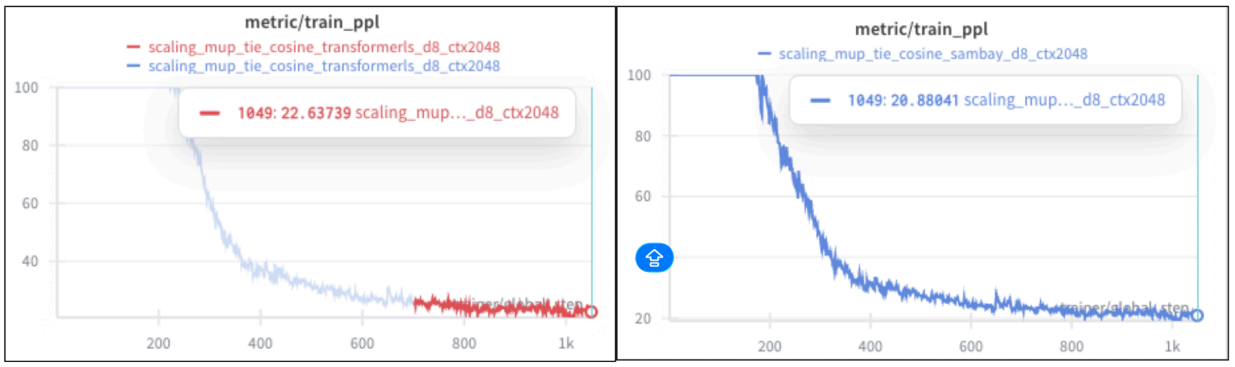


Figura 6: Métrica "Perplexidade de treino". "SambaY 2048 ctx_len" (À direita) X "TransformerLs 2048 ctx_len" (À esquerda).

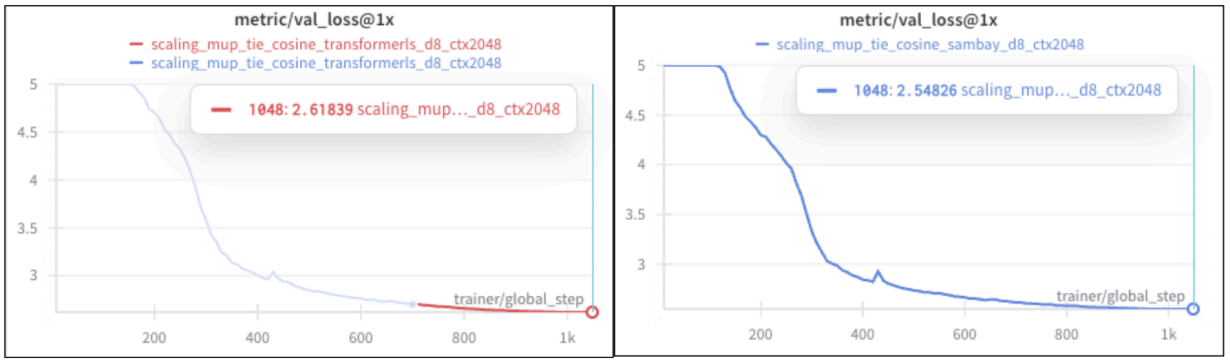


Figura 7: Métrica “Loss de validação”. “SambaY 2048 ctx_len” (À direita) X “TransformerLs 2048 ctx_len” (À esquerda).

Figura 8: Métrica “Perplexidade de validação”. “SambaY 2048 ctx_len” (À direita) X “TransformerLs 2048 ctx_len” (À esquerda).

Conclusão

SambaY obteve loss e perplexidade de treino e validação menores com o mesmo número de steps, o que corrobora, em menor escala, a constatação do paper de que arquiteturas híbridas tendem a convergir de forma mais eficiente do que Transformers convencionais.

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: 12 de nov. de 2025

Participantes da Entrega [matriculados em Residência em IA]:

GUSTAVO LUIZ BUENO PEREIRA

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

Tema central: " 'Large Language Models' e suas Otimizações Arquiteturais"
Enfoque: "Arquiteturas Híbridas baseadas em State Space Models e Transformers".

Durante a décima Semana:

- Foi definido o objetivo de realizar o processo de Supervised Fine-Tuning (SFT) com dados em português de um modelo híbrido, e compará-lo com um modelo Transformers "convencional". Para isso, foi escolhido o modelo híbrido Falcon-H1-0.5B-Base, justamente por disponibilizar versão base dos modelos deles (ou seja, somente pré-treinado, sem SFT ou outros pós-treinos), e também por ter essa versão bem pequena já pré-treinada, de 0.5B parâmetros, tornando os experimentos mais viáveis. Além dele, também foi escolhido o Qwen2.5-0.5B para realizar a comparação, que também é somente pré-treinado, e possui uma arquitetura transformers "convencional".
- Para o treinamento, foi escolhido o framework "trl", com a classe SFTTrainer, além do uso do framework "accelerate" para iniciar o treinamento, e do DeepSpeed somente para permitir o uso da memória RAM para armazenar o estado dos otimizadores, visto que não seria necessária a parte de "treinamento distribuído" dele, considerando o uso de somente uma GPU RTX 4090 - esta configuração desocupa parte da alocação da VRAM da GPU, permitindo batches maiores.
- Foi escolhido o dataset "UltrachatBR", que é um dataset totalmente em pt-br (traduzido do original em inglês) de diálogos multi-turnos, no estilo "human" e "assistant", voltado principalmente para instruct-tuning. O problema é que o dataset não estava no formato mais adequado para o SFT, além de ter apresentado diversas inconsistências de formatação entre amostras, exigindo um tratamento especial para torná-lo compatível.
- Um dos grandes aprendizados esteve no estudo dos hiperparâmetros do SFTTrainer, e como cada um deles influencia o treinamento.
- O problema das dependências de modelos híbridos persistiu, mesmo sendo com o uso de um framework consolidado. Dessa vez, não foi possível resolver esses problemas de ambiente, então durante o treinamento do Falcon-H1, o SFTTrainer automaticamente usou as implementações padrões referentes ao bloco Mamba, ao invés das mais eficientes, o que infelizmente limitou os experimentos, dada a extrema lentidão demonstrada. Por exemplo, a intenção inicial era de utilizar 100K amostras para o SFT, mas a estimativa inicial de conclusão era de mais de 40 horas. Portanto, o treinamento foi limitado para o uso de cerca de 10K amostras, resultando em mais de 4h de processamento. Em contrapartida, o Qwen2.5, que não teve nenhum problema de dependência, finalizou em cerca de somente 30 minutos de treinamento.

- Apesar desses problemas, o Falcon-H1 apresentou um SFT mais estável e, na mesma quantidade de steps, conseguiu se ajustar melhor em comparação com o Qwen, obtendo losses de treino e validação menores, além de ter apresentado uma acurácia média de tokens por amostra maior em ambas as divisões de dataset. Infelizmente não foi possível a realização de benchmarks em domínios específicos.
- Mais detalhes do processo de SFT podem ser encontrados em: Semana 10

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

Como planejamentos futuros, dado o aprofundamento realizado nos processos de Pré-Treino e Supervised Fine-Tuning durante a Residência, pretendo continuar meus estudos, mas na área de “Alinhamento de Modelos”, que normalmente é uma etapa sempre realizado após a SFT nos LLMs atuais.

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

Agradeço ao Fazzioni por ter me indicado um código-base de SFT e o dataset utilizado, além de ter respondido diversas dúvidas sobre o processo de fine-tuning.

ACEITE DA ENTREGA:

CEDRIC LUIZ DE CARVALHO: Go!

Documento de Apoio - Semana 10

*Referente ao documento “ Semana 10 ”

Materiais de Apoio:

- https://github.com/huggingface/gpt-oss-recipes/blob/main/sft.py - Script de referência da Hugging Face para realização de SFT (Supervised Fine-Tuning) em modelos GPT-OSS. Mostra a estrutura básica de carregamento de dados, preparação do modelo e execução do fine-tuning supervisionado.
- https://huggingface.co/tiiuae/Falcon-H1-0.5B-Base - Versão base do modelo Falcon H1 com 0.5B parâmetros, utilizada principalmente como ponto inicial para fine-tuning ou experimentos de pré-treino adicional.
- https://huggingface.co/tiiuae/Falcon-H1-0.5B-Instruct - Versão já com fine-tuning “instruct” do Falcon H1 0.5B, já ajustada para seguir comandos e realizar tarefas conversacionais. É voltada para uso direto ou para servir como baseline em tarefas de instrução ou outras etapas de pós-treino.
- https://huggingface.co/docs/trl/sft_trainer - Documentação oficial do SFTTrainer da biblioteca TRL (Transformers Reinforcement Learning). Explica como utilizar a ferramenta para realizar fine-tuning supervisionado de modelos de linguagem, incluindo parâmetros, exemplos e boas práticas.
- https://wandb.ai/capecape/alpaca_ft/reports/How-to-Fine-tune-an-LLM-Part-3-The-HuggingFace-Trainer–Vmlldzo1OTEyNjMy - Tutorial no Weights & Biases sobre fine-tuning de LLMs usando o HuggingFace Trainer. Apresenta uma visão geral do processo, com códigos de exemplo, boas práticas e análise de resultados.
- https://huggingface.co/Qwen/Qwen2-0.5B - Versão base do modelo Qwen2 com 0.5B parâmetros, utilizada principalmente como ponto inicial para fine-tuning ou experimentos de pré-treino adicional, seguindo o mesmo propósito do Falcon-Base em sua respectiva linha.
- https://huggingface.co/Qwen/Qwen2-0.5B-Instruct - Versão já com fine-tuning “instruct” do Qwen2 0.5B, já ajustada para seguir comandos e realizar tarefas

conversacionais. É voltada para uso direto ou para servir como baseline em tarefas de instrução ou outras etapas de pós-treino.

Objetivo

Realizar o processo de Supervised Fine-Tuning (SFT) com dados em português em um modelo híbrido e compará-lo com um modelo Transformer convencional. Para isso, optei por utilizar o modelo híbrido Falcon-H1-0.5B-Base, justamente por disponibilizar uma versão base (apenas pré-treinada, sem SFT ou outros pós-treinos) e por possuir uma variante pequena, de 0.5B parâmetros, o que torna os experimentos mais viáveis.

Conforme a imagem abaixo, os criadores do Falcon-H1 reportam resultados promissores em relação a outros modelos, todos após SFT. No entanto, o objetivo aqui não é replicar esses resultados, como ocorreu com o pré-treino do SambaY em comparação com o TransformersLs. A proposta é tentar adaptar esses modelos para o português ao mesmo tempo em que buscamos transformá-los em modelos instruct, isto é, capazes de seguir instruções e responder perguntas. Um ponto interessante é que, no paper do Falcon-H1 [2], é explicitamente informado que o modelo de 0.5B parâmetros foi o único pré-treinado exclusivamente com dados em inglês. Ou seja, ele não foi projetado para ser multilíngue, provavelmente devido às limitações impostas pelo seu tamanho.

Figura 1: Relação entre tamanho do modelo (número de parâmetros, em bilhões) e desempenho agregado em tarefas de avaliação [1]. A imagem compara modelos das famílias Falcon-H1, Qwen3, Gemma3, Llama3 e Falcon3. Cada ponto representa o desempenho médio do modelo em categorias como ciência, matemática, tarefas gerais, programação e instruction-following. Curvas mais altas indicam desempenho superior para um mesmo tamanho de modelo.

Dataset Utilizado

https://huggingface.co/datasets/recogna-nlp/UltrachatBR - Dataset brasileiro de conversas multi-turno, usado para treinar e avaliar modelos de linguagem em tarefas de diálogo. Contém interações diversas e é útil para SFT em português.

Problema Identificado

A referência utilizada foi o dataset disponível em: https://huggingface.co/datasets/trl-lib/Capybara/viewer/default/train?row=0&views%5B%5D=train. O ideal seria que os dados estivessem no formato:

[{ “content”: “lorem ipsum dolor”, “role”: “user” },
{ “content”: “lorem ipsum”, “role”: “assistant” }]

Esse formato facilita o uso de hiperparâmetros como “assistant_only_loss”, que permite computar gradientes apenas nos tokens do assistente, mascarando automaticamente os tokens do usuário sem necessidade de configuração manual no carregamento do dataset. No entanto, o dataset está no seguinte formato:

[{ “humano”: “lorem ipsum” }, { “assistente”: “lorem ipsum” }, …]

Além de não estar no formato content–role, as roles foram traduzidas e, mais problemático ainda, o dataset foi salvo de maneira irregular. Ou seja, aparentemente não foi carregado como JSON/dict antes de ser salvo. Foi armazenado como string sem padronização, preservando a tradução literal das tags. Isso causou problemas ao utilizar o dataset no processo de SFT, mas todos foram corrigidos. As correções encontram-se no arquivo “sft.py” no repositório citado na seção “Código”.

Código

- O repositório com o código está em: https://github.com/Gustavoaluiz/falcon-h1-0.5b-sft-ptbr.
- Vale destacar que foi completamente baseado no “https://github.com/huggingface/gpt-oss-recipes/blob/main/sft.py”, citado nos materiais de apoio.

Supervised Fine-Tuning (SFT)

Problema identificado em relação a modelos híbridos

A utilização desse tipo de modelo sempre exige mais tempo para configuração do ambiente, pois as bibliotecas mamba-ssm, flash-attn e causal-conv1d frequentemente entram em conflito entre si e com a versão do torch instalada, conforme já comentado durante o uso do ArchScale nas semanas anteriores. No caso do SFTTrainer, a ausência dessas bibliotecas impede o uso de kernels mais eficientes e resulta em uma lentidão além

do esperado. Quando tentei instalá-las, surgiram novamente inúmeros erros. Esse foi um problema crucial que limitou a possibilidade de experimentos na semana, tendo impacto direto e significativo no treinamento.

Processo

- Comparado à experiência anterior de pré-treino com o ArchScale, foi bem mais tranquilo iniciar o fine-tuning utilizando o framework “trl” e a classe SFTTrainer. A maior parte dos erros esteve relacionada ao dataset, além do problema de dependências já mencionado. Isso permitiu dedicar mais tempo ao estudo dos hiperparâmetros da classe e ao entendimento de como cada um influencia o treinamento. Também estudei o uso de DeepSpeed, principalmente para mover a memória do otimizador para a RAM da CPU e economizar VRAM, já que não seria necessário configurar treinamento distribuído por conta do uso de uma única GPU RTX 4090.
- Devido à lentidão e ao uso de GPU acima do esperado causados pelas dependências problemáticas, utilizei um dataset de treino com apenas 10.450 amostras e um dataset de validação com 550 amostras, treinando por apenas uma época. A estimativa inicial era de cerca de 4h30min. A tentativa de utilizar aproximadamente 100 mil amostras resultou em uma previsão superior a 40 horas, tornando o experimento inviável.
- Para agilizar o treino, reduzi o max_length das sequências para 1024, diminuindo a alocação na GPU e permitindo aumentar o batch real para 4 sequências. Também optei por utilizar acumulação de gradiente de 16 steps, elevando o batch efetivo para algo próximo de 16K tokens. Isso manteve o uso de VRAM entre aproximadamente 20 GB e 23 GB.
- As demais configurações estão disponíveis no repositório indicado na seção “Código”.
- Em paralelo, para fins de teste, iniciei o treinamento do “Qwen/Qwen2.5-0.5B” e, além de não apresentar qualquer problema, o modelo treinou significativamente mais rápido, com exatamente as mesmas configurações, estimando menos de 30 minutos

de execução. Esse comportamento reforça a conclusão de que a ineficiência no treinamento do Falcon-H1 foi causada pelos problemas de dependências.

Resultados

Falcon-H1

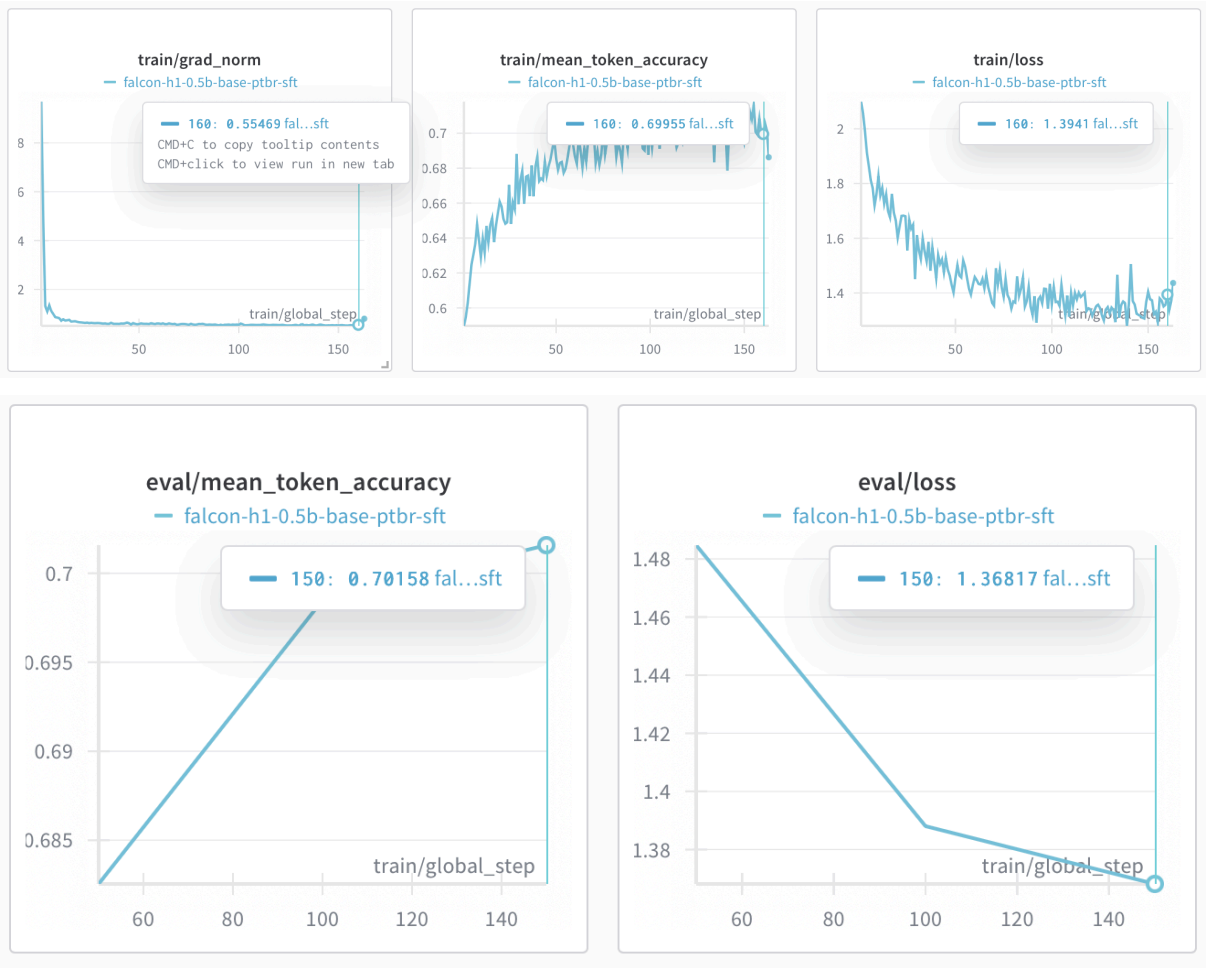


Figura 2: Métricas de treinamento e avaliação do modelo Falcon-H1-0.5B-Base após SFT em português. Os gráficos superiores mostram a evolução da norma dos gradientes, da acurácia média por token e da loss de treino ao longo dos steps. Os gráficos inferiores apresentam a acurácia média por token e a loss no conjunto de validação.

Qwen2.5

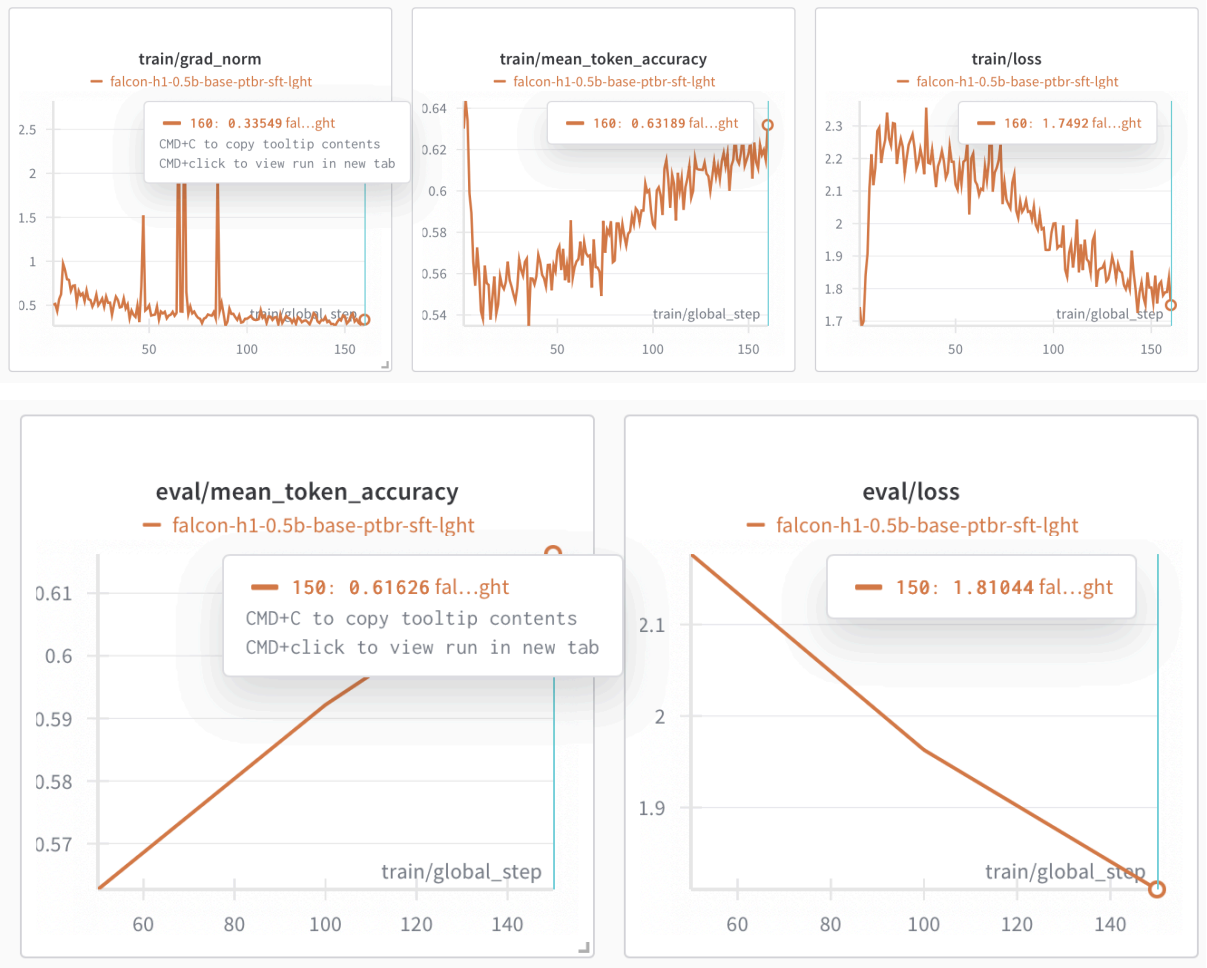


Figura 3: Métricas de treinamento e avaliação do modelo Qwen2-0.5B-Base após SFT em português. Os gráficos superiores mostram a evolução da norma dos gradientes, da acurácia média por token e da loss de treino ao longo dos steps. Os gráficos inferiores apresentam a acurácia média por token e a loss no conjunto de validação. A legenda está indicando o nome do modelo anterior por um **erro** ao executar o código de treino do Qwen.

Comparação

Por mais que não tenhamos conseguido finalizar essa etapa de SFT nas condições ideais, já que 10 mil amostras é uma quantidade muito baixa para fazer com que o modelo deixe de simplesmente completar o input e passe a seguir instruções, foi possível observar que, nas mesmas condições, o Falcon-H1 apresentou um SFT mais estável e, com a mesma quantidade de steps, ajustou-se melhor do que o Qwen. Ele obteve loss de treino e

de validação menores e apresentou acurácia média de tokens por amostra superior, tanto em treino quanto em validação. Em um primeiro momento, constatamos um padrão semelhante ao observado no pré-treino, em que a arquitetura híbrida demonstrou um treinamento mais eficiente. Isso não prova que, com SFT em maior escala ou com modelos maiores, o Qwen continuaria apresentando desempenho inferior, mas serve como primeiro indício. Além disso, reconheço que o ideal na etapa de SFT seria utilizar benchmarks relacionados ao domínio que se deseja avaliar, como tarefas de perguntas e respostas gerais, mas não foi possível realizar essa etapa a tempo.

Referências

[1] TEAM, Falcon. Falcon-H1: A Family of Hybrid-Head Language Models Redefining Efficiency and Performance. Disponível em: <https://falcon-lm.github.io/blog/falcon-h1/>. Acesso em: 2 dez. 2025.

[2] ZUO, Jingwei *et al.* Falcon-H1: A Family of Hybrid-Head Language Models Redefining Efficiency and Performance. arXiv, , 30 jul. 2025. Disponível em: <http://arxiv.org/abs/2507.22448>. Acesso em: 12 nov. 2025