UNIVERSIDADE DE SÃO PAULO
Instituto de Ciências Matemáticas e de Computação

Um caminho estratégico para a soberania digital através de LLMs nacionais

Cristovam Belizário Peres

Trabalho de Conclusão de Curso do MBA em Ciência de Dados (MBA-CD)

ICMC USP
SÃO CARLOS

Cristovam Belizário Peres

Um caminho estratégico para a soberania digital através de LLMs nacionais

Trabalho de Conclusão de Curso apresentado ao Instituto de Ciências Matemáticas e de Computação – ICMC-USP como parte dos requisitos para conclusão do MBA em Ciência de Dados.

Área de Concentração: Ciência de Dados

Orientador: Prof. Dr. Pedro Luiz Ramos

USP – São Carlos
Janeiro de 2026

Ficha catalográfica elaborada pela Biblioteca Prof. Achille Bassi
e Seção Técnica de Informática, ICMC/USP,
com os dados fornecidos pelo(a) autor(a)

A634m

Antonelli, Humberto Lidio
Modelo de teses e dissertações em LaTeX do ICMC / Humberto Lidio Antonelli; orientadora Renata Pontin de Mattos Fortes. -- São Carlos, 2017.
79 p.

Tese (Doutorado - Programa de Pós-Graduação em Ciências de Computação e Matemática Computacional) -- Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, 2017.

1. Modelo. 2. Monografia de qualificação. 3. Dissertação. 4. Tese. 5. Latex. I. Fortes, Renata Pontin de Mattos, orient. II. Título.

Cristovam Belizário Peres

A strategic path to digital sovereignty through national LLMs

Course completion work submitted to the Institute of Mathematics and Computer Sciences – ICMC-USP as part of the requirements for the MBA in Data Science.

Concentration Area: Data Science

Advisor: Prof. Dr. Pedro Luiz Ramos

USP – São Carlos
January 2026

RESUMO

PERES, C. B. **Um caminho estratégico para a soberania digital através de LLMs nacionais**. 2026. 121 p. Trabalho (Conclusão de Curso – MBA em Ciência de Dados) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos – SP, 2026.

Este trabalho investiga, sob a ótica de soberania digital, o desenvolvimento de modelos de linguagem em português brasileiro sob restrições realistas de hardware. A contribuição central é caracterizar empiricamente (i) desafios do treino do zero, por meio de um pipeline reprodutível iterado em seis versões (v1–v6) treinadas em subconjuntos do BrWaC, evidenciando custo de engenharia e sensibilidade a decisões de limpeza e tokenização; e (ii) uma alternativa pragmática baseada em pós-treinamento eficiente (QLoRA) sobre um modelo open-source de maior capacidade (LLaMA 3.1-8B). Na Trilha 2, foram executados *continued pretraining* (CPT) em português e *supervised fine-tuning* (SFT) instrucional (Canarim 10k), seguidos de avaliação extrínseca em QA, sumarização e reescrita. Os resultados mostram ganhos consistentes do SFT em relação ao baseline instruccional, com custo total de aproximadamente 5.22 horas de GPU (US$ 1.57) em uma única RTX 4090 sob demanda. Scripts, logs e artefatos são versionados para auditoria e replicação.

Palavras-chave: Large Language Models, Soberania Digital, Tokenização, QLoRA, Avaliação Extrínseca.

ABSTRACT

PERES, C. B. **A strategic path to digital sovereignty through national LLMs**. 2026. 121 p. Trabalho (Conclusão de Curso – MBA em Ciência de Dados) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos – SP, 2026.

This work investigates, through the lens of digital sovereignty, the development of Portuguese (PT-BR) language models under realistic hardware constraints. The main contribution is to empirically characterize (i) the challenges of training from scratch, via a reproducible pipeline iterated across six versions (v1–v6) trained on BrWaC subsets, highlighting engineering cost and sensitivity to cleaning and tokenization decisions; and (ii) a pragmatic alternative based on parameter-efficient post-training (QLoRA) on a higher-capacity open-source model (LLaMA 3.1-8B). In Track 2, we run continued pretraining (CPT) in Portuguese and instruction supervised fine-tuning (SFT) (Canarim 10k), followed by extrinsic evaluation on QA, summarization, and rewriting. Results show consistent gains from SFT over the instruction baseline, with a total cost of approximately 5.22 GPU-hours (US$ 1.57) on a single on-demand RTX 4090. Scripts, logs, and artifacts are versioned to support auditing and replication.

Keywords: Large Language Models, Digital Sovereignty, Tokenization, QLoRA, Extrinsic Evaluation.

LISTA DE ILUSTRAÇÕES

LISTA DE ALGORITMOS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

BPE	Byte Pair Encoding
CPT	Continued Pretraining
CUDA	Compute Unified Device Architecture
EM	Exact Match
F1	F1-score
FLOPs	Floating Point Operations
GPT	Generative Pre-trained Transformer
GPU	Graphics Processing Unit
GRU	Gated Recurrent Unit
JSONL	JSON Lines
LLM	Large Language Model
LoRA	Low-Rank Adaptation
LSTM	Long Short-Term Memory
PEFT	Parameter-Efficient Fine-Tuning
PII	Personally Identifiable Information
PPL	Perplexity
QA	Question Answering
QLoRA	Quantized Low-Rank Adaptation
RLHF	Reinforcement Learning from Human Feedback
RNN	Recurrent Neural Network
ROUGE	Recall-Oriented Understudy for Gisting Evaluation
SFT	Supervised Fine-Tuning
VRAM	Video Random Access Memory

SUMÁRIO

CAPÍTULO

1

INTRODUÇÃO

O domínio sobre modelos de linguagem de grande escala (LLMs) deixou de ser uma questão puramente técnica para se tornar um vetor central de poder geopolítico, econômico e informacional. Nações e corporações que lideram o desenvolvimento de modelos como o GPT da OpenAI (ChatGPT), Claude e Gemini não apenas impulsionam a fronteira da inovação, mas também moldam a forma como milhões de pessoas acessam e interpretam informações. Essa concentração de poder levanta questões críticas sobre soberania digital, segurança informacional e a transferência implícita de valores culturais e epistemológicos (ROSS; CONIA; NAVIGLI, 2023).

Diante deste cenário, a criação de um LLM nacional para o Brasil surge como uma necessidade estratégica. Contudo, os custos e desafios técnicos para treinar um modelo de ponta do zero são proibitivos para a maioria dos países. Este trabalho argumenta que existe um caminho alternativo e viável: o pós-treinamento e a adaptação de modelos de código aberto robustos. Em vez de apenas teorizar sobre a importância de um LLM brasileiro, este projeto se propõe a desenvolver um protótipo funcional (MVP/PoC), demonstrando um caminho prático para a capacitação tecnológica nacional.

A corrida pela inteligência artificial, sobretudo pelo controle de LLMs, determinará a relevância estratégica das nações nas próximas décadas. Modelos de fundação são hoje considerados infraestruturas críticas, com efeitos sistêmicos sobre a economia, a ciência e as políticas públicas (BOMMASANI *et al.*, 2021). A dependência de tecnologias estrangeiras cria uma vulnerabilidade estratégica, submetendo o país a decisões e vieses de modelos treinados em contextos socioculturais distintos.

Além disso, vivemos uma era de guerra informacional, onde o controle sobre os dados e os algoritmos que os processam é crucial. A capacidade de um país de desenvolver, alinhar e auditar seus próprios modelos de linguagem é fundamental para proteger dados sensíveis, garantir a integridade de processos democráticos e promover uma inovação alinhada às necessidades

locais, refletindo os valores, a legislação e a diversidade cultural brasileira (BAI *et al.*, 2023).

Contudo, a concretização de tal capacidade nacional enfrenta desafios técnicos consideráveis. A evolução dos LLMs, por exemplo, foi acelerada pela arquitetura Transformer (VASWANI *et al.*, 2017), que permitiu um salto em escala e capacidade de compreensão contextual. No entanto, o treinamento de modelos de fronteira demanda investimentos massivos em poder computacional, com custos que podem chegar a milhões de dólares (PATTERSON *et al.*, 2023). Empresas como a NVIDIA se tornaram centrais neste ecossistema, desenvolvendo chips especializados (GPUs) que são essenciais para o treinamento. Esse hardware de ponta, concentrado em poucas nações, principalmente EUA e China, torna o desenvolvimento de um modelo do zero um desafio de longo prazo.

Felizmente, o ecossistema de código aberto oferece uma alternativa estratégica. Iniciativas como LLaMA (TOUVRON *et al.*, 2023) e Qwen (Qwen Team, 2024) disponibilizaram modelos de alta performance que podem servir como base para adaptação. Essa abordagem reduz drasticamente os custos e permite focar esforços em etapas de maior valor agregado, como a especialização do modelo para o contexto brasileiro.

Diante desse panorama de desafios e oportunidades, este trabalho propõe uma abordagem prática para avaliar a viabilidade de um LLM nacional, que se desdobra em duas frentes principais. Primeiramente, realiza uma **caracterização prática** de custos e requisitos (VRAM, tempo e custo estimado) para o treinamento do zero em escala reduzida, sustentada por logs e medições ao longo das versões (v1–v6). Em segundo lugar, e como foco principal, desenvolve evidência empírica de uma alternativa mais viável: a adaptação de modelos de código aberto por pós-treinamento eficiente (CPT/SFT via LoRA/QLoRA).

A metodologia de implementação se organiza em duas trilhas complementares, conectadas pela mesma restrição prática de hardware e tempo de treinamento:

1. **Linha experimental (v1–v6): treino do zero em escala reduzida.** Esta trilha implementa e itera um pipeline completo (curadoria, limpeza, tokenização e treinamento) para quantificar, na prática, o custo e a sensibilidade de treinar um modelo Transformer do zero. O objetivo não é competir com modelos de fronteira, mas evidenciar a dificuldade inerente (dados, engenharia e tempo de GPU) e registrar as decisões que levaram à versão final (v6).

2. **Linha viável: adaptação via LoRA/QLoRA.** A partir de um modelo open-source de maior capacidade, aplica-se pós-treinamento eficiente em GPU para maximizar utilidade prática dentro do mesmo orçamento computacional. No escopo mínimo viável executado neste trabalho, esta trilha contempla (i) *continued pretraining* (CPT) com amostras do BrWaC para adaptação linguística e (ii) *supervised fine-tuning* (SFT) com um dataset instrucional em português brasileiro (Canarim), seguido de avaliação extrínseca em QA, sumarização e reescrita.

O objetivo deste projeto é, portanto, entregar (i) um protótipo funcional com evidências quantitativas do custo de uma linha "do zero" (v1–v6) e (ii) evidência empírica de uma alternativa pragmática de pós-treinamento eficiente (CPT/SFT via LoRA/QLoRA) em GPU única, com logs, métricas e artefatos reprodutíveis. Os próximos capítulos apresentam o referencial teórico que embasa estas escolhas, a descrição da metodologia aplicada e a análise dos resultados obtidos.

1.1 Contribuições

1. **Pipeline reprodutível e rastreável (v1–v6):** implementação iterativa de um fluxo completo (dados, tokenização, treino, avaliação) para explicitar custos e desafios do treinamento do zero em escala reduzida.

2. **Evidência do impacto da tokenização:** registro sistemático de métricas intrínsecas, amostras padronizadas e *guardrails* para mostrar que decisões de tokenização podem dominar a qualidade percebida.

3. **Execução de pós-treinamento eficiente (Trilha 2):** CPT e SFT via QLoRA em um modelo 8B open-source, com logs estruturados e custos reportados.

4. **Avaliação extrínseca comparável:** protocolo de QA/sumarização/reescrita com repetição em múltiplas seeds de amostragem e artefatos versionados para auditoria.

CAPÍTULO

2

REVISÃO HISTÓRICA DOS LLMS

2.1 Primeiras tentativas de processamento de linguagem natural

O interesse em fazer máquinas compreenderem e gerarem linguagem humana remonta aos primórdios da computação. Uma das iniciativas mais emblemáticas desse período foi o desenvolvimento do programa **ELIZA**, criado por Joseph Weizenbaum no MIT, durante a década de 1960 (WEIZENBAUM, 1966). ELIZA foi projetada para simular um psicoterapeuta rogeriano, interagindo com usuários por meio de diálogos escritos. Seu funcionamento baseava-se em regras simples de reconhecimento de padrões e substituição de palavras-chave, permitindo que o sistema identificasse termos relevantes nas frases do usuário e respondesse com perguntas ou afirmações genéricas.

A principal característica de ELIZA era sua abordagem simbólica e determinística: não havia aprendizado automático, mas sim um conjunto fixo de scripts e regras pré-definidas (JURAFSKY; MARTIN, 2023). Apesar dessa simplicidade, ELIZA surpreendeu ao criar a ilusão de compreensão, levando muitos usuários a atribuírem ao programa uma inteligência que ele não possuía de fato. Esse fenômeno ficou conhecido como o "efeito ELIZA", ilustrando como interações superficiais podem ser percebidas como inteligentes quando envolvem linguagem natural (WEIZENBAUM, 1966; JURAFSKY; MARTIN, 2023).

Além de ELIZA, outros sistemas simbólicos surgiram nas décadas seguintes, como parsers sintáticos e chatbots baseados em regras. Um exemplo notável é o sistema SHRDLU, desenvolvido por Terry Winograd (WINOGRAD, 1972), que utilizava gramáticas formais e árvores sintáticas para decompor frases e responder a comandos em um ambiente restrito. Esses sistemas buscavam decompor frases em estruturas gramaticais ou responder a comandos específicos, utilizando gramáticas formais e árvores sintáticas (JURAFSKY; MARTIN, 2023). Embora tenham representado avanços importantes para a época, esses métodos enfrentavam

limitações severas: eram altamente dependentes de regras manuais, pouco escaláveis e incapazes de lidar com a ambiguidade, a variabilidade e a riqueza semântica da linguagem humana.

Os principais problemas encontrados nessas abordagens estavam relacionados à rigidez dos sistemas e à incapacidade de generalizar para contextos não previstos pelos desenvolvedores. A ausência de mecanismos de aprendizado impedia que os sistemas evoluíssem a partir de novas interações ou dados, restringindo seu uso a domínios muito específicos e controlados (JURAFSKY; MARTIN, 2023).

Apesar dessas limitações, as primeiras tentativas de processamento de linguagem natural foram fundamentais para estabelecer as bases conceituais do campo (TURING, 1950; JURAFSKY; MARTIN, 2023). A rigidez e a falta de escalabilidade dos sistemas simbólicos deixaram claro que uma nova abordagem era necessária — uma que pudesse aprender com dados, em vez de ser explicitamente programada. Esse foi o vácuo que o aprendizado de máquina começou a preencher.

2.2 Evolução para Machine Learning

Com o avanço da computação e o aumento da disponibilidade de dados digitais, as limitações dos sistemas puramente simbólicos tornaram-se cada vez mais evidentes. Isso impulsionou a transição para abordagens baseadas em **aprendizado de máquina (Machine Learning, ML)**, especialmente a partir das décadas de 1980 e 1990 (SEBASTIANI, 2002; JURAFSKY; MARTIN, 2023). O objetivo passou a ser permitir que os sistemas aprendessem padrões linguísticos a partir de exemplos, em vez de depender exclusivamente de regras manuais.

Os primeiros algoritmos de ML aplicados ao processamento de linguagem natural incluíam métodos estatísticos como **Naive Bayes**, **máquinas de vetores de suporte (SVM)** (JOACHIMS, 1998) e **árvores de decisão**. Essas técnicas eram empregadas em tarefas como classificação de textos, análise de sentimentos, detecção de spam e extração de informações (SEBASTIANI, 2002). Uma característica marcante desse período foi o uso de representações vetoriais simples, como o modelo **bag-of-words** e o **TF-IDF** (Term Frequency-Inverse Document Frequency), que transformavam textos em conjuntos de números baseados na frequência das palavras (SALTON; BUCKLEY, 1988).

Essas abordagens trouxeram avanços significativos em relação aos sistemas simbólicos. Ao permitir que modelos fossem treinados com grandes volumes de dados rotulados, tornou-se possível capturar regularidades estatísticas da linguagem e adaptar os sistemas a diferentes domínios e tarefas (SEBASTIANI, 2002; JURAFSKY; MARTIN, 2023). Além disso, a automação do processo de aprendizado reduziu a dependência de especialistas em linguística para a criação de regras, tornando o desenvolvimento de aplicações de processamento de linguagem natural mais ágil e escalável.

No entanto, os métodos baseados em machine learning tradicional também apresentavam limitações importantes. Por tratarem o texto como uma coleção de palavras independentes, ignoravam a ordem e o contexto em que as palavras apareciam, o que dificultava a compreensão de estruturas sintáticas e semânticas mais complexas (JURAFSKY; MARTIN, 2023). Além disso, a necessidade de grandes conjuntos de dados rotulados para treinamento era um desafio, especialmente em domínios especializados ou em idiomas com menos recursos.

Apesar desses obstáculos, a adoção do aprendizado de máquina representou um avanço fundamental para o campo. Ela abriu caminho para o desenvolvimento de técnicas mais sofisticadas, capazes de capturar relações contextuais e semânticas, e preparou o terreno para a próxima grande revolução: o **aprendizado profundo (Deep Learning)** (JURAFSKY; MARTIN, 2023).

2.3 Avanço com Deep Learning

A partir do final dos anos 2000, o campo do processamento de linguagem natural (PLN) foi profundamente transformado pelo **aprendizado profundo (Deep Learning)**. Esse avanço foi impulsionado pelo aumento do poder computacional, pela disponibilidade de grandes volumes de dados e pelo desenvolvimento de novas arquiteturas de redes neurais (LECUN; BENGIO; HINTON, 2015; JURAFSKY; MARTIN, 2023).

O primeiro grande desafio que o deep learning superou foi a representação de palavras. Modelos anteriores, como bag-of-words, tratavam as palavras como itens isolados. A revolução veio com os **word embeddings** a partir de 2013. Modelos como **Word2Vec** (MIKOLOV, 2013) e **GloVe** (PENNINGTON; SOCHER; MANNING, 2014) aprenderam a representar palavras como vetores densos em um espaço multidimensional. Nessa representação, palavras com significados semelhantes ficavam próximas, permitindo que os modelos capturassem relações semânticas e sintáticas (por exemplo, a relação entre "rei"e "rainha"ser similar à de "homem"e "mulher"). Essa capacidade de representar o significado das palavras foi um pilar para os avanços subsequentes.

Com palavras representadas de forma significativa, o desafio seguinte foi compreender a ordem e o contexto em que elas aparecem. As **Redes Neurais Recorrentes (RNNs)** foram a primeira arquitetura de deep learning projetada para processar sequências, mantendo um estado interno ou "memória"do que foi visto anteriormente. No entanto, as RNNs tradicionais sofriam com o problema do gradiente desvanecente, o que as impedia de capturar dependências em trechos longos de texto (BENGIO; SIMARD; FRASCONI, 1994). Para solucionar essa limitação, foram desenvolvidas variantes mais robustas, como as **Long Short-Term Memory (LSTM)** (HOCHREITER; SCHMIDHUBER, 1997) e as **Gated Recurrent Units (GRU)** (CHO, 2014). Essas arquiteturas introduziram "portões"(gates) que controlam o fluxo de informação, permitindo que a rede se lembre de informações relevantes por períodos muito mais longos.

Paralelamente, as **Redes Neurais Convolucionais (CNNs)**, tradicionalmente usadas em visão computacional, foram adaptadas com sucesso para o PLN (KIM, 2014). Em vez de

processar o texto sequencialmente, as CNNs aplicam filtros que deslizam sobre as sequências de palavras para identificar padrões locais, como frases ou expressões idiomáticas, independentemente de sua posição no texto.

A combinação de word embeddings com arquiteturas sequenciais (LSTMs, GRUs) e convolucionais (CNNs) permitiu que os modelos de linguagem alcançassem um novo patamar de compreensão contextual, preparando o terreno para a arquitetura que viria a definir a era moderna do PLN: os Transformers.

2.4 A Revolução dos Transformers e os Modelos de Linguagem de Grande Escala (LLMs)

Em 2017, o campo do processamento de linguagem natural foi novamente transformado com a publicação do artigo "Attention is All You Need" (VASWANI *et al.*, 2017), que apresentou ao mundo a arquitetura **Transformer**. Esse modelo representou uma ruptura fundamental com as abordagens sequenciais, como as RNNs. A principal inovação dos Transformers é a capacidade de processar todas as palavras de uma sequência simultaneamente, permitindo uma paralelização massiva e um treinamento muito mais eficiente.

O coração do Transformer é o **mecanismo de atenção** (*attention mechanism*). Em vez de depender de uma memória sequencial, a atenção permite que o modelo, ao analisar uma palavra, pondere dinamicamente a importância de todas as outras palavras na sequência. Matematicamente, para cada palavra, o modelo gera vetores de *query* (consulta), *key* (chave) e *value* (valor). A pontuação de atenção é calculada comparando o *query* de uma palavra com a *key* de todas as outras, determinando o quanto de "foco"cada palavra deve receber. O resultado é uma representação contextual rica, que captura relações complexas de curto e longo alcance, resolvendo ambiguidades que eram desafiadoras para modelos anteriores.

A eficiência e o poder de contextualização dos Transformers abriram caminho para a era dos **Modelos de Linguagem de Grande Escala (Large Language Models, LLMs)**. A nova paradigma consistia em duas fases: primeiro, o **pré-treinamento**, no qual um modelo é treinado em uma quantidade massiva de texto não rotulado para aprender padrões gerais da linguagem, gramática, fatos e raciocínio. Em seguida, a fase de **ajuste fino** (*fine-tuning*), onde o modelo pré-treinado é adaptado para tarefas específicas (como classificação, tradução ou resposta a perguntas) com um conjunto de dados muito menor e rotulado.

Essa abordagem levou a avanços sem precedentes, com o surgimento de modelos icônicos:

- **BERT (Bidirectional Encoder Representations from Transformers, 2018):** Desenvolvido pelo Google, o BERT inovou ao utilizar o encoder do Transformer para aprender representações de palavras considerando o contexto de ambos os lados (esquerdo e direito)

simultaneamente, alcançando resultados estado da arte em uma vasta gama de tarefas de compreensão de linguagem (DEVLIN *et al.*, 2018).

- **GPT (Generative Pre-trained Transformer, 2018 em diante):** Criado pela OpenAI, a família de modelos GPT utiliza o decoder do Transformer com foco na geração de texto. Do GPT-1 ao GPT-3 e além, esses modelos demonstraram uma capacidade impressionante de gerar textos coerentes, criativos e contextualmente relevantes a partir de um prompt (RADFORD, 2018; BROWN, 2020).

- **T5 (Text-to-Text Transfer Transformer, 2020):** Também do Google, o T5 propôs um framework unificado, tratando todas as tarefas de PLN como um problema de "texto-para-texto". O modelo recebe uma instrução em texto (ex: "traduza Inglês para Português: ...") e gera a saída também em texto, simplificando a aplicação do modelo a diferentes problemas (RAFFEL, 2020).

A trajetória desses modelos, especialmente da série GPT, demonstrou que o escalonamento massivo — aumentando exponencialmente a quantidade de parâmetros e os dados de treinamento — resultava em capacidades emergentes e um salto de performance. O avanço, contudo, não se limitou a essa corrida por escala. À medida que os LLMs se tornaram mais capazes, o desafio de alinhar seu comportamento a objetivos e valores humanos tornou-se crítico. Surgiram, então, técnicas de pós-treinamento para refinar as respostas dos modelos, como o Aprendizado por Reforço com Feedback Humano (RLHF), que utiliza avaliações humanas para guiar o aprendizado e tornar os modelos mais úteis e seguros (BAI *et al.*, 2023).

2.4.1 Pós-treinamento: adaptação e alinhamento

Embora o termo *fine-tuning* seja frequentemente usado de forma ampla, é útil diferenciar objetivos distintos na fase de pós-treinamento: (i) adaptação linguística/domínio, (ii) seguimento de instruções e (iii) alinhamento por preferências. No primeiro caso, o *continued pretraining* (CPT) continua o pré-treinamento com o mesmo objetivo auto-supervisionado (predição do próximo token), porém com dados do idioma/domínio de interesse (GURURANGAN *et al.*, 2020). No segundo, o *supervised fine-tuning* (SFT) — frequentemente referido como *instruction tuning* quando usa pares instrução–resposta — ensina o modelo a responder a comandos em formatos controlados (e.g., QA, sumarização, reescrita), com exemplos rotulados (OUYANG *et al.*, 2022). Por fim, técnicas de alinhamento como RLHF refinam o comportamento a partir de sinais de preferência humana, complementando o SFT ao penalizar respostas indesejadas e reforçar as preferidas (BAI *et al.*, 2023).

Do ponto de vista computacional, essas etapas tornaram-se viáveis em hardware de consumo com técnicas de *parameter-efficient fine-tuning* (PEFT) — como LoRA — nas quais o modelo base é mantido congelado e apenas adaptadores pequenos são treinados, representando uma fração dos parâmetros totais (HU *et al.*, 2021). QLoRA estende essa estratégia ao manter o

modelo base quantizado (e.g., 4-bit) enquanto treina os adaptadores LoRA em maior precisão, reduzindo VRAM e viabilizando o ajuste de modelos maiores em GPU única (DETTMERS *et al.*, 2023). Estes conceitos são detalhados no Capítulo 3 e aplicados na trilha de pós-treinamento descrita no Capítulo 4.

Paralelamente, o ecossistema de IA dividiu-se. Enquanto modelos de ponta como os da série GPT permaneceram proprietários, um movimento de código aberto (open-source) ganhou força, buscando democratizar o acesso a essa tecnologia. Iniciativas como LLaMA (TOUVRON *et al.*, 2023) e Qwen (Qwen Team, 2024) disponibilizaram modelos poderosos para a comunidade, viabilizando a pesquisa e o desenvolvimento de aplicações adaptadas a contextos específicos — exatamente a abordagem adotada neste trabalho.

2.5 A Era dos LLMs e o Desafio da Representação Cultural

A ascensão dos modelos de linguagem de grande escala (LLMs), como BERT, GPT e T5, representa o estado da arte em processamento de linguagem natural. Seu sucesso é inegável, mas também expôs uma limitação fundamental: a sua perspectiva predominantemente anglocêntrica.

Quando aplicados a outros idiomas, como o português do Brasil, esses modelos podem apresentar um desempenho satisfatório em tarefas genéricas, mas frequentemente falham em capturar as complexidades e especificidades locais. Gírias, expressões idiomáticas, contextos culturais, referências regionais e a própria estrutura semântica da língua podem ser mal interpretados ou simplesmente ignorados por modelos que não foram treinados nativamente com dados brasileiros (CASWELL *et al.*, 2021).

Essa lacuna cria uma barreira para o desenvolvimento de aplicações de IA verdadeiramente eficazes e culturalmente conscientes para o contexto brasileiro. Mais do que uma simples questão de tradução, trata-se de um desafio de representação. A dependência de modelos estrangeiros pode perpetuar vieses e criar tecnologias que não atendem adequadamente às necessidades da população local.

É nesse cenário que surge a necessidade crítica de desenvolver LLMs 100% brasileiros. Um modelo treinado desde o início com um corpus massivo e diversificado de textos em português do Brasil tem o potencial de não apenas compreender melhor as sutilezas do idioma, mas também de incorporar o vasto conhecimento cultural, social e histórico do país. Este trabalho se insere exatamente nessa fronteira, buscando contribuir para a construção de uma inteligência artificial que fale, de fato, a nossa língua.

CAPÍTULO

3

FUNDAMENTOS TÉCNICOS E ARQUITETURAIS DOS LLMS

Este capítulo consolida os **fundamentos técnicos essenciais** para entender as escolhas de arquitetura, as limitações de hardware e a metodologia experimental do Capítulo 4. O foco é didático: apresentar apenas o necessário para acompanhar (i) a Trilha 1 (v1–v6) e (ii) a Trilha 2 (CPT/SFT via QLoRA), evitando aprofundamentos que não foram implementados.

Seção 3.1	Tokenização (char-level → subword) → usada na Trilha 1 (v1–v6); a escolha do tokenizer se mostrou determinante para legibilidade/qualidade.
Seção 3.2	Arquiteturas na Trilha 1 (RNN → Transformer) → v1–v4 (modelos sequenciais) e v5–v6 (Transformer em GPU única).
Seção 3.3	Compute e VRAM (GPU única) → justifica escala reduzida em v1–v6 e limitações de contexto/treino.
Seção 3.4	LoRA/QLoRA → viabiliza a Trilha 2 (CPT e SFT) em 1x RTX 4090 (24 GB).
Seção 3.5	Scaling laws (Kaplan/Chinchilla) → contextualiza por que v1–v6 não compete com modelos de fronteira e por que pós-treinamento é o caminho prático.

Figura 1 – Roadmap de conexão entre fundamentos (Cap. 3) e experimentos (Cap. 4).

3.1 Tokenização (v1–v6): char-level e subword

Um modelo de linguagem não consome texto diretamente: ele opera sobre uma sequência discreta de **tokens** (IDs), definida por um **tokenizer** e seu vocabulário. Essa escolha afeta simultaneamente (i) a **qualidade** (fragmentação, legibilidade e consistência lexical) e (ii) o **custo** (comprimento efetivo de sequência e taxa de tokens/segundo).

Para fixar a terminologia usada ao longo do capítulo: neste trabalho, **token** é a unidade básica que o modelo lê e prediz (por exemplo, um caractere, uma subpalavra ou um pedaço de

palavra). Já o **tokenizer** é o componente (procedimento + vocabulário) que transforma texto em tokens/IDs (*encode*) e permite reconstruir texto a partir de IDs (*decode*).

3.1.1 Char-level

Em tokenização *char-level*, cada caractere é um token. É uma escolha simples, sem OOV (*out-of-vocabulary*) e sem necessidade de treinar um tokenizer, mas tende a produzir sequências muito mais longas (mais passos de predição), o que aumenta o custo computacional e pode dificultar o aprendizado de regularidades semânticas de alto nível.

Exemplo (ilustrativo). Texto `casa` → tokens `[c, a, s, a]` (4 tokens).

3.1.2 Subword (SentencePiece)

Em tokenização *subword*, palavras são segmentadas em unidades menores (subpalavras) aprendidas a partir de um corpus (por exemplo, SentencePiece). SentencePiece é uma biblioteca amplamente usada para treinar tokenizers *subword* diretamente sobre texto cru, com algoritmos como *unigram* e BPE. Em geral, isso reduz o comprimento da sequência em relação ao char-level e melhora a modelagem de morfologia e composição de palavras, especialmente em português. Em contrapartida, um tokenizer mal ajustado pode gerar **fragmentação excessiva** (muitas peças curtas), o que degrada a legibilidade mesmo quando métricas intrínsecas (loss/perplexidade) parecem aceitáveis.

Exemplo (ilustrativo). Texto `infraestrutura` → tokens `[infra, estrutura]` ou `[in, fra, estrutura]`, dependendo do vocabulário aprendido.

No Capítulo 4, as versões v1–v3 utilizam abordagem char-level para viabilizar iteração rápida e eliminar variáveis adicionais no início do pipeline. A v4 introduz subwords (SentencePiece), representando um salto qualitativo perceptível nos exemplos. Nas versões seguintes (v5–v6), a principal descoberta empírica foi que **a escolha do tokenizer pode dominar ganhos arquiteturais marginais**, motivando a inclusão de *guardrails* e análises sistemáticas de amostras.

3.2 Arquiteturas na Trilha 1 (v1–v6): RNN → Transformer

3.2.1 Modelos sequenciais (v1–v4): RNN/GRU/LSTM

Nas primeiras versões, a Trilha 1 utilizou modelos sequenciais (RNNs e variações como GRU/LSTM). Esses modelos processam a sequência token a token, mantendo um *estado oculto* que resume o contexto anterior. A escolha foi adequada para prototipagem: implementação

simples, treinamento viável em GPU única e foco na construção do pipeline (dados, validação e geração).

Exemplo. Dada uma sequência de tokens $[t_1, t_2, t_3]$, uma RNN atualiza um estado h_t passo a passo (primeiro t_1, depois t_2, etc.) e usa h_t para predizer o próximo token. Isso contrasta com Transformers, que calculam atenção entre posições e exploram paralelismo (discutido a seguir).

3.2.2 Transformer (v5–v6): atenção e paralelização

Transformers (VASWANI *et al.*, 2017) substituem recorrência por **atenção** para modelar dependências entre tokens. A ideia central é que, ao processar uma posição i, o modelo calcula pesos que indicam o quanto cada outra posição j deve contribuir para a representação final daquela posição.

Em sua forma padrão (*scaled dot-product attention*), para matrizes de consultas Q, chaves K e valores V, define-se:

$$\text{Attention}(Q,K,V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V \tag{3.1}$$

O *multi-head attention* replica esse mecanismo em múltiplas "cabeças", permitindo que diferentes padrões de relacionamento (por exemplo, concordância, contexto temático, dependências longas) sejam capturados em paralelo, e depois combinados.

No Capítulo 4, a Trilha 1 usa v1–v4 com modelos sequenciais (RNN/GRU/LSTM) para viabilizar prototipagem e evolução rápida do pipeline. A partir de v5, a adoção de Transformer permite explorar paralelização em GPU, mas introduz custo quadrático no comprimento de sequência, o que impõe limites práticos de `seq_len` sob VRAM fixa (discutido a seguir).

3.3 Requisitos computacionais em GPU única: compute e VRAM

Em treinamento de modelos autoregressivos, o custo computacional cresce com (i) o tamanho do modelo (parâmetros) e (ii) a quantidade de dados (tokens). Além disso, a **VRAM** é frequentemente o gargalo principal em GPU única, pois precisa acomodar pesos, estados do otimizador e ativações.

3.3.1 Regra prática de compute (ordem de grandeza)

Como aproximação útil, o compute total de treino pode ser escrito como:

$$C \approx 6ND \tag{3.2}$$

onde N é o número de parâmetros e D o número de tokens vistos no treinamento (contexto de *scaling laws* (KAPLAN *et al.*, 2020; HOFFMANN *et al.*, 2022)). O objetivo aqui é didático: mostrar por que treinar do zero em escala de LLMs rapidamente sai do orçamento de uma única GPU.

3.3.2 Por que 7B não cabe em 24 GB (treino completo)

Na prática, o treinamento é feito com otimizadores adaptativos como o Adam, que mantêm estados adicionais por parâmetro (estimativas do primeiro e do segundo momento dos gradientes). Em configurações típicas, esses estados ficam em 32 bits (*precisão plena*), mesmo quando o treinamento usa precisão mista para acelerar. Assim, em ordem de grandeza, cada parâmetro requer: parâmetro, gradiente e dois momentos, o que leva a uma conta de VRAM que excede 24 GB para modelos de bilhões de parâmetros, motivando (i) protótipos reduzidos na Trilha 1 e (ii) pós-treinamento eficiente na Trilha 2.

No Capítulo 4, essa restrição aparece diretamente: (i) v1–v6 são modelos pequenos para permitir iteração e rastreabilidade em GPU única; (ii) a Trilha 2 usa QLoRA para adaptar um modelo maior (LLaMA 3.1-8B) com custo e VRAM viáveis.

3.4 Pós-treinamento eficiente: LoRA e QLoRA

O pós-treinamento eficiente busca adaptar modelos grandes com poucos parâmetros treináveis. Em **LoRA** (HU *et al.*, 2021), em vez de atualizar uma matriz de pesos W, aprende-se uma atualização de *rank* reduzido (*low-rank*):

$$W' = W + BA \tag{3.3}$$

onde $A \in \mathbb{R}^{r\times k}$ e $B \in \mathbb{R}^{d\times r}$ usam uma dimensão interna r pequena (por exemplo, 8 ou 16). Em vez de treinar todos os $d \cdot k$ elementos de $W \in \mathbb{R}^{d\times k}$, treina-se apenas $r(d+k)$ parâmetros nos adaptadores: quanto maior r, maior a capacidade do ajuste, mas também maior o custo em VRAM e tempo. Nesse sentido, $\Delta W = BA$ fica restrita a um subespaço de dimensão r.

Em termos de ordem de grandeza, a redução pode ser grande: por exemplo, com $d = k = 4096$ e $r = 8$, atualizar W exigiria treinar $\approx 16{,}8$ milhões de parâmetros, enquanto LoRA treina $r(d+k) = 65\,536$ parâmetros (redução de $\sim 256\times$). Na prática, a economia de VRAM vem do fato de que gradientes e estados do otimizador (por exemplo, do Adam) precisam ser mantidos apenas para os parâmetros treináveis; ao treinar apenas os adaptadores, evita-se armazenar estados completos para todos os pesos do modelo base.

QLoRA (DETTMERS *et al.*, 2023) combina LoRA com quantização do modelo base (por exemplo, 4-bit), reduzindo o consumo de VRAM e tornando possível treinar/adaptar em GPU única com 24 GB, sem re-treinar todos os pesos.

No Capítulo 4, a Trilha 2 usa QLoRA para executar (i) CPT em PT-BR (BrWaC 10k) e (ii) SFT instrucional (Canarim 10k), com avaliação extrínseca em tarefas downstream. O foco é maximizar utilidade prática dentro de um orçamento computacional realista.

3.5 Scaling laws: implicações para a estratégia em duas trilhas

Scaling laws são regularidades **empíricas** observadas em grandes famílias de experimentos: elas descrevem como métricas como *loss* tendem a diminuir quando se aumenta (i) o tamanho do modelo (número de parâmetros), (ii) a quantidade de dados (tokens) e (iii) o compute total. Em geral, essas curvas seguem leis de potência com **retornos decrescentes** (KAPLAN *et al.*, 2020), e por isso são úteis para estimar ordem de grandeza de custo e ganhos esperados. Hoffmann et al. (HOFFMANN *et al.*, 2022) reforçam que, dado um budget de compute, há um balanço ótimo entre tamanho do modelo e quantidade de dados (regra conhecida como *compute-optimal*).

Essas leis motivam a estrutura do projeto apresentada no Capítulo 4: v1–v6 não buscam competir com modelos de fronteira; seu valor está em evidenciar custos, armadilhas e decisões de pipeline (limpeza/tokenização/treino) em um cenário reprodutível. A Trilha 2, por sua vez, opera em um regime de escala mais próximo do uso prático ao adaptar um modelo 8B já pré-treinado, demonstrando ganhos mensuráveis em tarefas de QA, sumarização e reescrita com baixo custo.

CAPÍTULO

4

PROJETO PRÁTICO: DESENVOLVIMENTO DE LLM ESPECIALIZADO PARA O CONTEXTO BRASILEIRO

4.1 Estratégia experimental

O trabalho prático foi estruturado para sustentar a narrativa central desta dissertação: (i) quantificar, em uma escala reduzida e reprodutível, a dificuldade de treinar um modelo do zero (v1–v6) e (ii) contrastar esta abordagem com uma alternativa viável sob restrições reais de hardware, baseada em pós-treinamento eficiente (LoRA/QLoRA) de um modelo open-source.

Trilha 1 – Treinamento do zero (v1–v6). Implementa-se e itera um pipeline completo de dados (limpeza e tokenização) e treinamento de um Transformer em português (a partir do BrWaC), registrando decisões e métricas por versão. O objetivo é fornecer evidência concreta do custo (tempo, engenharia e dados) mesmo em uma escala muito menor do que modelos de fronteira.

Trilha 2 – Pós-treinamento eficiente (LoRA/QLoRA). Utiliza-se um modelo base open-source de maior capacidade, realizando (i) *continued pretraining* (CPT) em português e (ii) *supervised fine-tuning* (SFT) com dados instrucionais em PT-BR. Esta trilha busca maximizar utilidade prática no mesmo orçamento computacional, e é detalhada junto aos experimentos correspondentes.

Para garantir rastreabilidade, os artefatos experimentais (tokenizers, modelos, mapeamentos, logs e amostras) são armazenados no repositório do projeto (diretório <versions/>).

4.2 Análise Técnica do Ambiente Computacional

O desenvolvimento de LLMs requer uma análise rigorosa dos recursos computacionais disponíveis para determinar estratégias viáveis de implementação. Esta seção descreve o ambiente computacional utilizado e as principais restrições que motivam as escolhas metodológicas.

4.2.1 Ambiente de execução (Vast.ai)

Os experimentos desta dissertação foram executados em instâncias sob demanda na Vast.ai, utilizando GPUs de consumo que variaram ao longo das versões: v1–v4 em RTX 3080, v5/v5b em RTX 3090 (24 GB) e v6 e Trilha 2 em RTX 4090 (24 GB). Para garantir reprodutibilidade, logs e artefatos (tokenizers, modelos, mapeamentos, histórico de treino e amostras) foram armazenados no repositório do projeto, com os resultados consolidados do v6 no diretório <versions/v6-release-candidate/> e os artefatos da Trilha 2 em <versions/trilha2-lora/>.

4.2.2 Análise de Capacidades para Treinamento de LLMs

4.2.2.1 Limitações Críticas de Memória GPU

Para treinamento de LLMs, a VRAM disponível é o fator limitante principal. Baseado nas análises dos fundamentos técnicos, os requisitos de memória para diferentes tamanhos de modelo são:

Modelo de 7B parâmetros (treinamento completo):

$$\text{Memória}_{\text{parâmetros}} = 7 \times 10^9 \times 4 \text{ bytes} = 28 \text{ GB} \tag{4.1}$$

$$\text{Memória}_{\text{gradientes}} = 7 \times 10^9 \times 4 \text{ bytes} = 28 \text{ GB} \tag{4.2}$$

$$\text{Memória}_{\text{otimizador}} = 7 \times 10^9 \times 8 \text{ bytes} = 56 \text{ GB} \tag{4.3}$$

$$\text{Memória}_{\text{total}} = 28 + 28 + 56 = 112 \text{ GB} \tag{4.4}$$

Análise de viabilidade:

$$\text{Déficit}_{\text{VRAM}} = 112 \text{ GB} - 24 \text{ GB} = 88 \text{ GB} \tag{4.5}$$

O déficit de 88 GB evidencia que o treinamento completo de um modelo de 7B parâmetros é inviável em uma única GPU de 24 GB sem estratégias avançadas de paralelismo e otimizações de memória. Por esse motivo, a trilha experimental (v1–v6) foi conduzida em escala reduzida, e a trilha prática prioriza pós-treinamento eficiente (LoRA/QLoRA) em modelos pré-treinados.

4.2.2.2 Custo computacional

Além da VRAM, o custo de treinamento cresce com o número de parâmetros e com o número de tokens processados. Mesmo protótipos reduzidos podem demandar muitas horas de

GPU; por isso, este trabalho reporta tempos observados e logs reprodutíveis, e contrasta treino do zero (v1–v6) com alternativas de pós-treinamento eficientes em GPU única.

Para sustentar a evidência dos custos reportados, a Tabela 1 resume as fontes (no repositório) usadas para computar tempos e verificar a GPU utilizada em cada execução principal (valores arredondados). Os tempos apresentados nas Tabelas 7, 8 e 9 foram extraídos desses logs.

Execução	Tempo (h)	Fonte (tempo)	Fonte (GPU)
<v1>	1.1	<versions/v1-char-rnn/logs/train_brwac_v1_20k.json>	<versions/v1-char-rnn/logs/train_brwac_v1_20k.json> (RTX 3080)
<v2>	0.9	<versions/v2-char-lm/logs/train_brwac_v2_20k.json>	<versions/v2-char-lm/logs/train_brwac_v2_20k.json> (RTX 3080)
<v3>	2.0	<versions/v3-stacked-lstm/logs/train_brwac_v3_20k.json>	<versions/v3-stacked-lstm/logs/train_brwac_v3_20k.json> (RTX 3080)
<v4_subword>	1.2	<versions/v4-subword-lstm/logs/train_brwac_v4_subword.json>	<versions/v4-subword-lstm/logs/train_brwac_v4_subword.json> (RTX 3080)
<v4_subword_ep6>	3.7	<versions/v4-subword-lstm/logs/train_brwac_v4_subword_ep6.json>	<versions/v4-subword-lstm/logs/train_brwac_v4_subword_ep6.json> (RTX 3080)
<v4k_bf_ep2>	1.2	<versions/v4-subword-lstm/logs/train_brwac_v4_subword_v4k_bf_ep2.json>	<versions/v4-subword-lstm/logs/train_brwac_v4_subword_v4k_bf_ep2.json> (RTX 3080)
<v4k_bf_lstm_ep2>	1.4	<versions/v4-subword-lstm/logs/train_brwac_v4_lstm_v4k_bf_ep2.json>	<versions/v4-subword-lstm/logs/train_brwac_v4_lstm_v4k_bf_ep2.json> (RTX 3080)
<v5_main>	3.2	<versions/v5-transformer/logs/train_v5_main.json>	<versions/v5-transformer/logs/train_v5_scale.log> (RTX 3090)
<v5b_long>	2.9	<versions/v5b-transformer/logs/metrics_v5b_long/metrics_epoch_*.json>	<versions/v5b-transformer/logs/long_stdout_20251105_relaunch.log> (RTX 3090)
<v6_rc16k_v5b>	13.3	<versions/v6-release-candidate/logs/train_v6_rc16k_v5b.json>	<versions/v6-release-candidate/logs/run_v6_rc16k_v5b.log> (RTX 4090)
<v6_rc16k_v5b_aligned>	16.6	<versions/v6-release-candidate/logs/train_v6_rc16k_v5b_aligned.json>	<versions/v6-release-candidate/logs/train_v6_rc16k_v5b_aligned.out> (RTX 4090)
<cpt_qlora_llama31_8b_brwac10k>	2.35	<versions/trilha2-lora/logs/train_cpt_qlora.json>	<versions/trilha2-lora/logs/train_cpt_qlora.json> (RTX 4090)
<sft_qlora_llama31_8b_instruct_canarim10k>	1.06	<versions/trilha2-lora/logs/train_sft_qlora.json>	<versions/trilha2-lora/logs/train_sft_qlora.json> (RTX 4090)
<eval_trilha2_extrinsic_3seeds>	1.81	<versions/trilha2-lora/analysis/eval_trilha2_extrinsic_multiseed.json>	<versions/trilha2-lora/analysis/eval_trilha2_extrinsic_multiseed.json> (RTX 4090)

Tabela 1 – Rastreabilidade de tempo e hardware por execução (logs no repositório). Em alguns casos (e.g., `v5_main`), tempo e hardware são registrados em arquivos distintos.

Para materializar o componente econômico do "orçamento computacional", a Tabela 2 apresenta preços de referência por hora (Vast.ai, percentil 25) utilizados para estimar o custo por execução em GPU única. Esses valores servem como aproximação (os preços variam no tempo e não incluem custos indiretos como armazenamento e tempo ocioso).

GPU (1x)	Preço (USD/h, P25)
RTX 3080	$0.07
RTX 3090	$0.13
RTX 4090	$0.30

Tabela 2 – Preços de referência por hora (Vast.ai, P25) usados nas estimativas de custo.

Execução	GPU	Tempo (h)	Preço (USD/h)	Custo est. (USD)
<v1>	RTX 3080	1.1	0.07	0.08
<v2>	RTX 3080	0.9	0.07	0.06
<v3>	RTX 3080	2.0	0.07	0.14
<v4_subword>	RTX 3080	1.2	0.07	0.08
<v4_subword_ep6>	RTX 3080	3.7	0.07	0.26
<v4k_bf_ep2>	RTX 3080	1.2	0.07	0.08
<v4k_bf_lstm_ep2>	RTX 3080	1.4	0.07	0.10
<v5_main>	RTX 3090	3.2	0.13	0.42
<v5b_long>	RTX 3090	2.9	0.13	0.38
<v6_rc16k_v5b>	RTX 4090	13.3	0.30	3.99
<v6_rc16k_v5b_aligned>	RTX 4090	16.6	0.30	4.98
<cpt_qlora_llama31_8b_brwac10k>	RTX 4090	2.35	0.30	0.71
<sft_qlora_llama31_8b_instruct_canarim10k>	RTX 4090	1.06	0.30	0.32
<eval_trilha2_extrinsic_3seeds>	RTX 4090	1.81	0.30	0.54

Tabela 3 – Estimativa de custo por execução em GPU única (Vast.ai, P25), a partir dos tempos reportados nos logs. Os valores são aproximações para comparabilidade: o custo efetivamente pago pode variar conforme o preço/h da instância no momento da execução e tempos ociosos fora do job.

Componente	Tempo (h)	Custo (USD)
CPT-QLoRA (BrWaC 10k)	2.35	0.71
SFT-QLoRA (Canarim 10k)	1.06	0.32
Avaliação extrínseca (3 seeds)	1.81	0.54
Total Trilha 2 + avaliação	**5.22**	**1.57**

Tabela 4 – Custo total da Trilha 2 (treinos) e da avaliação extrínseca, em 1x RTX 4090 (Vast.ai, P25).

4.2.3 Estratégia Ótima: Fine-Tuning de Modelos Pré-Treinados

4.2.3.1 Análise Comparativa de Abordagens

Dadas as limitações identificadas, a estratégia de fine-tuning (pós-treinamento eficiente via LoRA/QLoRA) emerge como a abordagem mais viável em GPU única. A Tabela 5 resume as principais alternativas e sua viabilidade em 24 GB de VRAM.

Abordagem	VRAM (GB)	Viável?	Observação
Treinamento do zero (7B)	≈ 112	Não	requer paralelismo/otimizações (ZeRO, multi-GPU)
Continued pretraining (CPT) 8B + LoRA/QLoRA	≤ 24	Sim	texto bruto em PT-BR (BrWaC)
Supervised fine-tuning (SFT) 8B + LoRA/QLoRA	≤ 24	Sim	pares instrução–resposta (PT-BR)
QLoRA em modelos muito maiores ($\gg$ 8B)	> 24	Depende	pode exigir múltiplas GPUs e/ou offloading

Tabela 5 – Viabilidade resumida de estratégias de desenvolvimento em GPU única (RTX 4090, 24 GB).

4.2.3.2 CPT e SFT: definição e papel na adaptação

Nesta dissertação, a trilha de pós-treinamento é decomposta em duas etapas conceitualmente distintas:

- ***Continued Pretraining* (CPT):** continuação do pré-treinamento com o mesmo objetivo auto-supervisionado do modelo base (predição do próximo token), porém com dados direcionados ao idioma/domínio de interesse (e.g., amostras do BrWaC em PT-BR). O CPT busca adaptar a distribuição linguística do modelo sem impor um formato de instruções (GURURANGAN *et al.*, 2020).

- ***Supervised Fine-Tuning* (SFT):** ajuste supervisionado em pares instrução–resposta (ou formatos *chat*), visando comportamento útil e aderente a comandos do usuário. No SFT, o modelo aprende padrões de seguimento de instrução e estrutura de resposta a partir de exemplos rotulados (OUYANG *et al.*, 2022).

Do ponto de vista computacional, CPT e SFT podem ser executados com as mesmas técnicas de adaptação eficiente: **LoRA** injeta matrizes de baixo-rank em camadas alvo para treinar um pequeno subconjunto de parâmetros (HU *et al.*, 2021), enquanto **QLoRA** combina LoRA com quantização (NF4) dos pesos do modelo base para reduzir VRAM e viabilizar sequências mais longas em GPU única (DETTMERS *et al.*, 2023). Assim, a diferença central entre CPT e SFT está no *tipo de dado e objetivo experimental* (texto bruto vs. instruções), e não no mecanismo de otimização em si.

4.2.3.3 Fine-Tuning com LoRA: Análise Matemática

Low-Rank Adaptation (LoRA) reduz drasticamente os requisitos de memória ao parametrizar atualizações como matrizes de baixo rank:

$$W' = W + \Delta W = W + BA \tag{4.6}$$

onde $B \in \mathbb{R}^{d\times r}$, $A \in \mathbb{R}^{r\times k}$, e $r \ll \min(d,k)$. Intuitivamente, a atualização $\Delta W = BA$ é dita de *baixo rank* porque a dimensão interna r é pequena, o que permite representar a adaptação como o produto de duas matrizes "finas" (reduzindo parâmetros treináveis e consumo de memória).

Redução de parâmetros treináveis: Para modelo Llama 3 8B com $r = 64$:

$$\text{Parâmetros}_{originais} = 8 \times 10^9 \tag{4.7}$$

$$\text{Parâmetros}_{LoRA} = \sum_i r_i(d_i + k_i) \approx 16 \times 10^6 \tag{4.8}$$

$$\text{Redução} = \frac{8 \times 10^9}{16 \times 10^6} = 500x \tag{4.9}$$

Requisitos de memória para LoRA:

$$\text{Memória}_{\text{base}} = 8 \times 10^9 \times 2 \text{ bytes (FP16)} = 16 \text{ GB} \tag{4.10}$$

$$\text{Memória}_{\text{LoRA}} = 16 \times 10^6 \times 4 \text{ bytes (FP32)} = 64 \text{ MB} \tag{4.11}$$

$$\text{Memória}_{\text{gradientes}} = 64 \text{ MB} \tag{4.12}$$

$$\text{Memória}_{\text{otimizador}} = 128 \text{ MB} \tag{4.13}$$

$$\text{Memória}_{\text{total}} \approx 16.3 \text{ GB por GPU} \tag{4.14}$$

4.2.4 QLoRA para Modelos de Grande Escala

4.2.4.1 Quantização 4-bit com NormalFloat

QLoRA permite fine-tuning de modelos com dezenas de bilhões de parâmetros ao quantizar o modelo base para 4-bit (NF4) e treinar apenas adaptadores LoRA. O valor "70B" é usado aqui apenas como **exemplo de ordem de grandeza** (um tamanho comum em famílias abertas), e os cálculos escalam aproximadamente de forma linear com o número de parâmetros:

Redução de memória base:

$$\text{Memória}_{\text{FP16}} = 70 \times 10^9 \times 2 = 140 \text{ GB} \tag{4.15}$$

$$\text{Memória}_{\text{NF4}} = 70 \times 10^9 \times 0.5 = 35 \text{ GB} \tag{4.16}$$

$$\text{Redução} = \frac{140}{35} = 4x \tag{4.17}$$

Requisitos totais (exemplo 70B):

$$\text{Memória}_{\text{base quantizada}} = 35 \text{ GB} \tag{4.18}$$

$$\text{Memória}_{\text{adapters LoRA}} = 200 \text{ MB} \tag{4.19}$$

$$\text{Memória}_{\text{gradientes}} = 200 \text{ MB} \tag{4.20}$$

$$\text{Memória}_{\text{overhead}} = 8 \text{ GB} \tag{4.21}$$

$$\text{Memória}_{\text{total}} \approx 43.4 \text{ GB} \tag{4.22}$$

Em uma GPU única de 24 GB, QLoRA amplia significativamente a viabilidade de pós-treinamento. Ainda assim, modelos muito grandes (na faixa de dezenas de bilhões) podem exigir múltiplas GPUs e/ou *offloading*; no contexto deste trabalho, o alvo prático foi o regime 8B em 24 GB (Trilha 2), onde CPT/SFT tornam-se executáveis com baixo custo.

4.3 Metodologia de Implementação

4.3.1 Pipeline de Pré-processamento de Dados

4.3.1.1 Fonte de dados e exportação do corpus (BrWaC)

A base textual utilizada na trilha experimental (v1–v6) foi o corpus BrWaC, carregado via HuggingFace Datasets. Para treinamento do tokenizer, uma amostra embaralhada foi exportada para um arquivo de texto (`corpus_v6.txt`), após limpeza com regras de qualidade (por exemplo: remoção de linhas muito curtas e linhas com baixa proporção de caracteres alfabéticos), preservando caixa e números para manter características do português brasileiro.

4.3.1.2 Treinamento do tokenizer (SentencePiece)

SentencePiece é uma biblioteca amplamente usada para treinar tokenizers *subword* diretamente sobre texto cru, com algoritmos como *unigram* e BPE. Neste trabalho, o tokenizer foi treinado com SentencePiece no modo *unigram*, com vocabulário de 16k subpalavras, gerando o modelo (`.model`) e estatísticas auxiliares para análise de fragmentação. Essa etapa é crítica para reduzir *byte fallback* e evitar excesso de tokens curtos em palavras comuns do português.

4.3.1.3 Datasets para pós-treinamento (CPT e SFT)

Para a trilha de pós-treinamento eficiente, os datasets foram definidos para manter comparabilidade de custo:

- **CPT:** amostra de 10k documentos do BrWaC (texto bruto), com `seq_len`=2048.
- **SFT:** 10k exemplos do Canarim-Instruct PT-BR (dataset instrucional em português brasileiro, com pares instrução–resposta), com split 80/10/10 (treino/val/teste) e `seq_len`=2048.

Algoritmo 1 – Pipeline de preparação de dados e treinamento (visão geral)

1. Carregar e amostrar dados (BrWaC / Canarim) com semente fixa.
2. Limpar/normalizar textos e exportar corpus para tokenizer (quando aplicável).
3. Treinar tokenizer SentencePiece e inspecionar métricas de fragmentação.
4. Treinar o modelo (treino do zero ou LoRA/QLoRA), registrando logs e checkpoints.
5. Gerar amostras e aplicar *guardrails* para validação qualitativa.

4.3.2 Configuração do Ambiente de Fine-Tuning

4.3.2.1 Restrição prática: GPU única de consumo

O objetivo deste projeto é propor um caminho viável mesmo fora de ambientes corporativos, portanto a configuração de pós-treinamento foi desenhada para caber em uma única GPU de consumo. Neste trabalho, o ambiente de referência para os experimentos de pós-treinamento (CPT/SFT) e para o release candidate v6 é uma NVIDIA RTX 4090 (24 GB), e a técnica preferencial é LoRA/QLoRA para reduzir custo de memória e tempo.

4.3.2.2 Configuração alvo (Llama 3.1–8B + LoRA/QLoRA)

- **Modelo base**: Llama 3.1–8B (*base* para CPT; *instruct* para SFT).
- **Técnica**: LoRA/QLoRA (NF4) como padrão para permitir janelas maiores de contexto com 24 GB de VRAM.
- **Comprimento de sequência**: `seq_len`=2048.
- **Datasets**: BrWaC (CPT) e Canarim-Instruct PT-BR (SFT), ambos com subconjuntos de 10k amostras para manter comparabilidade de custo.
- **Split**: 80/10/10 (treino/val/teste) e registro de tempo total, throughput e versões de software/hardware.

4.3.2.3 Otimizações de memória e estabilidade

- **Gradient accumulation** para ajustar *batch* efetivo sem exceder VRAM.
- **Gradient checkpointing** para reduzir memória de ativações em sequências longas.
- **Checkpoints e logs** para recuperação após falhas e auditoria dos resultados.

4.3.3 Cronograma e Estimativas de Tempo

4.3.3.1 Análise Temporal Detalhada

Em vez de depender de estimativas abstratas, este trabalho prioriza tempos observados e registrados em logs. A trilha do zero (v1–v6) produz medições diretas de custo em GPU para um protótipo treinado localmente, enquanto a trilha LoRA/QLoRA é reportada com o mesmo rigor (tempo total, throughput e configuração de hardware/software), permitindo comparações justas sob a mesma restrição de orçamento computacional.

4.3.3.2 Análise de Riscos e Mitigações

Riscos técnicos identificados:

Risco	Probabilidade	Impacto	Mitigação
Falha de GPU	Média	Alto	Reinstanciar em outro host e retomar a partir de checkpoints
Overflow de memória	Baixa	Alto	Gradient checkpointing
Instabilidade de rede	Baixa	Médio	Checkpoints frequentes
Qualidade de dados	Média	Médio	Validação rigorosa

Tabela 6 – Matriz de riscos do projeto

4.4 Avaliação e resultados

4.4.1 Linha experimental (v1–v6): evolução, limitações e resultados

Esta seção consolida os resultados das versões v1–v6 do protótipo, destacando custo computacional, limitações e descobertas ao longo das iterações. Os logs e artefatos experimentais (modelos, mapeamentos, históricos de treino, amostras e relatórios) estão disponíveis no repositório do projeto, com destaque para os diretórios `analysis/` e `versions/`.

4.4.1.1 Protocolo de avaliação e comparabilidade

Ao longo do processo, a tokenização e a arquitetura mudaram (caractere → subword → Transformer), portanto as métricas numéricas entre versões não são diretamente comparáveis de forma estrita. Por esse motivo, os resultados são reportados em três camadas:

- **Avaliação padronizada por janelas (v1–v4):** métricas em um conjunto fixo de janelas para comparação dentro de cada fase (char-level e subword), registradas em <analysis/results_char_models_20k.json> e <analysis/results_subword_models_*.json>.
- **Histórico de treino (v5–v6):** melhores métricas intrínsecas de validação registradas nos logs (<versions/v5-transformer/logs/train_v5_main.json> e <versions/v6-release-candidate/logs/train_*.json>).
- **Evidência qualitativa e guardrails:** amostras geradas e heurísticas de fragmentação para capturar falhas que perda/perplexidade não penalizam adequadamente.

Loss e perplexidade (PPL): Neste trabalho, *loss* refere-se à perda de validação (cross-entropy média por token; menor é melhor). A perplexidade é reportada como PPL $= e^{loss}$; valores menores indicam maior probabilidade atribuída ao próximo token. Como a unidade "token" depende do tokenizer, loss/PPL só são diretamente comparáveis quando a tokenização é a mesma; por isso, comparações entre tokenizers distintos (e.g., v5 *vs.* v5b *vs.* v6) devem ser interpretadas com cautela.

4.4.1.2 Protocolo de avaliação (Trilha 2: CPT/SFT-QLoRA)

Enquanto a linha experimental v1–v6 prioriza avaliação intrínseca e inspeção qualitativa, a Trilha 2 (CPT/SFT via LoRA/QLoRA) requer avaliação extrínseca em tarefas, comparando

modelos base vs. pós-treinados sob o mesmo orçamento computacional. O protocolo executado é detalhado a seguir, e seus resultados consolidados estão na Tabela 13.

Objetivos e hipóteses:

- **Objetivo:** demonstrar ganho consistente do SFT em PT-BR em tarefas de QA, sumarização e reescrita, versus baseline instruccional zero-shot.
- **Hipóteses:** (i) dados PT-BR + LoRA (HU *et al.*, 2021) melhoram coerência/estilo local; (ii) CPT em PT-BR reduz fricção linguística e melhora consistência lexical; (iii) avaliação por múltiplas seeds e rubricas reduz viés de análise qualitativa.

Tarefas e conjuntos de teste:

- **QA factual curto (PT-BR):** amostra filtrada do Canarim com referências curtas (`n=200`).
- **Sumarização (PT-BR):** amostra filtrada do Canarim (`n=200`).
- **Reescrita (PT-BR):** amostra filtrada do Canarim (`n=200`).
- **Construção dos conjuntos:** os subconjuntos foram gerados em <versions/trilha2-lora/analysis/extrinsic/>, com exclusão de overlap com o subconjunto usado no SFT:
  - <versions/trilha2-lora/datasets/canarim_sft_10k_train.jsonl>
  - <versions/trilha2-lora/datasets/canarim_sft_10k_val.jsonl>
  - <versions/trilha2-lora/datasets/canarim_sft_10k_test.jsonl>

  via <versions/trilha2-lora/scripts/09_prepare_extrinsic_eval_sets.py>.

Baselines e comparações:

- **Baseline-A:** LLaMA 3.1-8B-Instruct (zero-shot).
- **Baseline-B (opcional):** LLaMA 3.1-8B + CPT-QLoRA (BrWaC 10k), reportado por métricas intrínsecas; não incluído na avaliação extrínseca por não ser um modelo instruccional.
- **Modelo alvo:** LLaMA 3.1-8B-Instruct + SFT-QLoRA (Canarim 10k).

Métricas e protocolo:

- **QA:** EM (*exact match*) e F1 (sobreposição lexical entre predição e referência), em respostas curtas; normalização de caixa/acentos.

- **Sumarização e reescrita:** ROUGE-L (F1), baseado na maior subsequência comum (*LCS*), como métrica automática comparável.
- **Inferência:** geração determinística (`do_sample=false`, `temperature=0`, `top_p=1`), com registro de throughput/custo via logs estruturados.
- **Robustez:** repetição em 3 seeds de amostragem do conjunto de teste (123/456/789), reportando média±desvio padrão.
- **Robustez/segurança (opcional):** checagens simples (alucinação, PII, prompt injection) e registro de limitações.

Tabela de resultados: Os resultados consolidados da avaliação extrínseca estão apresentados na Tabela 13. Os resumos e o agregado multi-seed estão em <versions/trilha2-lora/analysis/eval_trilha2_extrinsic_multiseed.json>; os detalhes por exemplo (saídas e métricas) estão versionados em <versions/trilha2-lora/analysis/eval_trilha2_extrinsic_details.jsonl>, <versions/trilha2-lora/analysis/eval_trilha2_extrinsic_details_seed456.jsonl> e <versions/trilha2-lora/analysis/eval_trilha2_extrinsic_details_seed789.jsonl>, gerados por <versions/trilha2-lora/scripts/10_run_extrinsic_eval.py>.

Infra, custos e reprodutibilidade:

- **Setup:** 1× NVIDIA RTX 4090 (24 GB) via Vast.ai; LoRA/QLoRA (DETTMERS *et al.*, 2023); checkpoints frequentes.
- **Métricas de eficiência:** tokens/s, GPU-h e custo estimado (USD), além de limites de VRAM/IO.
- **Reprodutibilidade:** seeds, versões (CUDA/driver/transformers/peft/bitsandbytes), scripts/-configs versionados e logs estruturados com metadados.

Execução e artefatos: A Trilha 2 foi executada no nível de treinos (CPT e SFT) e de avaliação extrínseca (QA, sumarização e reescrita) no Canarim filtrado, com resultados e artefatos versionados em <versions/trilha2-lora/>.

4.4.1.3 Resultados v1–v3 (char-level)

Versão	Arquitetura	Tempo (h)	Tokens est. (B)	Loss	PPL	Acc.
v1	CharRNN (LSTM)	1.1	13.88	1.522	4.58	55.0%
v2	CharLM (LSTM)	0.9	3.47	1.437	4.21	56.7%
v3	Stacked LSTM	2.0	3.47	1.375	3.96	58.2%

Tabela 7 – Resultados de avaliação padronizada na fase char-level (v1–v3).

Nesta fase, o objetivo foi construir o pipeline e entender limites de custo/engenharia. Apesar dos ganhos graduais em métricas, observou-se baixa qualidade textual em português (acentos e segmentação), o que motivou a mudança para subword na v4.

4.4.1.4 Resultados v4 (subword + RNN)

Execução	RNN	Vocab	Épocas	Tempo (h)	Tokens est. (B)	PPL
v4_subword	GRU	3.2k	2	1.2	2.39	75.83
v4_subword_ep6	GRU	3.2k	6	3.7	7.18	72.73
v4k_bf_ep2	GRU	4k	2	1.2	2.32	84.31
v4k_bf_lstm_ep2	LSTM	4k	2	1.4	2.32	85.37

Tabela 8 – Resultados de avaliação padronizada na fase subword + RNN (v4).

A introdução de subwords (SentencePiece) foi o principal salto qualitativo desta etapa: frases passaram a ser mais legíveis e a tokenização reduziu a necessidade de "soletrar" caracteres. Ao mesmo tempo, os experimentos com vocabulário e fallback evidenciaram um novo desafio: escolhas de tokenizer afetam fortemente a fragmentação e podem degradar a geração mesmo quando as métricas intrínsecas não parecem alarmantes.

4.4.1.5 Resultados v5–v6 (Transformer)

Execução	Tokenizer	Tempo (h)	Tokens (B)	Melhor val_loss	Melhor val_ppl	
<v5_main>	4k (+fallback)	3.2	6.00	3.8918	49.00	
<v5b_long>	unigram 12k	2.9	0.66	3.3843	29.50	
<v6_rc16k_v5b>	unigram 16k	13.3	3.76	3.3001	27.12	
<v6_rc16k_v5b_aligned>	unigram 16k	16.6	3.63	3.2935	26.94	

Tabela 9 – Resumo das execuções com Transformer (v5–v6). Métricas de perplexidade não são diretamente comparáveis entre tokenizers.

Na v5, observou-se um fenômeno central para este trabalho: mesmo com melhora clara em val_loss/val_ppl, a qualidade percebida do texto degradou (fragmentação em subwords e artefatos visuais). Isso motivou duas mudanças metodológicas: (i) revisão do tokenizer (v5b) e (ii) inclusão de *guardrails* automatizados na avaliação de amostras.

Execução	fallback_ratio	short_piece_ratio	Passa?	
<v5b_long_final>	0.000	0.050	Sim	
<v6_rc16k_v5b>	0.000	0.149	Não	
<v6_rc16k_v5b_aligned>	0.000	0.155	Não	

Tabela 10 – Guardrails de tokenização em amostras geradas (5 sementes por execução).

Observa-se uma melhora pequena em val_loss/val_ppl após o alinhamento da limpeza entre corpus do tokenizer e treino; entretanto, o short_piece_ratio permaneceu acima do limiar definido, sugerindo que a fragmentação está mais relacionada à configuração do vocabulário/tokenizer do que exclusivamente à limpeza.

4.4.1.6 Exemplos qualitativos (amostras geradas)

Para complementar as métricas, apresenta-se a seguir um recorte de amostras geradas com **prompts padronizados** (os mesmos em todas as versões). Os trechos foram truncados apenas para facilitar leitura no documento; as amostras completas estão disponíveis em:

- <analysis/samples_brwac_custom_prompts_utf8.json> (v1–v4)
- <analysis/samples_v5_main.json>
- <analysis/samples_v5b_long_final_tp075.json>
- <versions/v6-release-candidate/analysis/samples_v6_rc16k_v5b_aligned.json>

Prompt 1 (Infraestrutura).

```
PROMPT:
O setor de infraestrutura logistica brasileira debate novas concessoes
ferroviarias,
cronogramas de duplicacao, metas de produtividade e integracao com portos para
desafogar
corredores de exportacao.

v1:
a matéria e as empresas e do conteúdo de gente como o mundo a contratos e
reverter para
concentração de encontro de papel e de acordo no estado de ator com a rede de
alguns
melhores palavras do projeto do seguro de um conteúdo do direito com o povo com
 a maior de
um pacacionaliz

v2:
como ter uma tentativa de mortes de tradicionalização de uma base de modo que
se interessa a
escolha de legislatura e constituinte de uma marca de crime infecto de
transporte com a
conceitualidade. em 0 o regime de vista é que o destaque com o processo de
estado com pai a
princi

v3:
a matéria especializada pode ser desprezada com o texto ativo do projeto de
verdadeiro
```

```
conjunto de processos promovidos na consciência externa de responsabilidade do
computador do
país a considerar o então de uma apresentação de design assistir com a escola
informada. a
abordage

v4 (v4k_bf_ep2):
- o seu nome se refere à borda de uma vida em círculo, é o que se o mais alto,
para o
momento, o que quer, com o seu papel, o que o sol não se é o maior que se pode.
 o
sequestramento de cavaleiros e a cápula, o que é, no sentido de passar,, de
modo que o seu
seu papel não se transforma em seu corpo e é o seu maior valo [...]

v5 (main):
clín informações viagem claro ta vê encamatos reserv suasóveisõe obrigu concre
usar ecoên
clubeson marca pen? áre visão servidoresicidade anal 1. peso num prefeitoílias
comput
sobfeito prazo banda autom total deseist flitasir suturatados poss tema
manterbas plapenho?
serem intro? debate ag minutos_?terelhibil jovenssp [...]

v5b (final):
O setor de infraestrutura logistica brasileira debate novas concessoes
ferroviarias,
cronogramas de duplicacao, metas de produtividade e integracao com portos para
desafogar
corredores de exportacao. projeto de lei no 0/0 - de 0/0/0 - dou 0/0/0 altera o
 artigo 0 da
lei no 0/0, que dispõe sobre a concessão de parcelamen [...]

v6 (aligned):
Segundo o secretário de Segurança Pública, Francisco Teixeira, a tendência é
que o setor de
infraestrutura dê continuidade à implementação. "Na realidade, os pequenos negó
cios se
intensem em mais de um terço do mercado no futuro. Já no cenário do setor, as
esferas
produtivas podem ser divididas entre as economias exter [...]
```

Prompt 2 (Futebol).

```
PROMPT:
A diretoria de um clube de futebol negocia patrocinio master, reforcos na
janela de
transferencia, metas de bilheteria, programa de socios e auditoria das contas
apos uma
temporada irregular nos gramados.

v1:
na estrutura de conta as contratados com a importância da vida para trabalho de
 motor como a
democratica e comparação com a criança de se presente são a diretoria do amor
na agosto, no
manutenção da extra a via tomar que foram de energia ao serviço do processo em
um estado de
co

v2:
o cerca de 0 anos e composição da fila de trabalho no presidente do trabalho de
 produção e
condições de responsável pelo processo de produção de seu primeiro caso de 0 de
 maio de 0, o
ministério de resultados adversários e três anos de previsão e
proporcionalidade, a primeira
di

v3:
o contrato que está como a proposta de despesas de projetos de serviços de
instituição de
energia de energia do trabalho e da educação no presidente da cidade de polícia
 federal e
profissionais e paraná a expectativa do contrato de disposição de futebol, na
administração
do curs

v4 (v4k_bf_ep2):
em 0, de 0 de março de 0, o primeiro cnb. o que se refere ao mesmo tempo, a
pena de que o
brasil tenha sido feito no brasil. o ano passado é a melhor opção de hoje, com
a
participação de um bom nível de atuação em quadrinhos. essa é o que você sabe
```

```
se não gostar.
é muito importante para o mundo, um grande exemplo do pa [...]

v5 (main):
esfor interess passedtro imposntasconhec funçãoícios pposto fórédioch filme
trabalh minist
negócios julho gal sen próxima? altura fundam op dese pec fevereiro micro rura
melhoícia
próximaencia mostra línur?agemiadotou ouv determinvalío poderia produz sãobra
dr dela
tamanho econôm eduardo figuro facm leirações interes i [...]

v5b (final):
A diretoria de um clube de futebol negocia patrocinio master, reforcos na
janela de
transferencia, metas de bilheteria, programa de socios e auditoria das contas
apos uma
temporada irregular nos gramados. o presidente do clube, roberto marinho, disse
 que a saída
de bola foi "um pouco mais dura" e que o time pode ter um [...]

v6 (aligned):
Para os campeonatos em Fortaleza, a diretoria só tem uma semana e meio para
chegar no fim,
enquanto o mesmo está sendo negociado no final de semana. O Atlético-MG
enfrenta o Vitória
no último domingo (30), diante do Fluminense por 2 a 1. Com 16 pontos e sete
pontos, o
Vitória recebe o Vitória-BA neste sábado (31) e o F [...]
```

4.4.1.7 Limitações, desafios e descobertas

- **Métricas não garantem legibilidade:** a v5 mostrou que reduzir `val_loss`/`val_ppl` não implica, necessariamente, melhora em fluência e integridade de palavras; por isso, a avaliação passou a incluir guardrails e inspeção sistemática de amostras.
- **Tokenização é componente crítico:** a transição char-level → subword (v4) e a ampliação de vocabulário (v5b) tiveram impacto qualitativo maior do que ajustes marginais de arquitetura/hiperparâmetros em fases anteriores.
- **Limitações computacionais moldam o método:** a necessidade de operar em GPU única impôs modelos menores e janelas limitadas, reforçando a estratégia de (i) prototipagem

do zero em escala reduzida e (ii) pós-treinamento eficiente (CPT/SFT) como alternativa prática.

- **Alinhamento de limpeza tem efeito limitado:** no v6, alinhar a limpeza do corpus do tokenizer com a do treino trouxe ganho marginal nas métricas, mas não resolveu fragmentação medida pelos guardrails.

4.4.2 Trilha 2 (QLoRA): CPT e SFT em PT-BR

Nesta trilha, investiga-se pós-treinamento eficiente (LoRA/QLoRA) em um modelo base open-source de maior capacidade, com dois estágios: (i) *continued pretraining* (CPT) em português (BrWaC) e (ii) *supervised fine-tuning* (SFT) com dados instrucionais (Canarim-Instruct PT-BR). O objetivo é demonstrar ganhos mensuráveis com baixo custo, mantendo rastreabilidade por meio de logs estruturados e artefatos versionados em <versions/trilha2-lora/>.

4.4.2.1 Resumo dos treinamentos

Execução	Modelo base	Dados	Tempo (h)	Custo (USD)	Melhor `eval_loss` (PPL e^{loss})
<cpt_qlora_llama31_8b_brwac10k>	LLaMA 3.1-8B	BrWaC 10k	2.35	0.71	2.140 (8.50)
<sft_qlora_llama31_8b_instruct_canarim10k>	LLaMA 3.1-8B-Instruct	Canarim 10k	1.06	0.32	1.104 (3.02)

Tabela 11 – Resumo dos treinamentos da Trilha 2 em 1x RTX 4090 (24 GB), com `seq_len`=2048.

Os logs completos estão em <versions/trilha2-lora/logs/train_cpt_qlora.json> e <versions/trilha2-lora/logs/train_sft_qlora.json>.

4.4.2.2 Avaliação intrínseca comparável (base vs. pós-treinado)

Para reduzir o risco de interpretações equivocadas em qualitativos fora do domínio, reporta-se a seguir uma avaliação intrínseca comparável: (i) no CPT, perda/perplexidade em split de validação do BrWaC amostrado; (ii) no SFT, perda calculada apenas sobre os tokens de resposta (os tokens do prompt são mascarados no cálculo da loss) no split de validação do Canarim. A execução consolidada está em <versions/trilha2-lora/analysis/eval_trilha2_baselines.json>.

Comparação	Split de avaliação	`eval_loss`	PPL (e^{loss})	Ganho
LLaMA 3.1-8B (base)	BrWaC (val.)	2.372	10.72	–
+ CPT-QLoRA	BrWaC (val.)	2.159	8.66	−19.2% (PPL)
LLaMA 3.1-8B-Instruct (base)	Canarim (val.)	1.778	5.92	–
+ SFT-QLoRA	Canarim (val.)	1.117	3.06	−48.4% (PPL)

Tabela 12 – Avaliação intrínseca (loss e perplexidade equivalente) para base vs. adaptado em CPT e SFT.

4.4.2.3 Exemplos qualitativos (SFT-QLoRA em Canarim)

Para complementar as métricas intrínsecas, apresenta-se abaixo um recorte de exemplos do **split de validação** do Canarim, escolhidos para ilustrar casos em que o SFT melhora a aderência ao *formato esperado* (respostas curtas e diretamente avaliáveis). Durante a inspeção, observou-se ruído em parte das referências do dataset, o que reforça a importância de filtros de qualidade e curadoria na etapa de SFT.

Exemplo 1 (classificação):

```
INSTRUÇÃO:
Classifique o sexo do autor. Artigo: A melhor maneira de cozinhar um bife na
grelha.

BASE (LLaMA 3.1-8B-Instruct):
Não é possível determinar o sexo do autor com base no artigo fornecido, pois nã
o há
informações sobre a identidade do autor. O artigo parece ser um texto
informativo sobre
uma receita de cozinha, mas não fornece informações sobre a identidade do autor
.

SFT-QLoRA (Canarim 10k):
Homem
```

Exemplo 2 (classificação de impacto):

```
INSTRUÇÃO:
Tarefa: Diga-me se uma notícia é positiva ou negativa para um preço de ação da
empresa.
Notícia: O CEO da empresa está deixando sua posição.

BASE (LLaMA 3.1-8B-Instruct):
Essa notícia é negativa para o preço da ação da empresa. A saída do CEO pode
ser vista
como um sinal de instabilidade e incerteza sobre o futuro da empresa, o que
pode afetar
negativamente a confiança dos investidores e, consequentemente, o preço da ação
.

SFT-QLoRA (Canarim 10k):
Negativo
```

4.4.2.4 Avaliação extrínseca (QA, sumarização e reescrita)

Para evidência mais direta de utilidade prática, executou-se uma avaliação extrínseca em três tarefas (QA, sumarização e reescrita) usando um conjunto de teste filtrado a partir do Canarim (`n=200` por tarefa), com exclusão de overlap com o subconjunto usado no SFT (train/val/test). A geração foi determinística (`do_sample=false`, `temperature=0`) e a avaliação foi repetida em **3 seeds de amostragem** do conjunto de teste (123/456/789), reportando média±desvio padrão. Os artefatos estão em <versions/trilha2-lora/analysis/eval_trilha2_extrinsic_multiseed.json> (agregado), com detalhes por execução em <versions/trilha2-lora/analysis/eval_trilha2_extrinsic_details.jsonl>, <versions/trilha2-lora/analysis/eval_trilha2_extrinsic_details_seed456.jsonl> e <versions/trilha2-lora/analysis/eval_trilha2_extrinsic_details_seed789.jsonl>.

Tarefa	n	Métrica	Baseline	SFT-QLoRA	Ganho
QA (resposta curta)	200	EM	0.037±0.013	0.090±0.017	+0.053 (+145%)
QA (resposta curta)	200	F1	0.063±0.018	0.113±0.018	+0.050 (+80%)
Sumarização	200	ROUGE-L (F1)	0.154±0.004	0.204±0.010	+0.050 (+33%)
Reescrita	200	ROUGE-L (F1)	0.173±0.021	0.295±0.020	+0.123 (+71%)

Tabela 13 – Avaliação extrínseca em Canarim filtrado (média±desvio padrão em 3 seeds de amostragem): baseline LLaMA 3.1-8B-Instruct vs. LLaMA 3.1-8B-Instruct + SFT-QLoRA (Canarim 10k), com geração determinística. Custo de inferência estimado (somado nas 3 execuções): 1.81 GPU-h (US$ 0.54) em 1x RTX 4090.

4.4.2.5 Exemplos qualitativos (avaliação extrínseca)

Para complementar a Tabela 13, apresenta-se abaixo um recorte de três exemplos (um por tarefa) do conjunto de teste filtrado, ilustrando um padrão recorrente: após o SFT, as respostas tendem a ficar mais curtas e mais aderentes ao formato esperado pelo enunciado. Os trechos foram truncados apenas para facilitar leitura no documento; os detalhes por exemplo estão versionados em <versions/trilha2-lora/analysis/eval_trilha2_extrinsic_details.jsonl>.

Exemplo 3 (QA):

```
INSTRUÇÃO:
Eu quero comprar uma guitarra elétrica, mas eu não tenho certeza qual escolher.
 Qual devo obter? (1: Squier Bullet Stratocaster; 2: Stratocaster padrão)

REFERÊNCIA:
1

BASE (LLaMA 3.1-8B-Instruct):
Ambas as opções são excelentes escolhas, mas dependem de suas preferências e
necessidades específicas...

SFT-QLoRA (Canarim 10k):
1
```

Exemplo 4 (Sumarização):

```
INSTRUÇÃO:
Encontre um artigo de notícias sobre o governo local e escreva um resumo dele.
Artigo: O conselho da cidade de São Francisco votou para proibir a venda de
peles dentro de seus limites. A proibição entrará em vigor em janeiro de 2020,
e se aplicará a lojas de varejo e vendas on-line.

REFERÊNCIA (truncada):
O conselho da cidade de São Francisco votou a proibição da venda de peles
dentro de seus limites... (Katy Tang)

BASE (LLaMA 3.1-8B-Instruct):
Infelizmente, não consegui encontrar um artigo de notícias específico... (segue
 com contexto incorreto e extrapolações)

SFT-QLoRA (Canarim 10k):
O conselho da cidade de São Francisco votou para proibir a venda de peles
dentro de seus limites. A proibição entrará em vigor em janeiro de 2020, e se
aplicará a lojas de varejo e vendas on-line.
```

Exemplo 5 (Reescrita):

```
INSTRUÇÃO:
Descubra qual das seguintes frases é escrita em voz passiva e reescreva-a em
voz ativa: o gato foi comido pelo cachorro.

REFERÊNCIA:
O cão comeu o gato.

BASE (LLaMA 3.1-8B-Instruct):
A frase está em voz passiva... Aqui está a reescrita em voz ativa: O cachorro
comeu o gato. (com explicação adicional)

SFT-QLoRA (Canarim 10k):
O cão comeu o gato.
```

4.4.3 Métricas de Performance

4.4.3.1 Benchmarks Específicos para Português

No escopo do protótipo v1–v6, a avaliação principal foi intrínseca (perda e perplexidade em validação) e por inspeção/guardrails em amostras geradas. As métricas abaixo representam benchmarks complementares desejáveis, especialmente relevantes para a etapa de pós-treinamento via LoRA/QLoRA.

Avaliação quantitativa:

- **BLEU score**: Tradução português-inglês
- **Perplexidade**: Corpus de teste brasileiro
- **ROUGE**: Sumarização de textos em português
- **Accuracy**: Classificação de sentimentos

Benchmarks culturais específicos:

- Conhecimento sobre geografia brasileira
- Compreensão de expressões regionais
- Conhecimento histórico e cultural
- Capacidade em domínios específicos (direito, medicina)

4.4.3.2 Comparação com Modelos Existentes

Baseline models para comparação:

- GPT-3.5 (OpenAI)
- Llama 3 8B base (Meta)
- Sabiá-7B (português brasileiro)
- BERTimbau (BERT português)

Métricas de eficiência:

$$\text{Eficiência}_{computacional} = \frac{\text{Performance}}{\text{FLOPs utilizados}} \tag{4.23}$$

$$\text{Eficiência}_{dados} = \frac{\text{Performance}}{\text{Tokens de treinamento}} \tag{4.24}$$

$$\text{ROI}_{projeto} = \frac{\text{Valor gerado}}{\text{Custo computacional}} \tag{4.25}$$

4.4.4 Impacto e Aplicações

4.4.4.1 Aplicações Imediatas

Nesta dissertação, a utilidade prática é evidenciada pela avaliação extrínseca da Trilha 2 (QA, sumarização e reescrita), que mede ganho em tarefas representativas de uso real em PT-BR sob baixo custo computacional.

Potencial de impacto: Embora esteja fora do escopo deste trabalho quantificar benefícios econômicos com precisão, a motivação de soberania digital e utilidade prática se manifesta em aplicações que reduzem fricções linguísticas, aumentam produtividade e permitem maior controle sobre dados e modelos. A quantificação de impacto (e.g., ROI setorial) requer estudos dedicados e será tratada como trabalho futuro.

4.4.4.2 Contribuição para Soberania Digital

No contexto deste trabalho, a contribuição para soberania digital se materializa principalmente na viabilidade de adaptar modelos open-source ao PT-BR em hardware acessível (GPU única), aliada à rastreabilidade do pipeline (dados, scripts, logs e artefatos), o que facilita auditoria e reprodutibilidade.

4.4.4.3 Roadmap de expansão

1. **Curto prazo** (6 meses): Modelo especializado funcional
2. **Médio prazo** (1 ano): Família de modelos para diferentes domínios
3. **Longo prazo** (2 anos): Infraestrutura nacional de IA

Esta discussão reforça que, apesar das limitações de hardware, é possível produzir evidência empírica do custo de uma linha “do zero” (em escala reduzida) e estruturar um caminho pragmático de pós-treinamento eficiente (LoRA/QLoRA) sobre modelos open-source, contribuindo para a discussão de soberania digital brasileira com base técnica e rastreabilidade.

CAPÍTULO

5

CONCLUSÃO E TRABALHOS FUTUROS

5.1 Síntese dos resultados

Esta dissertação investigou, sob a perspectiva de soberania digital, o desafio prático de construir modelos de linguagem em português brasileiro sob restrições realistas de hardware. O trabalho foi estruturado em duas trilhas complementares: (i) uma linha experimental de treinamento do zero (v1–v6), em escala reduzida, para evidenciar custo e fragilidades do pipeline; e (ii) uma linha viável baseada em pós-treinamento eficiente (CPT/SFT com LoRA/QLoRA) sobre um modelo base open-source.

Na trilha v1–v6, o principal resultado é a evidência de que a dificuldade não é apenas "ter uma GPU", mas dominar um conjunto de decisões acopladas: limpeza do corpus, tokenização, definição de arquitetura, instrumentação do treinamento e critérios de avaliação. A evolução entre versões mostrou que:

- Alterações de tokenização podem produzir impactos qualitativos maiores do que melhorias marginais em perda/perplexidade, exigindo métricas complementares e inspeção sistemática de amostras.
- Métricas intrínsecas (e.g., `val_loss`/`val_ppl`) são úteis para monitorar convergência, mas não capturam, isoladamente, problemas de fragmentação e legibilidade em geração.
- Restrições de VRAM em GPU única tornam inviável treinar modelos de múltiplos bilhões de parâmetros do zero sem estratégias avançadas (paralelismo, offloading, otimizações de memória), reforçando o papel do pós-treinamento como estratégia pragmática.

Na Trilha 2, os experimentos de pós-treinamento eficiente via QLoRA apresentaram ganhos mensuráveis com baixo custo. Em particular, o SFT-QLoRA em Canarim 10k reduziu a perda intrínseca no split de validação (Tabela 12) e, na avaliação extrínseca em tarefas (QA,

sumarização e reescrita) com Canarim filtrado, apresentou melhorias consistentes versus o baseline instruccional (Tabela 13). No total, CPT+SFT consumiram 3.41 horas de GPU (US$ 1.03) e a avaliação extrínseca (3 seeds) 1.81 horas (US$ 0.54), em 1x RTX 4090.

Do ponto de vista econômico, os experimentos também ilustram que protótipos em escala reduzida podem ser executados com baixo custo direto em infraestrutura sob demanda, mas isso não se traduz automaticamente em viabilidade de escalar para modelos de fronteira. A evidência de custo apresentada no Capítulo 4 busca, sobretudo, tornar explícita a relação entre tempo de GPU, escolhas metodológicas e resultados obtidos.

5.2 Limitações

As principais limitações do estudo são:

- **Comparabilidade entre tokenizers:** perdas e perplexidades não são diretamente comparáveis quando a tokenização muda, razão pela qual a análise combinou métricas intrínsecas com exemplos qualitativos e *guardrails* de fragmentação.
- **Avaliação centrada no protótipo:** na linha v1–v6, priorizou-se rastreabilidade e aprendizado de engenharia; métricas extrínsecas e avaliação humana permanecem como próximos passos para medir utilidade prática de forma mais direta.
- **Validade externa da avaliação extrínseca:** na Trilha 2, a avaliação em tarefas foi baseada em subconjuntos filtrados do Canarim, com métricas automáticas (EM/F1 e ROUGE-L). Apesar de ter sido evitado overlap com o subconjunto usado no SFT, ainda podem existir similaridades residuais e ruído nas referências, o que limita a generalização para benchmarks externos.

5.3 Trabalhos futuros

Para consolidar a tese central de que o pós-treinamento eficiente oferece um caminho pragmaticamente viável no mesmo orçamento computacional, os próximos passos naturais são:

- **Benchmarks externos:** repetir a avaliação extrínseca em conjuntos padronizados (QA/sumarização/reescrita) para além do Canarim, com splits e rubricas fixas.
- **Robustez e variância:** avaliar múltiplas configurações de geração (seeds/temperatura) e reportar variância; complementar métricas automáticas com juiz/humano em amostras pequenas.
- **Ablations:** testar variações (por exemplo, incluir CPT-QLoRA como etapa intermediária) para isolar o efeito de CPT vs. SFT em tarefas downstream.

- **Robustez e segurança:** executar checagens simples (alucinação, PII, prompt injection) e registrar limitações.

Com isso, o documento passa a apresentar, de forma mais completa, tanto a evidência prática do custo de uma linha "do zero" quanto a evidência de uma alternativa mais eficiente para maximizar utilidade em português brasileiro sob restrições realistas de recursos.

REFERÊNCIAS

BAI, Y.; CUI, G.; LI, B.; LIN, Z.; ZHANG, Y.; ZHOU, Z.; WANG, X.; LIU, J. A comprehensive survey of llm alignment techniques: Rlhf, rlaif, ppo, dpo and more. **arXiv preprint arXiv:2307.06952**, 2023. Citado nas páginas 24 e 31.

BELTAGY, I.; PETERS, M. E.; COHAN, A. Longformer: The long-document transformer. **arXiv preprint arXiv:2004.05150**, 2020. Citado na página 81.

BENGIO, Y.; SIMARD, P.; FRASCONI, P. Learning long-term dependencies with gradient descent is difficult. **IEEE Transactions on Neural Networks**, v. 5, n. 2, p. 157–166, 1994. Citado na página 29.

BOMMASANI, R.; HUDSON, D. A.; ADELI, E.; ALTMAN, R.; ARORA, S.; ARX, S. von; BERNSTEIN, M. S.; BOHG, J.; BOSSELUT, A.; BRUNSKILL, E. *et al.* **On the Opportunities and Risks of Foundation Models**. [S.l.], 2021. ArXiv:2108.07258. Citado na página 23.

BROWN, T. B. e. a. Language models are few-shot learners. **Advances in Neural Information Processing Systems**, v. 33, p. 1877–1901, 2020. Citado na página 31.

CASWELL, I.; KREUTZER, J.; WANG, L.; WAHAB, A.; ESCH, D. van; UL-HASAN, U.; MCCARTHY, A.; ADITYA, S.; AL-RFOU, R.; ALABI, J. *et al.* Quality at a glance: An audit of web-crawled multilingual datasets. **Transactions of the Association for Computational Linguistics**, v. 9, p. 1086–1102, 2021. Disponível em: <https://doi.org/10.1162/tacl_a_00411>. Citado na página 32.

CHO, K. e. a. Learning phrase representations using rnn encoder-decoder for statistical machine translation. **arXiv preprint arXiv:1406.1078**, 2014. Citado na página 29.

DAO, T.; FU, D. Y.; ERMON, S.; RUDRA, A.; RÉ, C. Flashattention: Fast and memory-efficient exact attention with io-awareness. **Advances in Neural Information Processing Systems**, v. 35, p. 16344–16359, 2022. Citado na página 79.

DETTMERS, T.; PAGNONI, A.; HOLTZMAN, A.; ZETTLEMOYER, L. Qlora: Efficient finetuning of quantized llms. **Advances in Neural Information Processing Systems**, v. 36, 2023. Citado nas páginas 32, 36, 43, 49, 108 e 119.

DEVLIN, J.; CHANG, M.-W.; LEE, K.; TOUTANOVA, K. Bert: Pre-training of deep bidirectional transformers for language understanding. **arXiv preprint arXiv:1810.04805**, 2018. Citado na página 31.

GURURANGAN, S.; MARASOVIC, A.; SWAYAMDIPTA, S.; LO, K.; BELTAGY, I.; DOWNEY, D.; SMITH, N. A. Don't stop pretraining: Adapt language models to domains and tasks. In: **Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (ACL)**. [S.l.: s.n.], 2020. Citado nas páginas 31, 43 e 119.

HOCHREITER, S.; SCHMIDHUBER, J. Long short-term memory. **Neural Computation**, v. 9, n. 8, p. 1735–1780, 1997. Citado na página 29.

HOFFMANN, J.; BORGEAUD, S.; MENSCH, A.; BUCHATSKAYA, E.; CAI, T.; RUTHERFORD, E.; CASAS, D. d. L.; HENDRICKS, L. A.; WELBL, J.; CLARK, A. *et al.* Training compute-optimal large language models. **arXiv preprint arXiv:2203.15556**, 2022. Citado nas páginas 36, 37 e 110.

HU, E. J.; SHEN, Y.; WALLIS, P.; ALLEN-ZHU, Z.; LI, Y.; WANG, S.; WANG, L.; CHEN, W. Lora: Low-rank adaptation of large language models. **arXiv preprint arXiv:2106.09685**, 2021. Citado nas páginas 31, 36, 43 e 48.

JOACHIMS, T. Text categorization with support vector machines: Learning with many relevant features. In: **European Conference on Machine Learning**. [S.l.: s.n.], 1998. p. 137–142. Citado na página 28.

JURAFSKY, D.; MARTIN, J. H. **Speech and Language Processing**. 3rd ed. draft. ed. [S.l.]: Pearson, 2023. Citado nas páginas 27, 28 e 29.

KAPLAN, J.; MCCANDLISH, S.; HENIGHAN, T.; BROWN, T. B.; CHESS, B.; CHILD, R.; GRAY, S.; RADFORD, A.; WU, J.; AMODEI, D. Scaling laws for neural language models. **arXiv preprint arXiv:2001.08361**, 2020. Citado nas páginas 36, 37 e 109.

KIM, Y. Convolutional neural networks for sentence classification. In: **EMNLP 2014**. [S.l.: s.n.], 2014. Citado na página 29.

KWON, W.; LI, Z.; ZHUANG, S.; SHENG, Y.; ZHENG, L.; YU, C. H.; GONZALEZ, J. E.; ZHANG, H.; STOICA, I. Efficient memory management for large language model serving with pagedattention. **Proceedings of the 29th Symposium on Operating Systems Principles**, p. 611–626, 2023. Citado na página 104.

LECUN, Y.; BENGIO, Y.; HINTON, G. Deep learning. **Nature**, v. 521, n. 7553, p. 436–444, 2015. Citado na página 29.

MIKOLOV, T. e. a. Efficient estimation of word representations in vector space. In: **ICLR 2013**. [S.l.: s.n.], 2013. Citado na página 29.

OUYANG, L.; WU, J.; JIANG, X.; ALMEIDA, D.; WAINWRIGHT, C. *et al.* Training language models to follow instructions with human feedback. **arXiv preprint arXiv:2203.02155**, 2022. Citado nas páginas 31, 43 e 119.

PATTERSON, D. *et al.* **The Carbon Footprint of Machine Learning Training**. 2023. Technical report. Citado na página 24.

PENNINGTON, J.; SOCHER, R.; MANNING, C. D. Glove: Global vectors for word representation. In: **EMNLP 2014**. [S.l.: s.n.], 2014. Citado na página 29.

Qwen Team. **Qwen2.5: A Family of Strong and Scalable Language Models**. [S.l.]: Alibaba Cloud, 2024. <https://qwenlm.github.io/blog/qwen2.5/>. Citado nas páginas 24 e 32.

RADFORD, A. e. a. Improving language understanding by generative pre-training. **OpenAI Blog**, 2018. Citado na página 31.

RAFFEL, C. e. a. Exploring the limits of transfer learning with a unified text-to-text transformer. **Journal of Machine Learning Research**, v. 21, n. 140, p. 1–67, 2020. Citado na página 31.

RAJBHANDARI, S.; RASLEY, J.; RUWASE, O.; HE, Y. Zero: Memory optimizations toward training trillion parameter models. **Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis**, p. 1–16, 2020. Citado na página 96.

ROSS, E.; CONIA, S.; NAVIGLI, R. Lost in translation: An analysis of the cultural and linguistic biases of large language models. **arXiv preprint arXiv:2310.13137**, 2023. Citado na página 23.

SALTON, G.; BUCKLEY, C. Term-weighting approaches in automatic text retrieval. **Information Processing & Management**, v. 24, n. 5, p. 513–523, 1988. Citado na página 28.

SEBASTIANI, F. Machine learning in automated text categorization. **ACM Computing Surveys**, v. 34, n. 1, p. 1–47, 2002. Citado na página 28.

TOUVRON, H.; LAVRIL, T.; IZACARD, G.; MARTINET, X.; LACHAUX, M.-A.; LACROIX, T.; ROZIÈRE, B.; GOYAL, N.; HAMBRO, E.; AZHAR, F. *et al.* Llama: Open and efficient foundation language models. **arXiv preprint arXiv:2302.13971**, 2023. Citado nas páginas 24 e 32.

TURING, A. M. Computing machinery and intelligence. **Mind**, v. 59, n. 236, p. 433–460, 1950. Citado na página 28.

VASWANI, A.; SHAZEER, N.; PARMAR, N.; USZKOREIT, J.; JONES, L.; GOMEZ, A. N.; KAISER, L.; POLOSUKHIN, I. Attention is all you need. **Advances in Neural Information Processing Systems**, v. 30, p. 5998–6008, 2017. Citado nas páginas 24, 30, 35 e 71.

WEI, J.; TAY, Y.; BOMMASANI, R.; RAFFEL, C.; ZOPH, B.; BORGEAUD, S.; YOGATAMA, D.; BOSMA, M.; ZHOU, D.; METZLER, D. *et al.* Emergent abilities of large language models. **Transactions on Machine Learning Research**, 2022. Citado na página 113.

WEIZENBAUM, J. ELIZA— a computer program for the study of natural language communication between man and machine. **Communications of the ACM**, v. 9, n. 1, p. 36–45, 1966. Citado na página 27.

WINOGRAD, T. **Understanding Natural Language**. New York: Academic Press, 1972. Citado na página 27.

YUN, C.; BHOJANAPALLI, S.; RAWAT, A. S.; REDDI, S. J.; KUMAR, S. Are transformers universal approximators of sequence-to-sequence functions? In: **International Conference on Learning Representations**. [S.l.: s.n.], 2020. ICLR; earlier version as arXiv preprint. Citado na página 72.

ZAHEER, M.; GURUGANESH, G.; DUBEY, K. A.; AINSLIE, J.; ALBERTI, C.; ONTANON, S.; PHAM, P.; RAVULA, A.; WANG, Q.; YANG, L. *et al.* Big bird: Transformers for longer sequences. **Advances in Neural Information Processing Systems**, v. 33, p. 17283–17297, 2020. Citado na página 81.

GLOSSÁRIO

Framework: é uma abstração que une códigos comuns entre vários projetos de *software* provendo uma funcionalidade genérica. *Frameworks* são projetados com a intenção de facilitar o desenvolvimento de *software*, habilitando designers e programadores a gastarem mais tempo determinando as exigências do *software* do que com detalhes de baixo nível do sistema.

Padrões de projeto: ou *Design Pattern*, descreve uma solução geral reutilizável para um problema recorrente no desenvolvimento de sistemas de *software* orientados a objetos. Não é um código final, é uma descrição ou modelo de como resolver o problema do qual trata, que pode ser usada em muitas situações diferentes.

Template: é um documento sem conteúdo, com apenas a apresentação visual (apenas cabeçalhos por exemplo) e instruções sobre onde e qual tipo de conteúdo deve entrar a cada parcela da apresentação.

Web: Sinônimo mais conhecido de *World Wide Web* (WWW). É a interface gráfica da Internet que torna os serviços disponíveis totalmente transparentes para o usuário e ainda possibilita a manipulação multimídia da informação.

WYSIWYG: "What You See Is What You Get" ou "O que você vê é o que você obtém". Recurso tem por objetivo permitir que um documento, enquanto manipulado na tela, tenha a mesma aparência de sua utilização, usualmente sendo considerada final. Isso facilita para o desenvolvedor que pode trabalhar visualizando a aparência do documento sem precisar salvar em vários momentos e abrir em um *software* separado de visualização.

APÊNDICE

A

TRANSFORMER E ATENÇÃO (MATERIAL COMPLEMENTAR)

A.1 Arquitetura Transformer: Fundamentos Matemáticos e Computacionais

A arquitetura Transformer, introduzida por (VASWANI *et al.*, 2017), representa uma ruptura paradigmática no processamento sequencial de dados, substituindo a recorrência por mecanismos de atenção paralelos. Esta seção apresenta uma análise matemática rigorosa dos componentes fundamentais e suas implicações computacionais.

A.1.1 Formalização Matemática do Mecanismo de Atenção

A.1.1.1 Definição Formal e Propriedades

O mecanismo de atenção pode ser formalizado como uma função que mapeia uma consulta (*query*) e um conjunto de pares chave-valor (*key-value*) para uma saída. Matematicamente, dado um conjunto de consultas $Q \in \mathbb{R}^{n \times d_k}$, chaves $K \in \mathbb{R}^{m \times d_k}$ e valores $V \in \mathbb{R}^{m \times d_v}$, a função de atenção é definida como:

$$\text{Attention}(Q,K,V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{A.1}$$

onde n é o comprimento da sequência de consulta, m é o comprimento da sequência de chave-valor, d_k é a dimensão das chaves e consultas, e d_v é a dimensão dos valores.

Propriedades matemáticas fundamentais:

1. Invariância por permutação: A função de atenção é equivariante por permutação

nas posições de entrada, ou seja:

$$\text{Attention}(Q\Pi, K\Pi, V\Pi) = \text{Attention}(Q, K, V)\Pi \tag{A.2}$$

onde Π é uma matriz de permutação.

2. Normalização probabilística: A matriz de atenção $A = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)$ satisfaz:

$$\sum_{j=1}^{m} A_{ij} = 1, \quad A_{ij} \geq 0, \quad \forall i \in [1, n] \tag{A.3}$$

3. Aproximação universal: Para qualquer função contínua $f : \mathbb{R}^{d_v} \to \mathbb{R}^{d_v}$, existe uma configuração de atenção que pode aproximar f arbitrariamente bem (YUN *et al.*, 2020).

A.1.1.2 Análise do Fator de Escala

O fator de escala $\frac{1}{\sqrt{d_k}}$ é crucial para estabilidade numérica. Sem este fator, o produto escalar QK^T pode produzir valores com magnitude $O(\sqrt{d_k})$, empurrando a função softmax para regiões onde os gradientes são extremamente pequenos.

Análise estatística: Se q e k são vetores aleatórios com componentes independentes $q_i, k_i \sim \mathcal{N}(0, 1)$, então:

$$\mathbb{E}[q \cdot k] = 0 \tag{A.4}$$

$$\text{Var}(q \cdot k) = \mathbb{E}\left[\left(\sum_{i=1}^{d_k} q_i k_i\right)^2\right] = \sum_{i=1}^{d_k} \mathbb{E}[q_i^2 k_i^2] = d_k \tag{A.5}$$

O fator de escala normaliza esta variância para 1, mantendo a distribuição dos scores de atenção em uma faixa estável para a função softmax.

Algoritmo 2 – Scaled Dot-Product Attention

1. **Entrada**: $Q \in \mathbb{R}^{n \times d_k}, K \in \mathbb{R}^{m \times d_k}, V \in \mathbb{R}^{m \times d_v}$
2. scores $\leftarrow QK^T / \sqrt{d_k}$ // Complexidade: $O(nmd_k)$
3. weights $\leftarrow$ softmax(scores) // Complexidade: $O(nm)$
4. output $\leftarrow$ weights $\cdot V$ // Complexidade: $O(nmd_v)$
5. **Retorna**: output $\in \mathbb{R}^{n \times d_v}$

A.1.2 Multi-Head Attention: Paralelização de Representações

A.1.2.1 Formulação Matemática

O mecanismo de Multi-Head Attention permite que o modelo atenda a informações de diferentes subespaços de representação simultaneamente. Formalmente, é definido como:

$$\text{MultiHead}(Q,K,V) = \text{Concat}(\text{head}_1,\ldots,\text{head}_h)W^O \tag{A.6}$$

$$\text{onde head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V) \tag{A.7}$$

onde as matrizes de projeção são $W_i^Q \in \mathbb{R}^{d_{model}\times d_k}$, $W_i^K \in \mathbb{R}^{d_{model}\times d_k}$, $W_i^V \in \mathbb{R}^{d_{model}\times d_v}$ e $W^O \in \mathbb{R}^{hd_v\times d_{model}}$.

Escolha de dimensões: Tipicamente, define-se $d_k = d_v = d_{model}/h$, onde h é o número de cabeças de atenção. Esta escolha:

- Mantém o custo computacional total comparável ao de uma única cabeça
- Permite especialização de cada cabeça em diferentes tipos de dependências
- Preserva a capacidade representacional total do modelo

Análise de capacidade representacional: O espaço de funções representáveis por Multi-Head Attention é:

$$\mathscr{F}_{MHA} = \left\{ f : \mathbb{R}^{n\times d} \to \mathbb{R}^{n\times d} \mid f(X) = \sum_{i=1}^{h} A_i(X)XW_i^V W^O \right\} \tag{A.8}$$

onde $A_i(X)$ são as matrizes de atenção para cada cabeça.

Algoritmo 3 – Multi-Head Attention

1. **Entrada**: $Q,K,V \in \mathbb{R}^{n\times d_{model}}$, número de cabeças h
2. $d_k \leftarrow d_v \leftarrow d_{model}/h$
3. Para $i = 1$ até h:
   a) $Q_i \leftarrow Q \cdot W_i^Q$ // Projeção da query
   b) $K_i \leftarrow K \cdot W_i^K$ // Projeção da key
   c) $V_i \leftarrow V \cdot W_i^V$ // Projeção do value
   d) $\text{head}_i \leftarrow \text{Attention}(Q_i, K_i, V_i)$
4. $\text{concat} \leftarrow \text{Concatenate}(\text{head}_1,\ldots,\text{head}_h)$
5. $\text{output} \leftarrow \text{concat} \cdot W^O$
6. **Retorna**: $\text{output} \in \mathbb{R}^{n\times d_{model}}$

A.1.3 Análise de Complexidade Computacional

A.1.3.1 Complexidade Temporal e Espacial

A complexidade computacional do mecanismo de atenção é um fator crítico para a escalabilidade dos Transformers. Para uma sequência de comprimento n e dimensão do modelo d, temos:

Complexidade temporal por camada:

$$T_{attention} = O(n^2 d + nd^2) \tag{A.9}$$

$$T_{feedforward} = O(nd^2) \tag{A.10}$$

$$T_{total} = O(n^2 d + nd^2) \tag{A.11}$$

Complexidade espacial:

$$S_{attention\_matrix} = O(n^2) \tag{A.12}$$

$$S_{activations} = O(nd) \tag{A.13}$$

$$S_{total} = O(n^2 + nd) \tag{A.14}$$

Operações de ponto flutuante detalhadas: Para uma camada Transformer completa:

$$\text{FLOPs}_{QKV\_proj} = 3nd^2 \tag{A.15}$$

$$\text{FLOPs}_{attention\_scores} = n^2 d \tag{A.16}$$

$$\text{FLOPs}_{attention\_output} = n^2 d \tag{A.17}$$

$$\text{FLOPs}_{output\_proj} = nd^2 \tag{A.18}$$

$$\text{FLOPs}_{feedforward} = 8nd^2 \text{ (assumindo } d_{ff} = 4d) \tag{A.19}$$

$$\text{FLOPs}_{total} = 12nd^2 + 2n^2 d \tag{A.20}$$

A.1.3.2 Análise de Escalabilidade

Esta complexidade quadrática em relação ao comprimento da sequência representa o principal gargalo computacional dos Transformers. Para sequências longas:

Ponto de crossover: O termo $n^2 d$ domina quando $n > 6d$. Para modelos típicos com $d = 512$, isso ocorre para $n > 3072$ tokens.

Limitações de memória: Para sequências de comprimento $n = 8192$ e $d = 1024$:

$$\text{Memória}_{attention} = 8192^2 \times 4 \text{ bytes} = 268 \text{ MB por cabeça} \tag{A.21}$$

A.1.4 Paralelização e Eficiência Computacional

A.1.4.1 Vantagens da Paralelização

A principal vantagem computacional dos Transformers reside na capacidade de paralelização. Enquanto RNNs processam sequências de forma inerentemente sequencial ($h_t = f(h_{t-1}, x_t)$), o Transformer pode computar todas as posições simultaneamente.

Análise comparativa de paralelização:

RNN:

$$\text{Passos sequenciais} = n \tag{A.22}$$

$$\text{Operações por passo} = O(d^2) \tag{A.23}$$

$$\text{Paralelização máxima} = O(d^2) \text{ operações simultâneas} \tag{A.24}$$

Transformer:

$$\text{Passos sequenciais} = O(1) \tag{A.25}$$

$$\text{Operações por passo} = O(n^2 d + n d^2) \tag{A.26}$$

$$\text{Paralelização máxima} = O(n^2 d) \text{ operações simultâneas} \tag{A.27}$$

Eficiência em hardware moderno: Em GPUs com P cores de processamento, o speedup teórico é:

$$\text{Speedup}_{Transformer} = \min\left(P, \frac{n^2 d + n d^2}{d^2}\right) = \min\left(P, \frac{n^2}{d} + n\right) \tag{A.28}$$

Para $n \gg d$, o speedup aproxima-se de $\min(P, n^2/d)$, que pode ser substancial.

A.1.5 Gradientes e Estabilidade de Treinamento

A.1.5.1 Problema do Desvanecimento de Gradientes em RNNs

Um dos problemas fundamentais das RNNs é o desvanecimento ou explosão de gradientes durante o treinamento. Para uma RNN com T passos temporais, o gradiente em relação aos parâmetros no tempo t é:

$$\frac{\partial L}{\partial \theta_t} = \frac{\partial L}{\partial h_T} \prod_{k=t+1}^{T} \frac{\partial h_k}{\partial h_{k-1}} \frac{\partial h_t}{\partial \theta_t} \tag{A.29}$$

O produto $\prod_{k=t+1}^{T} \frac{\partial h_k}{\partial h_{k-1}}$ pode decair exponencialmente (desvanecimento) ou crescer exponencialmente (explosão), dependendo dos autovalores da matriz de transição.

Análise quantitativa: Se λ_{max} é o maior autovalor de $\frac{\partial h_k}{\partial h_{k-1}}$, então:

$$\left\| \prod_{k=t+1}^{T} \frac{\partial h_k}{\partial h_{k-1}} \right\| \leq \lambda_{max}^{T-t} \tag{A.30}$$

Para $\lambda_{max} < 1$, o gradiente decai exponencialmente com a distância temporal.

A.1.5.2 Solução dos Transformers

Os Transformers evitam este problema através de:

1. Conexões residuais: O gradiente flui diretamente através das conexões residuais:

$$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial y}\left(I + \frac{\partial F(x)}{\partial x}\right) \tag{A.31}$$

onde I garante um caminho direto para o gradiente.

2. Mecanismo de atenção: Cria caminhos diretos entre todas as posições da sequência, com gradientes que não dependem da distância temporal:

$$\frac{\partial \text{Attention}(Q,K,V)}{\partial V_j} = \sum_{i=1}^{n} A_{ij} \frac{\partial L}{\partial \text{output}_i} \tag{A.32}$$

onde A_{ij} são os pesos de atenção, independentes da distância $|i-j|$.

A.1.6 Embeddings Posicionais e Representação Temporal

A.1.6.1 Necessidade de Informação Posicional

Como os Transformers processam sequências em paralelo, eles não possuem noção inerente de ordem. Os embeddings posicionais são adicionados aos embeddings de entrada para injetar informação sobre a posição relativa ou absoluta dos tokens.

Embeddings posicionais sinusoidais: Os embeddings posicionais sinusoidais originais são definidos como:

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d_{model}}}\right) \tag{A.33}$$

$$PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{2i/d_{model}}}\right) \tag{A.34}$$

onde pos é a posição e i é a dimensão.

Propriedade de linearidade: Esta formulação permite que o modelo aprenda a atender a posições relativas, pois para qualquer deslocamento fixo k:

$$PE_{pos+k} = \mathbf{M}_k \cdot PE_{pos} \tag{A.35}$$

onde $\mathbf{M}_k$ é uma matriz de transformação linear que depende apenas de k.

Algoritmo 4 – Geração de Embeddings Posicionais

1. **Entrada**: comprimento máximo L, dimensão d_{model}
2. Inicializar $PE \in \mathbb{R}^{L \times d_{model}}$
3. Para $pos = 0$ até $L-1$:
   a) Para $i = 0$ até $d_{model}/2 - 1$:
      i. angle $\leftarrow pos/10000^{2i/d_{model}}$
      ii. $PE[pos, 2i] \leftarrow \sin(\text{angle})$
      iii. $PE[pos, 2i+1] \leftarrow \cos(\text{angle})$
4. **Retorna**: PE

A.1.6.2 Embeddings Posicionais Aprendidos

Alternativamente, os embeddings posicionais podem ser aprendidos durante o treinamento:

$$PE_{learned} \in \mathbb{R}^{L_{max} \times d_{model}} \tag{A.36}$$

onde L_{max} é o comprimento máximo de sequência suportado.

Vantagens e desvantagens:

- **Sinusoidais**: Generalizam para sequências mais longas que as vistas no treinamento
- **Aprendidos**: Podem capturar padrões posicionais específicos da tarefa, mas limitados ao comprimento de treinamento

Esta análise rigorosa dos fundamentos matemáticos dos Transformers fornece a base teórica necessária para compreender as otimizações e variações arquiteturais discutidas nas seções subsequentes.

A.2 Mecanismo de Atenção: Implementação e Otimizações

O mecanismo de atenção constitui o núcleo computacional dos Transformers, mas sua implementação naive apresenta limitações severas de escalabilidade. Esta seção analisa rigorosamente as otimizações fundamentais que tornam viável o treinamento de LLMs modernos.

A.2.1 Análise de Complexidade da Implementação Naive

A.2.1.1 Gargalos Computacionais e de Memória

A implementação naive da atenção requer materializar a matriz de atenção completa $A \in \mathbb{R}^{n \times n}$, resultando em complexidade espacial $O(n^2)$. Para sequências longas, isso rapidamente se torna proibitivo.

Análise quantitativa de memória: Para um modelo com n tokens, h cabeças de atenção e precisão FP16:

$$\text{Memória}_{attention\_matrix} = h \times n^2 \times 2 \text{ bytes} \tag{A.37}$$

$$\text{Memória}_{activations} = h \times n \times d_k \times 2 \text{ bytes} \tag{A.38}$$

$$\text{Memória}_{total} = 2h(n^2 + nd_k) \text{ bytes} \tag{A.39}$$

Para $n = 8192, h = 32, d_k = 128$:

$$\text{Memória}_{total} = 2 \times 32 \times (8192^2 + 8192 \times 128) = 4.3 \text{ GB} \tag{A.40}$$

Algoritmo 5 – Atenção Padrão (Baseline)

1. **Entrada**: $Q, K, V \in \mathbb{R}^{n \times d}$
2. $S \leftarrow QK^T/\sqrt{d}$ // $O(n^2 d)$ FLOPs, $O(n^2)$ memória
3. $P \leftarrow \text{softmax}(S)$ // $O(n^2)$ FLOPs
4. $O \leftarrow PV$ // $O(n^2 d)$ FLOPs
5. **Retorna**: O
6. **Complexidade**: Tempo $O(n^2 d)$, Memória $O(n^2 + nd)$

A.2.1.2 Análise de Bandwidth e Throughput

Em GPUs modernas, o gargalo frequentemente é bandwidth de memória, não capacidade computacional. Para A100 com 1.6 TB/s de bandwidth HBM:

Tempo de transferência de dados:

$$T_{memory} = \frac{\text{Dados transferidos}}{\text{Bandwidth}} \tag{A.41}$$

$$= \frac{2h(n^2 + nd_k) \text{ bytes}}{1.6 \times 10^{12} \text{ bytes/s}} \tag{A.42}$$

Para o exemplo anterior: $T_{memory} = 2.7$ ms apenas para transferência de dados.

Tempo computacional:

$$T_{compute} = \frac{\text{FLOPs}}{\text{Peak throughput}} \tag{A.43}$$

$$= \frac{2n^2d + n^2}{312 \times 10^{12} \text{ FLOPs/s}} \approx 0.9 \text{ ms} \tag{A.44}$$

Como $T_{memory} > T_{compute}$, o algoritmo é memory-bound, não compute-bound.

A.2.2 Flash Attention: Otimização Fundamental

A.2.2.1 Fundamentos Teóricos

Flash Attention (DAO *et al.*, 2022) reformula o cálculo da atenção para reduzir drasticamente o uso de memória, aproveitando a hierarquia de memória das GPUs modernas. A ideia central é computar a atenção em blocos, mantendo apenas estatísticas necessárias na memória rápida (SRAM).

Propriedade fundamental do softmax incremental: O softmax pode ser computado de forma incremental usando a seguinte identidade:

$$\text{softmax}([x_1, x_2]) = \left[\frac{e^{x_1 - m}}{e^{x_1 - m} + e^{x_2 - m}}, \frac{e^{x_2 - m}}{e^{x_1 - m} + e^{x_2 - m}} \right] \tag{A.45}$$

onde $m = \max(x_1, x_2)$ para estabilidade numérica.

Generalização para blocos: Para dois blocos de scores $S^{(1)}, S^{(2)}$:

$$m^{new} = \max(m^{(1)}, m^{(2)}) \tag{A.46}$$

$$\ell^{new} = e^{m^{(1)} - m^{new}} \ell^{(1)} + e^{m^{(2)} - m^{new}} \ell^{(2)} \tag{A.47}$$

$$O^{new} = \frac{e^{m^{(1)} - m^{new}} \ell^{(1)} O^{(1)} + e^{m^{(2)} - m^{new}} \ell^{(2)} O^{(2)}}{\ell^{new}} \tag{A.48}$$

onde $m^{(i)}$ é o máximo por linha, $\ell^{(i)}$ é a soma exponencial, e $O^{(i)}$ é a saída parcial.

A.2.2.2 Algoritmo Flash Attention Detalhado

A.2.2.3 Análise de Complexidade do Flash Attention

Complexidade de memória:

$$\text{SRAM usage} = O(B_r d + B_c d + B_r B_c) \tag{A.49}$$

$$\text{HBM usage} = O(nd) \quad \text{(apenas para } Q, K, V, O\text{)} \tag{A.50}$$

Escolhendo $B_r = B_c = \sqrt{M/4d}$ onde M é o tamanho da SRAM:

$$\text{Redução de memória} = \frac{n^2}{M/4d} = \frac{4dn^2}{M} \tag{A.51}$$

Algoritmo 6 – Flash Attention

1. **Parâmetros**: Tamanhos de bloco B_r, B_c baseados na capacidade SRAM
2. **Entrada**: $Q, K, V \in \mathbb{R}^{n \times d}$
3. Dividir Q em $T_r = \lceil n/B_r \rceil$ blocos: $Q_1, \ldots, Q_{T_r}$
4. Dividir K, V em $T_c = \lceil n/B_c \rceil$ blocos: $K_1, V_1, \ldots, K_{T_c}, V_{T_c}$
5. Inicializar $O = (0)_{n \times d} \in \mathbb{R}^{n \times d}$
6. Inicializar $\ell = (0)_n, m = (-\infty)_n \in \mathbb{R}^n$
7. Para $i = 1$ até T_r:
   a) Carregar $Q_i \in \mathbb{R}^{B_r \times d}$ para SRAM
   b) Inicializar $O_i = (0)_{B_r \times d}, \ell_i = (0)_{B_r}, m_i = (-\infty)_{B_r}$
   c) Para $j = 1$ até T_c:
      i. Carregar $K_j, V_j \in \mathbb{R}^{B_c \times d}$ para SRAM
      ii. $S_{ij} = Q_i K_j^T / \sqrt{d_k} \in \mathbb{R}^{B_r \times B_c}$
      iii. $\tilde{m}_{ij} = \text{rowmax}(S_{ij}) \in \mathbb{R}^{B_r}$
      iv. $\tilde{P}_{ij} = \exp(S_{ij} - \tilde{m}_{ij}) \in \mathbb{R}^{B_r \times B_c}$
      v. $\tilde{\ell}_{ij} = \text{rowsum}(\tilde{P}_{ij}) \in \mathbb{R}^{B_r}$
      vi. $m_i^{new} = \max(m_i, \tilde{m}_{ij})$
      vii. $\ell_i^{new} = e^{m_i - m_i^{new}} \ell_i + e^{\tilde{m}_{ij} - m_i^{new}} \tilde{\ell}_{ij}$
      viii. $O_i \leftarrow \frac{e^{m_i - m_i^{new}} \ell_i O_i + e^{\tilde{m}_{ij} - m_i^{new}} \tilde{P}_{ij} V_j}{\ell_i^{new}}$
      ix. $\ell_i \leftarrow \ell_i^{new}, m_i \leftarrow m_i^{new}$
   d) Escrever O_i para HBM
8. **Retorna**: O

Para A100 com $M = 40$ MB SRAM, $d = 128, n = 8192$:

$$\text{Redução} = \frac{4 \times 128 \times 8192^2}{40 \times 10^6} \approx 69x \tag{A.52}$$

Complexidade de I/O:

$$\text{HBM reads} = O(nd \cdot T_c) = O\left(\frac{n^2 d^2}{M}\right) \tag{A.53}$$

$$\text{HBM writes} = O(nd) \tag{A.54}$$

Comparado com $O(n^2)$ da implementação naive, Flash Attention reduz I/O por fator $O(M/(nd))$.

A.2.3 Padrões de Atenção Esparsa

A.2.3.1 Motivação Teórica

Para sequências extremamente longas, mesmo Flash Attention pode ser insuficiente. Padrões de atenção esparsa reduzem a complexidade computacional limitando quais posições podem atender umas às outras, baseando-se na observação empírica de que muitas dependências são locais ou seguem padrões estruturados.

Análise de esparsidade natural: Em modelos treinados, a matriz de atenção frequentemente exibe padrões esparsos naturais. Para uma matriz de atenção A, definimos esparsidade como:

$$\text{Sparsity}(A) = \frac{\text{número de elementos } < \varepsilon}{\text{número total de elementos}} \tag{A.55}$$

Estudos empíricos mostram que $\text{Sparsity}(A) > 0.8$ para $\varepsilon = 0.01$ em muitas tarefas.

A.2.3.2 Longformer: Atenção Local + Global

Longformer (BELTAGY; PETERS; COHAN, 2020) combina atenção local (janela deslizante) com atenção global (tokens especiais):

$$A_{ij} = \begin{cases} \text{atenção completa} & \text{se } j \in \mathcal{G}(i) \\ \text{atenção local} & \text{se } |i-j| \leq w/2 \\ 0 & \text{caso contrário} \end{cases} \tag{A.56}$$

onde $\mathcal{G}(i)$ são os tokens globais e w é o tamanho da janela local.

Complexidade do Longformer:

$$\text{Tokens locais} : O(nw) \tag{A.57}$$

$$\text{Tokens globais} : O(ng) \text{ onde } g = |\mathcal{G}| \tag{A.58}$$

$$\text{Total} : O(n(w+g)) \tag{A.59}$$

Para $w \ll n$ e $g \ll n$, isso resulta em complexidade linear $O(n)$.

A.2.3.3 BigBird: Padrão Híbrido Estruturado

BigBird (ZAHEER *et al.*, 2020) utiliza um padrão que combina três tipos de atenção:

1. Atenção global: g tokens especiais atendem a todos os outros **2. Atenção local:** Janela deslizante de tamanho w **3. Atenção aleatória:** r conexões aleatórias por token

Propriedades teóricas do BigBird:

- **Conectividade**: O grafo de atenção é conectado com alta probabilidade

Algoritmo 7 – BigBird Attention Pattern

1. **Entrada**: sequência de comprimento n, parâmetros w, g, r
2. Inicializar matriz de atenção esparsa $A \in \{0,1\}^{n \times n}$
3. Para cada token i:
   a) **Atenção local**: $A_{i,j} = 1$ para $j \in [i - w/2, i + w/2]$
   b) **Atenção global**: Se $i \in \mathscr{G}$, então $A_{i,j} = 1$ para todo j
   c) **Atenção aleatória**: Selecionar r índices aleatórios $\mathscr{R}_i$, $A_{i,j} = 1$ para $j \in \mathscr{R}_i$
4. **Retorna**: Padrão de atenção A

- **Diâmetro**: Diâmetro esperado é $O(\log n)$
- **Expressividade**: Pode aproximar atenção completa com erro limitado

Teorema (Aproximação BigBird): Para qualquer sequência de entrada, existe uma configuração BigBird tal que:

$$\|O_{BigBird} - O_{Full}\|_F \leq \varepsilon \tag{A.60}$$

com $w = O(\log n), g = O(\sqrt{n}), r = O(\log n)$.

A.2.4 Otimizações de Precisão Numérica

A.2.4.1 Mixed Precision Training

Mixed Precision Training utiliza precisão de 16 bits (FP16) para a maioria dos cálculos, mantendo 32 bits (FP32) apenas onde necessário para estabilidade numérica.

Análise de range numérico:

$$\text{FP16 range} : [6 \times 10^{-8}, 6.5 \times 10^{4}] \tag{A.61}$$

$$\text{FP32 range} : [1.4 \times 10^{-45}, 3.4 \times 10^{38}] \tag{A.62}$$

Para o mecanismo de atenção, a sequência de precisão é crítica:

Análise de estabilidade numérica: O softmax é particularmente sensível a overflow. Para scores s_i, o softmax naive pode overflow se $\max_i s_i > 88$ (limite FP32). A implementação estável usa:

$$\text{softmax}(s_i) = \frac{e^{s_i - s_{max}}}{\sum_j e^{s_j - s_{max}}} \tag{A.63}$$

Algoritmo 8 – Mixed Precision Attention

1. Computar $S = QK^T/\sqrt{d_k}$ em FP16
2. Converter S para FP32 antes do softmax
3. $P = \text{softmax}(S)$ em FP32 // Evita overflow/underflow
4. Converter P para FP16
5. $O = PV$ em FP16
6. **Retorna**: O em FP16

A.2.4.2 Gradient Scaling

Gradient scaling é usado para evitar underflow de gradientes em FP16:

$$\text{loss\_scaled} = \text{loss} \times S \tag{A.64}$$

onde S é o fator de escala, tipicamente $S = 2^{16}$.

Algoritmo 9 – Dynamic Gradient Scaling

1. Inicializar $S = 2^{16}$, contador de overflow $C = 0$
2. Para cada iteração:
   a) loss_scaled $=$ loss $\times S$
   b) grads_scaled $=$ backward(loss_scaled)
   c) Se overflow detectado:
      i. $S \leftarrow S/2$, $C \leftarrow 0$
      ii. Pular atualização de parâmetros
   d) Senão:
      i. grads $=$ grads_scaled$/S$
      ii. Atualizar parâmetros com grads
      iii. $C \leftarrow C + 1$
      iv. Se $C > 2000$: $S \leftarrow \min(S \times 2, 2^{16})$, $C \leftarrow 0$

Análise de convergência: O gradient scaling não afeta a convergência teórica, pois:

$$\frac{\partial(\text{loss} \times S)}{\partial \theta} = S \times \frac{\partial \text{loss}}{\partial \theta} \tag{A.65}$$

A divisão por S restaura a magnitude original dos gradientes.

A.2.5 Análise Comparativa de Performance

Tabela de complexidades:

Método	Tempo	Memória	Qualidade
Atenção Padrão	$O(n^2d)$	$O(n^2)$	100%
Flash Attention	$O(n^2d)$	$O(n)$	100%
Longformer	$O(nwd)$	$O(nw)$	$\sim$95%
BigBird	$O(n(w+g+r)d)$	$O(n(w+g+r))$	$\sim$98%

Tabela 14 – Comparação de complexidades para diferentes implementações de atenção

Esta análise rigorosa das otimizações do mecanismo de atenção demonstra como inovações algorítmicas fundamentais tornaram viável o treinamento de LLMs modernos com sequências longas e modelos de grande escala.

APÊNDICE

B

COMPUTE E INFRAESTRUTURA (MATERIAL COMPLEMENTAR)

B.1 Requisitos Computacionais e Arquitetura de Hardware

O treinamento de LLMs modernos demanda recursos computacionais massivos, exigindo análise rigorosa de requisitos de hardware, padrões de acesso à memória e otimizações arquiteturais. Esta seção apresenta uma análise quantitativa detalhada dos gargalos computacionais e suas soluções.

B.1.1 Análise Rigorosa de Complexidade Computacional

B.1.1.1 Decomposição Detalhada de FLOPs

O número de operações de ponto flutuante (FLOPs) é uma métrica fundamental para estimar requisitos computacionais. Para um Transformer com L camadas, h cabeças de atenção, dimensão do modelo d, dimensão do feed-forward d_{ff}, vocabulário V, e sequência de comprimento n:

Embeddings de entrada:

$$\text{FLOPs}_{embed} = 2Vd + 2nd \quad \text{(lookup + posicional)} \tag{B.1}$$

Por camada de atenção (detalhado):

$$\text{FLOPs}_{QKV\_proj} = 3 \times 2nd^2 = 6nd^2 \tag{B.2}$$

$$\text{FLOPs}_{attention\_scores} = 2n^2d \tag{B.3}$$

$$\text{FLOPs}_{attention\_weights} = 2n^2d \tag{B.4}$$

$$\text{FLOPs}_{output\_proj} = 2nd^2 \tag{B.5}$$

$$\text{FLOPs}_{attn\_total} = 8nd^2 + 4n^2d \tag{B.6}$$

Por camada feed-forward:

$$\text{FLOPs}_{ff\_up} = 2nd \cdot d_{ff} \tag{B.7}$$

$$\text{FLOPs}_{ff\_down} = 2nd_{ff} \cdot d \tag{B.8}$$

$$\text{FLOPs}_{ff\_total} = 4nd \cdot d_{ff} \tag{B.9}$$

Layer normalization (2 por camada):

$$\text{FLOPs}_{layernorm} = 2 \times 5nd = 10nd \tag{B.10}$$

Cabeça de saída (language modeling):

$$\text{FLOPs}_{output} = 2ndV \tag{B.11}$$

Total por forward pass:

$$\text{FLOPs}_{forward} = 2Vd + 2nd + L(8nd^2 + 4n^2d + 4nd_{ff}d + 10nd) + 2ndV \tag{B.12}$$

$$= 2nd(V/n + 1 + V/n) + L \cdot nd(8d + 4n + 4d_{ff} + 10) \tag{B.13}$$

$$\approx L \cdot nd(8d + 4n + 4d_{ff}) \quad \text{para } n, d \gg 1 \tag{B.14}$$

Para $d_{ff} = 4d$ (configuração padrão):

$$\text{FLOPs}_{forward} \approx L \cdot nd(8d + 4n + 16d) = L \cdot nd(24d + 4n) \tag{B.15}$$

Treinamento completo: Considerando backward pass (2x forward) e overhead de otimizador (1.2x):

$$\text{FLOPs}_{training} = 1.2 \times 3 \times L \cdot nd(24d + 4n) = 3.6L \cdot nd(24d + 4n) \tag{B.16}$$

Aproximação para modelos grandes: Para $n \ll d$ (típico em LLMs):

$$\text{FLOPs}_{training} \approx 3.6L \cdot 24nd^2 = 86.4Lnd^2 \tag{B.17}$$

Usando a relação $N \approx 12Ld^2$ (número de parâmetros):

$$\text{FLOPs}_{training} \approx 6ND \quad \text{(regra de Chinchilla)} \tag{B.18}$$

Algoritmo 10 – Cálculo de FLOPs para Modelo Arbitrário

1. **Entrada**: L, d, h, d_{ff}, n, V, D (tokens de treinamento)
2. params $\leftarrow 12Ld^2 + Vd$ // Aproximação de parâmetros
3. flops_per_token $\leftarrow 6 \times$ params
4. total_flops $\leftarrow$ flops_per_token $\times D$
5. **Retorna**: total_flops

B.1.1.2 Análise de Casos Específicos

Exemplo - GPT-3 (175B parâmetros):

$$L = 96, \quad d = 12288, \quad h = 96, \quad d_{ff} = 49152 \tag{B.19}$$

$$n = 2048, \quad V = 50257, \quad D = 300 \times 10^9 \tag{B.20}$$

$$\text{FLOPs per token} = 6 \times 175 \times 10^9 = 1.05 \times 10^{12} \tag{B.21}$$

$$\text{Total FLOPs} = 1.05 \times 10^{12} \times 300 \times 10^9 = 3.15 \times 10^{23} \tag{B.22}$$

Tempo de treinamento em A100:

$$t_{training} = \frac{3.15 \times 10^{23}}{312 \times 10^{12} \times 0.4} = 2.52 \times 10^6 \text{ segundos} = 29 \text{ dias} \tag{B.23}$$

onde 0.4 é a eficiência típica (40

B.1.2 Análise de Hierarquia de Memória e Bandwidth

B.1.2.1 Modelo de Performance Memory-Bound

O desempenho de LLMs é frequentemente limitado pela largura de banda de memória, não pela capacidade computacional. A hierarquia típica de memória em GPUs modernas apresenta trade-offs fundamentais entre capacidade, bandwidth e latência.

Hierarquia de memória quantificada:

Intensidade aritmética e roofline model: Para uma operação ser compute-bound em vez de memory-bound, deve satisfazer:

$$\text{Intensidade aritmética} = \frac{\text{FLOPs}}{\text{Bytes transferidos}} > \frac{\text{Peak FLOPS}}{\text{Peak Bandwidth}} \tag{B.24}$$

Para A100: $\frac{312 \times 10^{12}}{1.5 \times 10^{12}} = 208$ FLOP/byte.

Nível	Capacidade	Bandwidth	Latência	Energia/bit
Registradores	256 KB	20 TB/s	1 ciclo	0.1 pJ
Shared/L1	128 KB	10 TB/s	10 ciclos	1 pJ
L2 Cache	40 MB	3 TB/s	100 ciclos	10 pJ
HBM	40-80 GB	1.5-3.35 TB/s	300 ciclos	100 pJ

Tabela 15 – Hierarquia de memória em GPUs modernas (A100/H100)

Análise de operações típicas:

1. Multiplicação matriz-matriz (GEMM): Para $C = AB$ onde $A \in \mathbb{R}^{m\times k}, B \in \mathbb{R}^{k\times n}$:

$$\text{FLOPs} = 2mkn \tag{B.25}$$

$$\text{Bytes} = (mk + kn + mn) \times 4 \text{ (FP32)} \tag{B.26}$$

$$\text{Intensidade} = \frac{2mkn}{4(mk + kn + mn)} \tag{B.27}$$

Para matrizes quadradas $m = k = n$:

$$\text{Intensidade} = \frac{2n^3}{12n^2} = \frac{n}{6} \tag{B.28}$$

Para ser compute-bound: $n > 6 \times 208 = 1248$.

2. Atenção (naive):

$$\text{FLOPs} = 4n^2 d \tag{B.29}$$

$$\text{Bytes} = (3nd + n^2 + nd) \times 2 = 2(4nd + n^2) \tag{B.30}$$

$$\text{Intensidade} = \frac{4n^2 d}{2(4nd + n^2)} = \frac{2nd}{4d + n} \tag{B.31}$$

Para $d = 1024, n = 2048$: Intensidade = 341 FLOP/byte (compute-bound). Para $d = 128, n = 8192$: Intensidade = 128 FLOP/byte (memory-bound).

B.1.2.2 Otimizações de Acesso à Memória

Análise de eficiência:

$$\text{Reuso de dados} = \frac{T^3}{3T^2} = \frac{T}{3} \tag{B.32}$$

$$\text{Redução de bandwidth} = \frac{T}{3} \text{ vezes} \tag{B.33}$$

Para $T = 128$: Redução de 42x no tráfego de memória.

Algoritmo 11 – Tiled Matrix Multiplication

1. **Parâmetros**: Tamanho do tile T baseado na capacidade de shared memory
2. **Entrada**: Matrizes $A \in \mathbb{R}^{m \times k}, B \in \mathbb{R}^{k \times n}$
3. Para $i = 0$ até $m-1$ (passo T):
   a) Para $j = 0$ até $n-1$ (passo T):
      i. Inicializar $C_{tile} = 0$
      ii. Para $l = 0$ até $k-1$ (passo T):
         A. Carregar $A_{tile}[i : i+T, l : l+T]$ para shared memory
         B. Carregar $B_{tile}[l : l+T, j : j+T]$ para shared memory
         C. $C_{tile} \leftarrow C_{tile} + A_{tile} \times B_{tile}$
      iii. Escrever C_{tile} para memória global

B.1.3 Arquiteturas de Hardware Especializadas

B.1.3.1 Análise Comparativa de Aceleradores

NVIDIA A100/H100 - Tensor Cores: Tensor Cores executam operações $D = A \times B + C$ em precisão mista:

$$\text{Throughput}_{A100} = 312 \text{ TFLOPS (FP16)} \tag{B.34}$$

$$\text{Throughput}_{H100} = 989 \text{ TFLOPS (FP16)} \tag{B.35}$$

Eficiência energética:

$$\text{Energia}_{A100} = 400W \Rightarrow 0.78 \text{ TFLOPS/W} \tag{B.36}$$

$$\text{Energia}_{H100} = 700W \Rightarrow 1.41 \text{ TFLOPS/W} \tag{B.37}$$

Google TPU v4 - Systolic Arrays: Arquitetura otimizada para multiplicação de matrizes com dataflow determinístico:

$$\text{Dimensão do array} = 128 \times 128 \tag{B.38}$$

$$\text{Throughput} = 275 \text{ TFLOPS (BF16)} \tag{B.39}$$

$$\text{Eficiência energética} = 1.15 \text{ TFLOPS/W} \tag{B.40}$$

Comparação quantitativa:

Métrica de custo-efetividade:

$$\text{Cost-effectiveness} = \frac{\text{TFLOPS}}{\text{Price per hour}} \tag{B.41}$$

Hardware	TFLOPS	Memória	BW (TB/s)	FLOPS/W	$/hora
A100 40GB	312	40 GB	1.5	0.78	$3.06
A100 80GB	312	80 GB	2.0	0.78	$4.10
H100	989	80 GB	3.35	1.41	$8.00
TPU v4	275	32 GB	1.2	1.15	$2.40

Tabela 16 – Comparação de aceleradores para LLMs (preços cloud 2024)

$$\text{A100-40GB} : \frac{312}{3.06} = 102 \text{ TFLOPS/\$} \tag{B.42}$$

$$\text{H100} : \frac{989}{8.00} = 124 \text{ TFLOPS/\$} \tag{B.43}$$

$$\text{TPU v4} : \frac{275}{2.40} = 115 \text{ TFLOPS/\$} \tag{B.44}$$

B.1.3.2 Análise de Precision Formats

Formatos numéricos e trade-offs:

Formato	Bits	Range	Precisão	Speedup
FP32	32	$\pm 3.4 \times 10^{38}$	2^{-23}	1x
FP16	16	$\pm 6.5 \times 10^{4}$	2^{-10}	2x
BF16	16	$\pm 3.4 \times 10^{38}$	2^{-7}	2x
INT8	8	± 127	1	4x

Tabela 17 – Formatos numéricos e características

Análise de estabilidade numérica: Valores muito pequenos podem *underflow* em FP16 (isto é, serem arredondados para zero por estarem abaixo do menor valor representável). Tomando como referência o menor *subnormal* em FP16 ($g_{\min} \approx 6 \times 10^{-8}$), e modelando um gradiente escalar g como $g \sim \mathcal{N}(0, \sigma^2)$, a probabilidade de underflow pode ser aproximada por:

$$P(\text{underflow}) = P(|g| < g_{\min}) = 2\Phi\left(\frac{g_{\min}}{\sigma}\right) - 1 \tag{B.45}$$

onde $\Phi(\cdot)$ é a função de distribuição acumulada da Normal padrão.

Para $\sigma = 10^{-4}$, tem-se $g_{\min}/\sigma = 6 \times 10^{-4}$ e $P(\text{underflow}) \approx 4.8 \times 10^{-4}$ (0.048%). O ponto prático é que FP16 tem **menor faixa dinâmica** que BF16/FP32; por isso, em *mixed precision* é comum manter acumulação/estados críticos em FP32 e usar *loss scaling* para reduzir o risco de *underflow* e instabilidade.

B.1.4 Interconexão e Comunicação Distribuída

B.1.4.1 Análise de Bandwidth de Comunicação

Para treinamento distribuído, a largura de banda de comunicação entre nós é crítica. Durante o treinamento, gradientes devem ser sincronizados através de operações AllReduce.

Embora os experimentos deste trabalho tenham sido executados em GPU única (Capítulo 4), esta subseção é mantida como referencial para contextualizar as exigências de escala e os gargalos de comunicação em ambientes multi-nó.

Modelo de comunicação: Para P workers e modelo com M parâmetros:

$$\text{Dados por AllReduce} = M \times \text{sizeof(param)} \tag{B.46}$$

$$\text{Tempo ideal} = \frac{2M \times \text{sizeof(param)} \times (P-1)}{P \times \text{Bandwidth}} \tag{B.47}$$

Tecnologias de interconexão:

Tecnologia	Bandwidth	Latência	Alcance	Custo/Gb/s
NVLink 4.0	900 GB/s	1 μs	Intra-nó	\$0.1
InfiniBand HDR	200 Gb/s	0.6 μs	Inter-nó	\$5
Ethernet 400G	400 Gb/s	2 μs	Inter-nó	\$2

Tabela 18 – Tecnologias de interconexão para clusters de treinamento

Algoritmo 12 – Análise de Eficiência de Comunicação

1. **Entrada**: M (parâmetros), P (workers), BW (bandwidth), C (compute time)
2. $T_{comm} \leftarrow \frac{2M \times 4 \times (P-1)}{P \times BW}$ // Tempo de comunicação
3. $T_{total} \leftarrow \max(C, T_{comm})$ // Tempo total (overlap assumido)
4. Efficiency $\leftarrow \frac{C}{T_{total}}$ // Eficiência de paralelização
5. **Retorna**: Efficiency

Exemplo - GPT-3 em 8 nós A100:

$$M = 175 \times 10^9 \text{ parâmetros} \tag{B.48}$$

$$\text{Dados} = 175 \times 10^9 \times 4 = 700 \text{ GB} \tag{B.49}$$

$$T_{comm} = \frac{2 \times 700 \times 7}{8 \times 25} = 49 \text{ segundos (InfiniBand)} \tag{B.50}$$

$$T_{compute} = \frac{\text{FLOPs per batch}}{8 \times 312 \times 10^{12} \times 0.4} \approx 120 \text{ segundos} \tag{B.51}$$

Eficiência: $\frac{120}{120} = 100\%$ (communication-hidden), pois $T_{comm} < T_{compute}$ e assume-se *overlap* perfeito (`T_total = max(C, T_comm)`). Na prática, o overlap pode não ser perfeito e a eficiência real tende a ser menor.

B.1.4.2 Topologias de Rede Otimizadas

Fat-tree topology: Para k-ary fat-tree com $N = (k/2)^3$ nós:

$$\text{Bisection bandwidth} = \frac{k^3}{4} \times \text{link bandwidth} \tag{B.52}$$

$$\text{Diâmetro} = 6 \text{ hops máximo} \tag{B.53}$$

Dragonfly topology: Otimizada para tráfego all-to-all:

$$\text{Grupos} = g \tag{B.54}$$

$$\text{Routers por grupo} = a \tag{B.55}$$

$$\text{Nós por router} = p \tag{B.56}$$

$$\text{Total de nós} = gap \tag{B.57}$$

Análise de congestionamento: Para padrão de tráfego uniforme, a probabilidade de congestionamento é:

$$P(\text{congestion}) = 1 - \left(1 - \frac{\lambda}{C}\right)^N \tag{B.58}$$

onde λ é a taxa de chegada e C é a capacidade do link.

Esta análise rigorosa dos requisitos computacionais e arquiteturas de hardware fornece as bases quantitativas necessárias para o design eficiente de sistemas de treinamento de LLMs.

B.2 Infraestrutura Distribuída e Paralelização

O treinamento de LLMs modernos pode requerer infraestrutura distribuída sofisticada para lidar com modelos que excedem a capacidade de memória de uma única GPU. Esta seção apresenta uma análise rigorosa das estratégias de paralelização e suas implicações para eficiência computacional. Embora os experimentos deste trabalho tenham sido executados majoritariamente em GPU única (Capítulo 4), a discussão a seguir contextualiza as técnicas necessárias para escalar o treinamento além desse cenário.

B.2.1 Taxonomia de Paralelização

B.2.1.1 Classificação Fundamental

As estratégias de paralelização para LLMs podem ser categorizadas em três dimensões principais:

1. Paralelização de Dados (Data Parallelism): Cada worker processa um subconjunto diferente dos dados de treinamento, mantendo uma cópia completa do modelo.

2. Paralelização de Modelo (Model Parallelism): O modelo é dividido entre múltiplos workers, com cada worker responsável por uma parte dos parâmetros.

3. Paralelização de Pipeline (Pipeline Parallelism): O modelo é dividido em estágios sequenciais, com cada worker processando um estágio diferente.

Análise de complexidade de comunicação: Para P workers, modelo com M parâmetros, e batch size B:

Estratégia	Memória/Worker	Comunicação	Eficiência
Data Parallel	$O(M)$	$O(M)$	$O(P)$
Model Parallel	$O(M/P)$	$O(B \cdot d)$	$O(1)$
Pipeline Parallel	$O(M/P)$	$O(B \cdot d)$	$O(P)$

Tabela 19 – Comparação de estratégias de paralelização

B.2.2 Data Parallelism: Análise Matemática

B.2.2.1 Synchronous Data Parallelism

No data parallelism síncrono, cada worker computa gradientes em um subconjunto dos dados, seguido por uma operação AllReduce para sincronizar gradientes.

Algoritmo matemático: Para iteração t, worker i:

$$g_i^{(t)} = \nabla_\theta \mathscr{L}(\theta^{(t-1)}, \mathscr{B}_i^{(t)}) \tag{B.59}$$

$$\bar{g}^{(t)} = \frac{1}{P} \sum_{i=1}^{P} g_i^{(t)} \tag{B.60}$$

$$\theta^{(t)} = \theta^{(t-1)} - \eta \bar{g}^{(t)} \tag{B.61}$$

onde $\mathscr{B}_i^{(t)}$ é o mini-batch do worker i na iteração t.

Algoritmo 13 – Synchronous Data Parallelism

1. **Inicialização**: Cada worker inicializa com os mesmos parâmetros θ_0
2. Para cada iteração t:
   a) **Forward pass**: Cada worker i computa loss em $\mathscr{B}_i^{(t)}$
   b) **Backward pass**: Cada worker computa gradientes locais $g_i^{(t)}$
   c) **AllReduce**: Sincronizar gradientes: $\bar{g}^{(t)} = \frac{1}{P} \sum_{i=1}^{P} g_i^{(t)}$
   d) **Update**: Cada worker atualiza: $\theta^{(t)} = \theta^{(t-1)} - \eta \bar{g}^{(t)}$
3. **Retorna**: Modelo treinado $\theta^{(T)}$

Análise de convergência: Sob condições padrão (função convexa, gradientes limitados), o data parallelism mantém as mesmas garantias de convergência do SGD sequencial:

$$\mathbb{E}[\|\nabla f(\theta^{(t)})\|^2] \leq \frac{2(f(\theta^{(0)}) - f^*)}{\eta t} + \eta L \sigma^2 \tag{B.62}$$

onde L é a constante de Lipschitz e σ^2 é a variância dos gradientes.

B.2.2.2 Asynchronous Data Parallelism

No data parallelism assíncrono, workers atualizam parâmetros independentemente, sem sincronização global.

Parameter Server Architecture:

$$\text{Worker } i: \quad \theta_i^{(t)} = \text{pull}(\text{PS}) \tag{B.63}$$

$$g_i^{(t)} = \nabla_\theta \mathscr{L}(\theta_i^{(t)}, \mathscr{B}_i^{(t)}) \tag{B.64}$$

$$\text{push}(g_i^{(t)}, \text{PS}) \tag{B.65}$$

Parameter Server:

$$\theta^{(t+1)} = \theta^{(t)} - \eta g_i^{(t)} \quad \text{(quando recebe } g_i^{(t)}\text{)} \tag{B.66}$$

Análise de staleness: O delay entre workers introduz staleness τ. A convergência é garantida se:

$$\eta < \frac{1}{L(1+\tau)} \tag{B.67}$$

B.2.3 Model Parallelism: Decomposição Matemática

B.2.3.1 Tensor Parallelism

Tensor parallelism divide operações de álgebra linear entre múltiplos devices. Para uma multiplicação matriz-matriz $Y = XW$:

Column-wise partitioning:

$$W = [W_1, W_2, \ldots, W_P] \tag{B.68}$$

$$Y = X[W_1, W_2, \ldots, W_P] = [XW_1, XW_2, \ldots, XW_P] \tag{B.69}$$

Row-wise partitioning:

$$W = \begin{bmatrix} W_1 \\ W_2 \\ \vdots \\ W_P \end{bmatrix} \tag{B.70}$$

$$Y = X \begin{bmatrix} W_1 \\ W_2 \\ \vdots \\ W_P \end{bmatrix} = XW_1 + XW_2 + \ldots + XW_P \tag{B.71}$$

Análise de comunicação: Para tensor de dimensão $d \times d$ dividido entre P devices:

$$\text{Computation per device} : O\left(\frac{d^2}{P}\right) \tag{B.72}$$

$$\text{Communication volume} : O(d^2) \text{ (AllReduce)} \tag{B.73}$$

$$\text{Communication/Computation ratio} : O(P) \tag{B.74}$$

B.2.3.2 Megatron-LM: Tensor Parallelism para Transformers

MLP Layer Parallelization: Para MLP com duas camadas lineares:

$$Y = \text{GELU}(XW_1)W_2 \tag{B.75}$$

$$= \text{GELU}(X[W_1^{(1)}, W_1^{(2)}, \ldots, W_1^{(P)}])[W_2^{(1)}; W_2^{(2)}; \ldots; W_2^{(P)}] \tag{B.76}$$

Self-Attention Parallelization: Para multi-head attention com h cabeças:

$$\text{head}_i = \text{Attention}(XW_i^Q, XW_i^K, XW_i^V) \tag{B.77}$$

$$\text{Output} = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)W^O \tag{B.78}$$

Dividindo cabeças entre devices:

$$\text{Device } p : \text{heads}_{(p-1)h/P+1:ph/P} \tag{B.79}$$

B.2.4 Pipeline Parallelism: Análise Temporal

B.2.4.1 Naive Pipeline Parallelism

Pipeline parallelism divide o modelo em P estágios sequenciais. Para um modelo com L camadas:

$$\text{Stage } p : \text{Layers } \left\lfloor \frac{(p-1)L}{P} \right\rfloor + 1 : \left\lfloor \frac{pL}{P} \right\rfloor \tag{B.80}$$

Análise de bubble time: O tempo total para processar um batch é:

$$T_{total} = (P-1)T_{stage} + T_{stage} = PT_{stage} \tag{B.81}$$

O tempo de bubble (idle) é:

$$T_{bubble} = (P-1)T_{stage} \tag{B.82}$$

A eficiência é:

$$\text{Efficiency} = \frac{T_{stage}}{PT_{stage}} = \frac{1}{P} \tag{B.83}$$

B.2.4.2 Micro-batching para Pipeline Parallelism

Para melhorar eficiência, o batch é dividido em M micro-batches:

Algoritmo 14 – Pipeline Parallelism com Micro-batching

1. **Entrada**: Batch $\mathscr{B}$, número de micro-batches M, número de estágios P
2. Dividir $\mathscr{B}$ em micro-batches: $\{\mathscr{B}_1, \mathscr{B}_2, \ldots, \mathscr{B}_M\}$
3. Para cada micro-batch $m = 1, \ldots, M$:
   a) **Forward pass**: $\mathscr{B}_m$ flui através dos estágios $1 \to 2 \to \ldots \to P$
   b) **Backward pass**: Gradientes fluem $P \to P-1 \to \ldots \to 1$
4. **Gradient accumulation**: Acumular gradientes de todos os micro-batches
5. **Parameter update**: Atualizar parâmetros usando gradientes acumulados

Análise de eficiência com micro-batching:

$$\text{Efficiency} = \frac{M}{M+P-1} \tag{B.84}$$

Para $M \gg P$, a eficiência aproxima-se de 100%.

Trade-off memória vs. eficiência:

$$\text{Memory per stage} = \frac{M \cdot \text{activation\_size}}{P} \tag{B.85}$$

$$\text{Peak memory} = M \cdot \text{activation\_size\_per\_stage} \tag{B.86}$$

B.2.5 ZeRO: Zero Redundancy Optimizer

ZeRO (*Zero Redundancy Optimizer*) é uma família de técnicas de otimização de memória para treinamento distribuído de modelos grandes (RAJBHANDARI *et al.*, 2020). Em *data parallelism* padrão, cada GPU mantém uma cópia completa dos parâmetros, gradientes e estados

do otimizador; essa duplicação entre P workers é a "redundância" de memória. Ao particionar esses estados entre GPUs, ZeRO reduz a VRAM necessária por GPU e viabiliza modelos maiores. Nesta dissertação, ZeRO é citado apenas como referência conceitual (os experimentos executados ocorreram em GPU única).

B.2.5.1 Análise de Redundância de Memória

Durante treinamento com Adam, cada parâmetro requer:

- Parâmetro: 4 bytes (FP32)
- Gradiente: 4 bytes (FP32)
- Momentum: 4 bytes (FP32)
- Variance: 4 bytes (FP32)

Total: 16 bytes por parâmetro.

Para modelo com M parâmetros e P workers em data parallelism:

$$\text{Total memory} = P \times 16M \text{ bytes} \tag{B.87}$$

ZeRO-1: Optimizer State Partitioning

$$\text{Memory per worker} = 4M + \frac{8M}{P} \text{ bytes} \tag{B.88}$$

ZeRO-2: Gradient Partitioning

$$\text{Memory per worker} = 4M + \frac{12M}{P} \text{ bytes} \tag{B.89}$$

ZeRO-3: Parameter Partitioning

$$\text{Memory per worker} = \frac{16M}{P} \text{ bytes} \tag{B.90}$$

B.2.5.2 ZeRO-3 Implementation

Análise de comunicação ZeRO-3:

$$\text{AllGather volume} = 2 \times \frac{M}{P} \times P = 2M \tag{B.91}$$

$$\text{ReduceScatter volume} = 2 \times \frac{M}{P} \times P = 2M \tag{B.92}$$

$$\text{Total communication} = 4M \text{ bytes per iteration} \tag{B.93}$$

Comparado com data parallelism padrão (2M bytes), ZeRO-3 tem 2x overhead de comunicação, mas reduz memória por fator P.

Algoritmo 15 – ZeRO-3 Implementation

1. **Inicialização**: Particionar parâmetros entre workers
2. Para cada forward pass:
   a) Para cada camada l:
      i. **AllGather**: Coletar parâmetros θ_l de todos os workers
      ii. **Compute**: Executar forward pass da camada l
      iii. **Discard**: Descartar parâmetros não-locais de θ_l
3. Para cada backward pass:
   a) Para cada camada l (ordem reversa):
      i. **AllGather**: Coletar parâmetros θ_l
      ii. **Compute**: Executar backward pass da camada l
      iii. **ReduceScatter**: Reduzir e distribuir gradientes
      iv. **Discard**: Descartar parâmetros não-locais
4. **Update**: Cada worker atualiza apenas seus parâmetros locais

B.2.6 Gradient Accumulation e Mixed Precision

B.2.6.1 Gradient Accumulation Matemático

Para simular batch size grande B com memória limitada, usa-se gradient accumulation:

$$\text{Effective batch} = \bigcup_{i=1}^{K} \mathscr{B}_i \text{ onde } |\mathscr{B}_i| = \frac{B}{K} \tag{B.94}$$

$$\bar{g} = \frac{1}{B} \sum_{x \in \text{Effective batch}} \nabla_\theta \ell(\theta, x) \tag{B.95}$$

$$= \frac{1}{K} \sum_{i=1}^{K} \frac{1}{|\mathscr{B}_i|} \sum_{x \in \mathscr{B}_i} \nabla_\theta \ell(\theta, x) \tag{B.96}$$

B.2.7 Algoritmos de Comunicação Coletiva

B.2.7.1 AllReduce: Ring Algorithm

O algoritmo Ring AllReduce é fundamental para data parallelism eficiente:

Algoritmo 16 – Gradient Accumulation com Mixed Precision

1. **Inicialização**: $g_{acc} = 0$ (FP32), scale factor S
2. Para $i = 1$ até K:
   a) **Forward**: $\text{loss}_i = \text{model}(\mathscr{B}_i)$ (FP16)
   b) **Scale**: $\text{loss}_{scaled} = S \times \text{loss}_i$
   c) **Backward**: $g_i = \nabla \text{loss}_{scaled}$ (FP16)
   d) **Unscale**: $g_i = g_i / S$
   e) **Accumulate**: $g_{acc} += g_i$ (FP32)
3. **Average**: $g_{acc} = g_{acc}/K$
4. **Update**: $\theta = \theta - \eta g_{acc}$

Algoritmo 17 – Ring AllReduce

1. **Setup**: Organizar P workers em anel lógico
2. **Scatter-Reduce Phase**:
   a) Dividir dados em P chunks
   b) Para $i = 1$ até $P-1$:
      i. Cada worker envia chunk $(rank - i)$ mod P para próximo worker
      ii. Cada worker reduz chunk recebido com chunk local
3. **AllGather Phase**:
   a) Para $i = 1$ até $P-1$:
      i. Cada worker envia chunk $(rank - i + 1)$ mod P para próximo worker
      ii. Cada worker substitui chunk local pelo recebido

Análise de complexidade Ring AllReduce:

$$\text{Latência} = 2(P-1)\alpha \tag{B.97}$$

$$\text{Bandwidth} = \frac{2(P-1)}{P} \times \frac{M}{\beta} \tag{B.98}$$

$$\text{Tempo total} = 2(P-1)\alpha + \frac{2(P-1)M}{P\beta} \tag{B.99}$$

onde α é latência por hop, β é bandwidth, e M é tamanho dos dados.

B.2.7.2 Hierarchical AllReduce

Para clusters com múltiplos nós, hierarchical AllReduce otimiza comunicação:

Algoritmo 18 – Hierarchical AllReduce

1. **Intra-node AllReduce**: Reduzir dentro de cada nó usando shared memory
2. **Inter-node AllReduce**: Um worker por nó participa de AllReduce global
3. **Intra-node Broadcast**: Broadcast resultado para todos os workers no nó

Análise de performance: Para N nós com G GPUs por nó:

$$T_{hierarchical} = T_{intra} + T_{inter} + T_{broadcast} \tag{B.100}$$

$$= 2(G-1)\alpha_{fast} + \frac{2(G-1)M}{G\beta_{fast}} \tag{B.101}$$

$$+ 2(N-1)\alpha_{slow} + \frac{2(N-1)M}{N\beta_{slow}} \tag{B.102}$$

$$+ (G-1)\alpha_{fast} + \frac{(G-1)M}{G\beta_{fast}} \tag{B.103}$$

B.2.8 Análise de Escalabilidade

B.2.8.1 Métricas de Escalabilidade

Strong Scaling: Manter problema fixo, aumentar número de workers:

$$\text{Efficiency}_{strong} = \frac{T_1}{P \times T_P} \tag{B.104}$$

Weak Scaling: Aumentar problema proporcionalmente ao número de workers:

$$\text{Efficiency}_{weak} = \frac{T_1}{T_P} \tag{B.105}$$

Lei de Amdahl para LLMs:

$$\text{Speedup} = \frac{1}{f + \frac{1-f}{P}} \tag{B.106}$$

onde f é a fração serial (comunicação + sincronização).

Análise empírica: Para treinamento de LLMs, $f \approx 0.1 - 0.2$, limitando speedup máximo a 5-10x independente de P.

Esta análise rigorosa da infraestrutura distribuída fornece as bases teóricas e práticas necessárias para implementação eficiente de treinamento de LLMs em larga escala.

B.3 Otimizações de Sistema e Eficiência Computacional

A eficiência computacional de LLMs depende criticamente de otimizações em múltiplas camadas do sistema, desde kernels CUDA customizados até estratégias de gerenciamento de

memória. Esta seção apresenta uma análise rigorosa das principais técnicas de otimização e suas implicações quantitativas.

B.3.1 Kernel Fusion e Otimizações de Baixo Nível

B.3.1.1 Análise de Memory Bandwidth Utilization

A maioria das operações em LLMs são memory-bound, não compute-bound. A utilização eficiente da hierarquia de memória é fundamental para performance. Para uma operação genérica, a intensidade aritmética é definida como:

$$I = \frac{\text{FLOPs}}{\text{Bytes transferidos}} \tag{B.107}$$

O throughput máximo teórico é limitado por:

$$T_{max} = \min\left(\frac{\text{Peak FLOPS}}{I}, \text{Memory Bandwidth}\right) \tag{B.108}$$

Exemplo - Multiplicação matriz-vetor: Para $y = Ax$ onde $A \in \mathbb{R}^{m\times n}$:

$$\text{FLOPs} = 2mn \tag{B.109}$$

$$\text{Bytes} = (mn + n + m) \times \text{sizeof(dtype)} \tag{B.110}$$

$$I = \frac{2mn}{(mn+n+m)\times 4} \approx \frac{1}{2}\ \text{FLOP/byte} \tag{B.111}$$

Para A100 (1.5 TB/s bandwidth, 312 TFLOPS):

$$T_{max} = \min(312 \times 10^{12}, 1.5 \times 10^{12} \times 0.5) = 750\ \text{GFLOPS} \tag{B.112}$$

Apenas 0.24% da capacidade computacional teórica é utilizada!

B.3.1.2 Fused Attention Kernel

O mecanismo de atenção padrão requer múltiplas transferências de memória:

Implementação Naive:

1. $S = QK^T/\sqrt{d_k} \rightarrow$ Escrever S na memória global
2. $P = \text{softmax}(S) \rightarrow$ Ler S, escrever P
3. $O = PV \rightarrow$ Ler P, escrever O

Análise de transferências: Para sequência de comprimento n e dimensão d:

$$\text{Bytes}_{naive} = \underbrace{2n^2 \times 4}_{\text{S write/read}} + \underbrace{2n^2 \times 4}_{\text{P write/read}} + \underbrace{2nd \times 4}_{\text{Q,K,V,O}} \tag{B.113}$$

$$= 16n^2 + 8nd \text{ bytes} \tag{B.114}$$

Kernel Fusionado: Mantém S e P em shared memory, reduzindo transferências para:

$$\text{Bytes}_{fused} = 8nd \text{ bytes} \tag{B.115}$$

Speedup teórico:

$$\text{Speedup} = \frac{16n^2 + 8nd}{8nd} = \frac{2n}{d} + 1 \tag{B.116}$$

Para $n = 2048, d = 128$: Speedup = 33x em bandwidth!

B.3.1.3 Implementação de Kernel Customizado

Algoritmo 19 – Fused Attention Kernel

1. **Thread block organization**: (B_r, B_c) threads por bloco
2. **Shared memory allocation**:

$$\text{smem}_Q : B_r \times d \text{ elementos} \tag{B.117}$$
$$\text{smem}_K : B_c \times d \text{ elementos} \tag{B.118}$$
$$\text{smem}_V : B_c \times d \text{ elementos} \tag{B.119}$$
$$\text{smem}_S : B_r \times B_c \text{ elementos} \tag{B.120}$$

3. Para cada tile (i, j):
   a) Carregar $Q[i : i + B_r, :]$ para smem_Q
   b) Carregar $K[j : j + B_c, :]$ para smem_K
   c) Computar $\text{smem}_S = \text{smem}_Q \times \text{smem}_K^T / \sqrt{d_k}$
   d) Aplicar softmax em smem_S (com redução segura)
   e) Carregar $V[j : j + B_c, :]$ para smem_V
   f) Acumular output$+ = \text{smem}_S \times \text{smem}_V$

B.3.2 Otimizações de Inferência

B.3.2.1 KV-Cache: Análise Matemática e Otimizações

Durante geração autoregressiva, o cálculo de atenção no passo t é:

$$\text{Attention}_t(Q_t, K_{1:t}, V_{1:t}) = \text{softmax}\left(\frac{Q_t K_{1:t}^T}{\sqrt{d_k}}\right) V_{1:t} \tag{B.121}$$

Como $K_{1:t-1}$ e $V_{1:t-1}$ são invariantes, podem ser cached:

$$K_{1:t} = [K_{\text{cache}}, K_t], \quad V_{1:t} = [V_{\text{cache}}, V_t] \tag{B.122}$$

Análise de memória do KV-cache: Para modelo com L camadas, h cabeças, dimensão d, sequência de comprimento n:

$$\text{Memory}_{KV} = 2 \times L \times h \times n \times \frac{d}{h} \times \text{sizeof(dtype)} = 2Lnd \times \text{sizeof(dtype)} \tag{B.123}$$

Exemplo - LLaMA-7B:

$$L = 32, \quad d = 4096, \quad n = 4096 \tag{B.124}$$

$$\text{Memory}_{KV} = 2 \times 32 \times 4096 \times 4096 \times 2 \text{ bytes} \tag{B.125}$$

$$= 2.15 \text{ GB} \tag{B.126}$$

Para batch size B, a memória total é $B \times 2.15$ GB, rapidamente excedendo capacidade de GPU.

B.3.2.2 Quantização de KV-Cache

Quantização INT8: Mapear valores FP16 $x \in [-\alpha, \alpha]$ para INT8 $q \in [-128, 127]$:

$$q = \text{round}\left(\frac{x}{\alpha} \times 127\right) \tag{B.127}$$

$$\hat{x} = \frac{q \times \alpha}{127} \tag{B.128}$$

O fator de escala α é escolhido para minimizar erro de quantização:

$$\alpha^* = \arg\min_{\alpha} \mathbb{E}[(x - \hat{x})^2] \tag{B.129}$$

Para distribuição normal, $\alpha^* \approx 2.5\sigma$ onde σ é o desvio padrão.

Redução de memória:

$$\text{Compression ratio} = \frac{\text{sizeof(FP16)}}{\text{sizeof(INT8)}} = \frac{2}{1} = 2x \tag{B.130}$$

Análise de erro: O erro RMS de quantização é aproximadamente:

$$\text{RMSE} = \frac{\alpha}{127\sqrt{3}} \approx 0.0046\alpha \tag{B.131}$$

B.3.2.3 PagedAttention: Gerenciamento Eficiente de Memória

PagedAttention (KWON *et al.*, 2023) trata KV-cache como memória virtual paginada:

$$\text{KV-cache} = \bigcup_{i=1}^{N_{\text{pages}}} \text{Page}_i \tag{B.132}$$

onde cada página contém P tokens consecutivos.

Algoritmo 20 – PagedAttention

1. **Inicialização**:
   a) page_size $\leftarrow P$ (tipicamente 16-64)
   b) page_pool $\leftarrow$ initialize_pool()
   c) page_table $\leftarrow \{\}$ (mapeamento lógico $\rightarrow$ físico)
2. **Para cada novo token**:
   a) Se página atual está cheia:
      i. new_page $\leftarrow$ allocate_page(page_pool)
      ii. page_table[logical_addr] $\leftarrow$ new_page
   b) Escrever KV na página atual
   c) Atualizar ponteiros
3. **Durante atenção**:
   a) Para cada página p em page_table:
      i. Carregar K_p, V_p da página física
      ii. Computar atenção parcial
      iii. Acumular resultado

Benefícios quantitativos:

- **Fragmentação**: Redução de 20-40% no desperdício de memória
- **Sharing**: Múltiplas sequências podem compartilhar prefixos comuns
- **Preemption**: Sequências podem ser pausadas/resumidas eficientemente

B.3.3 Dynamic Batching e Continuous Batching

B.3.3.1 Análise de Utilização de GPU

Em inferência tradicional, todas as sequências em um batch devem ter o mesmo comprimento, causando padding:

$$\text{Utilização} = \frac{\sum_{i=1}^{B} \text{length}_i}{B \times \max_i(\text{length}_i)} \tag{B.133}$$

Para distribuição de comprimentos típica (exponencial truncada), a utilização média é apenas 60-70%.

B.3.3.2 Continuous Batching

Algoritmo 21 – Continuous Batching

1. **Estado global**:
   a) active_requests $\leftarrow \{\}$
   b) pending_requests $\leftarrow$ Queue()
   c) max_batch_size $\leftarrow B_{max}$
2. **A cada iteração**:
   a) **Remover sequências completas**:
      i. Para cada r em active_requests:
         - Se r.is_finished():
           - active_requests.remove(r)
           - return_result(r)
   b) **Adicionar novas sequências**:
      i. Enquanto |active_requests| $< B_{max}$ e pending_requests não vazia:
         - $r \leftarrow$ pending_requests.pop()
         - active_requests.add(r)
   c) **Processar batch**:
      i. batch $\leftarrow$ prepare_batch(active_requests)
      ii. outputs $\leftarrow$ model.forward(batch)
      iii. update_requests(active_requests, outputs)

Métricas de performance:

$$\text{Throughput} = \frac{\text{Tokens gerados por segundo}}{\text{Número de GPUs}} \tag{B.134}$$

$$\text{Latência média} = \frac{\sum_i (\text{end\_time}_i - \text{start\_time}_i)}{N_{\text{requests}}} \tag{B.135}$$

$$\text{Utilização} = \frac{\text{Tempo computando}}{\text{Tempo total}} \tag{B.136}$$

B.3.4 Otimizações de Precisão Numérica

B.3.4.1 Mixed Precision Training: Análise Teórica

Mixed precision utiliza FP16 para a maioria dos cálculos, mantendo FP32 para operações sensíveis. A representação FP16 tem:

- 1 bit de sinal
- 5 bits de expoente (range: 2^{-14} a 2^{15})
- 10 bits de mantissa (precisão: $2^{-10} \approx 0.001$)

Problemas de underflow: Gradientes pequenos podem underflow para zero. A probabilidade de underflow para gradiente g é:

$$P(\text{underflow}) = P(|g| < 2^{-14}) = P(|g| < 6.1 \times 10^{-5}) \tag{B.137}$$

Gradient Scaling: Para evitar underflow, gradientes são escalados por fator S:

$$\text{loss}_{scaled} = S \times \text{loss} \tag{B.138}$$

$$\text{grad}_{scaled} = S \times \text{grad} \tag{B.139}$$

$$\text{grad}_{final} = \frac{\text{grad}_{scaled}}{S} \tag{B.140}$$

O fator de escala ótimo maximiza a utilização do range dinâmico:

$$S^* = \frac{2^{15}}{\max(|\text{grad}|)} \times \text{safety\_factor} \tag{B.141}$$

B.3.4.2 Quantização Pós-Treinamento

Quantização Linear: Mapear pesos FP32 $w \in [w_{min}, w_{max}]$ para INT8 $q \in [-128, 127]$:

Algoritmo 22 – Adaptive Gradient Scaling

1. $S \leftarrow 2^{16}$ (inicialização)
2. growth_factor $\leftarrow 2.0$
3. backoff_factor $\leftarrow 0.5$
4. growth_interval $\leftarrow 2000$
5. Para cada iteração:
   a) $\text{loss}_{scaled} \leftarrow S \times \text{loss}$
   b) $\text{grad}_{scaled} \leftarrow \text{backward}(\text{loss}_{scaled})$
   c) Se has_inf_or_nan(grad_{scaled}):
      i. $S \leftarrow S \times$ backoff_factor
      ii. skip_update()
   d) Senão:
      i. $\text{grad} \leftarrow \text{grad}_{scaled}/S$
      ii. optimizer.step(grad)
      iii. Se iteração é múltiplo de growth_interval:
         - $S \leftarrow S \times$ growth_factor

$$\text{scale} = \frac{w_{max} - w_{min}}{255} \tag{B.142}$$

$$\text{zero\_point} = -128 - \frac{w_{min}}{\text{scale}} \tag{B.143}$$

$$q = \text{round}\left(\frac{w}{\text{scale}} + \text{zero\_point}\right) \tag{B.144}$$

$$\hat{w} = \text{scale} \times (q - \text{zero\_point}) \tag{B.145}$$

Análise de erro: O erro de quantização segue distribuição uniforme:

$$\varepsilon \sim \mathscr{U}\left(-\frac{\text{scale}}{2}, \frac{\text{scale}}{2}\right) \tag{B.146}$$

A variância do erro é:

$$\text{Var}(\varepsilon) = \frac{\text{scale}^2}{12} \tag{B.147}$$

Para minimizar impacto na performance, o erro deve satisfazer:

$$\frac{\text{Var}(\varepsilon)}{\text{Var}(w)} < \tau \tag{B.148}$$

onde τ é um threshold de tolerância (tipicamente 0.01-0.05).

Aplicação em pós-treinamento (QLoRA): Em fine-tuning de LLMs, quantização pós-treinamento pode ser combinada com PEFT para reduzir VRAM sem treinar todos os parâmetros. QLoRA mantém o modelo base quantizado (e.g., 4-bit NF4) e treina apenas adaptadores LoRA em maior precisão, viabilizando ajustes em GPU única com boa preservação de qualidade (DETTMERS *et al.*, 2023).

Esta análise rigorosa das otimizações de sistema fornece as bases teóricas e práticas necessárias para implementação eficiente de LLMs em ambientes com recursos limitados.

APÊNDICE

C

SCALING LAWS E CUSTOS (MATERIAL COMPLEMENTAR)

C.1 Scaling Laws e Análise de Compute Requirements

As leis de escala constituem um framework matemático fundamental para compreender a relação entre recursos computacionais, tamanho de modelos e performance. Esta seção apresenta uma análise rigorosa das descobertas empíricas e suas implicações teóricas para o desenvolvimento de LLMs.

C.1.1 Fundamentação Matemática das Scaling Laws

C.1.1.1 Lei de Potência de Kaplan et al. (2020)

O trabalho seminal de Kaplan et al. (KAPLAN *et al.*, 2020) estabeleceu que a performance de modelos de linguagem segue leis de potência previsíveis. A loss de validação L pode ser expressa como função de três variáveis independentes:

$$L(N) = \left(\frac{N_c}{N}\right)^{\alpha_N} + L_\infty \tag{C.1}$$

$$L(D) = \left(\frac{D_c}{D}\right)^{\alpha_D} + L_\infty \tag{C.2}$$

$$L(C) = \left(\frac{C_c}{C}\right)^{\alpha_C} + L_\infty \tag{C.3}$$

onde:

- N = número de parâmetros (excluindo embeddings)
- D = tamanho do dataset (em tokens)

- C = compute usado no treinamento (em FLOPs)
- L_∞ = loss irreducível (entropia intrínseca dos dados)

Parâmetros empíricos de Kaplan:

$$\alpha_N \approx 0.076, \quad N_c \approx 8.8 \times 10^{13} \tag{C.4}$$
$$\alpha_D \approx 0.095, \quad D_c \approx 5.4 \times 10^{13} \tag{C.5}$$
$$\alpha_C \approx 0.050, \quad C_c \approx 3.1 \times 10^{8} \tag{C.6}$$
$$L_\infty \approx 1.69 \tag{C.7}$$

Análise de sensibilidade: A derivada da loss em relação a cada variável indica sensibilidade:

$$\frac{\partial L}{\partial N} = -\alpha_N \frac{N_c^{\alpha_N}}{N^{\alpha_N+1}} \tag{C.8}$$
$$\frac{\partial L}{\partial D} = -\alpha_D \frac{D_c^{\alpha_D}}{D^{\alpha_D+1}} \tag{C.9}$$
$$\frac{\partial L}{\partial C} = -\alpha_C \frac{C_c^{\alpha_C}}{C^{\alpha_C+1}} \tag{C.10}$$

Para $N = 10^{10}, D = 10^{11}, C = 10^{20}$:

$$\left|\frac{\partial L}{\partial N}\right| \approx 2.1 \times 10^{-12} \tag{C.11}$$
$$\left|\frac{\partial L}{\partial D}\right| \approx 1.8 \times 10^{-13} \tag{C.12}$$
$$\left|\frac{\partial L}{\partial C}\right| \approx 3.4 \times 10^{-22} \tag{C.13}$$

Isso indica que aumentar N tem maior impacto que aumentar D ou C isoladamente.

C.1.1.2 Revisão de Chinchilla: Optimal Compute Allocation

Hoffmann et al. (HOFFMANN *et al.*, 2022) revisaram as conclusões de Kaplan, demonstrando que a alocação ótima de compute requer balanceamento entre N e D. Neste contexto, *Chinchilla* refere-se tanto ao modelo de 70B parâmetros descrito no trabalho quanto à regra prática de *compute-optimal scaling* derivada a partir desses experimentos (aqui usada apenas como referência de escala).

Modelo de Chinchilla: A loss é modelada como função de ambos N e D:

$$L(N,D) = E + \frac{A}{N^\alpha} + \frac{B}{D^\beta} \tag{C.14}$$

onde E é a entropia dos dados, e os parâmetros empíricos são:

$$A = 406.4, \quad \alpha = 0.34 \tag{C.15}$$

$$B = 410.7, \quad \beta = 0.28 \tag{C.16}$$

$$E = 1.69 \tag{C.17}$$

Otimização com restrição de compute: Para budget computacional fixo C, usando a relação $C = 6ND$ (aproximação para treinamento), o problema de otimização é:

$$\min_{N,D} L(N,D) \quad \text{sujeito a} \quad 6ND = C \tag{C.18}$$

Usando multiplicadores de Lagrange:

$$\mathscr{L} = E + \frac{A}{N^{\alpha}} + \frac{B}{D^{\beta}} + \lambda(6ND - C) \tag{C.19}$$

As condições de primeira ordem são:

$$\frac{\partial \mathscr{L}}{\partial N} = -\frac{A\alpha}{N^{\alpha+1}} + 6\lambda D = 0 \tag{C.20}$$

$$\frac{\partial \mathscr{L}}{\partial D} = -\frac{B\beta}{D^{\beta+1}} + 6\lambda N = 0 \tag{C.21}$$

Resolvendo o sistema:

$$N^* = \left(\frac{A\alpha}{B\beta}\right)^{\frac{1}{\alpha+\beta}} \left(\frac{C}{6}\right)^{\frac{\beta}{\alpha+\beta}} \tag{C.22}$$

$$D^* = \left(\frac{B\beta}{A\alpha}\right)^{\frac{1}{\alpha+\beta}} \left(\frac{C}{6}\right)^{\frac{\alpha}{\alpha+\beta}} \tag{C.23}$$

Relação de escala ótima: Substituindo os valores empíricos:

$$N^* \propto C^{0.50} \tag{C.24}$$

$$D^* \propto C^{0.50} \tag{C.25}$$

Isso implica que compute deve ser dividido igualmente entre aumentar o modelo e aumentar os dados.

C.1.2 Análise Quantitativa de Compute Requirements

C.1.2.1 Estimativa de FLOPs para Treinamento

O número de FLOPs para treinar um Transformer pode ser calculado precisamente:

Forward pass por token: Para modelo com L camadas, d dimensões, h cabeças de atenção, d_{ff} dimensão feed-forward:

$$\text{FLOPs}_{attention} = 4Ld^2 + 2Ln^2d \tag{C.26}$$

$$\text{FLOPs}_{feedforward} = 8Ldd_{ff} \tag{C.27}$$

$$\text{FLOPs}_{embeddings} = 2Vd \tag{C.28}$$

$$\text{FLOPs}_{total} = 4Ld^2 + 2Ln^2d + 8Ldd_{ff} + 2Vd \tag{C.29}$$

onde n é o comprimento da sequência e V é o tamanho do vocabulário.

Aproximação para modelos grandes: Para $d_{ff} = 4d$ e $V \ll Ld$ (típico para modelos grandes):

$$\text{FLOPs}_{forward} \approx 2Ld(2d + n + 16d) = 2Ld(18d + n) \tag{C.30}$$

Para $n \ll d$ (sequências curtas comparadas à dimensão):

$$\text{FLOPs}_{forward} \approx 36Ld^2 \tag{C.31}$$

Treinamento completo: Considerando backward pass (2x forward) e overhead (1.2x):

$$\text{FLOPs}_{training} = 1.2 \times 3 \times 36Ld^2 \times D = 129.6Ld^2D \tag{C.32}$$

Para modelos padrão onde $N \approx 12Ld^2$:

$$\text{FLOPs}_{training} \approx 6ND \tag{C.33}$$

C.1.2.2 Análise de Custos por Modelo

Tabela de custos computacionais:

Modelo	N (B)	D (B)	C (FLOPs)	Dias (A100)	Custo ($)
GPT-2	1.5	40	3.6×10^{20}	1.3	$10K
GPT-3	175	300	3.1×10^{23}	1,150	$8.5M
PaLM	540	780	2.5×10^{24}	9,300	$69M
Chinchilla	70	1,400	5.9×10^{23}	2,200	$16M
LLaMA-65B	65	1,400	5.5×10^{23}	2,040	$15M

Tabela 20 – Análise de custos para modelos representativos (preços A100 2024)

Cálculo detalhado para GPT-3:

$$C = 6 \times 175 \times 10^9 \times 300 \times 10^9 = 3.15 \times 10^{23} \text{ FLOPs} \tag{C.34}$$

$$t_{A100} = \frac{3.15 \times 10^{23}}{312 \times 10^{12} \times 0.4 \times 3600 \times 24} = 292 \text{ GPU-anos} \tag{C.35}$$

$$\text{Custo} = 292 \times 365 \times 24 \times \$3.06 = \$7.5M \tag{C.36}$$

C.1.3 Emergent Abilities e Critical Thresholds

A literatura relata que algumas capacidades em modelos de linguagem podem apresentar comportamento não linear conforme se aumenta a escala (parâmetros, dados e compute). Wei et al. (WEI *et al.*, 2022) descrevem esse fenômeno como *emergent abilities*, em que certas tarefas parecem "desbloquear" desempenho útil após determinado patamar.

Para esta dissertação, o ponto relevante é metodológico: como os protótipos v1–v6 operam em escala muito reduzida, não é objetivo observar emergência de habilidades. Isso reforça a separação em duas trilhas: (i) v1–v6 como evidência de custo/engenharia e sensibilidade a tokenização/limpeza; e (ii) Trilha 2 como adaptação eficiente de um modelo já em regime de escala (LLaMA 3.1-8B), onde ganhos de utilidade são medidos por avaliação extrínseca (QA, sumarização e reescrita).

C.1.4 Extrapolação e Predição de Performance

C.1.4.1 Modelo Preditivo Baseado em Scaling Laws

Combinando as leis de Kaplan e Chinchilla, podemos construir um modelo preditivo:

$$L_{pred}(C) = E + \frac{A}{N^*(C)^\alpha} + \frac{B}{D^*(C)^\beta} \tag{C.37}$$

onde $N^*(C)$ e $D^*(C)$ são as alocações ótimas de Chinchilla.

Intervalo de confiança: Considerando incerteza nos parâmetros, o intervalo de confiança é:

$$L_{pred}(C) \pm z_{\alpha/2}\sqrt{\text{Var}(L_{pred})} \tag{C.38}$$

onde:

$$\text{Var}(L_{pred}) = \sum_i \left(\frac{\partial L_{pred}}{\partial \theta_i}\right)^2 \text{Var}(\theta_i) \tag{C.39}$$

C.1.4.2 Extrapolação para Modelos Futuros

Predição para compute de 10^{26} FLOPs: Usando as leis de Chinchilla:

$$N^* = \left(\frac{10^{26}}{6}\right)^{0.50} \approx 4.1 \times 10^{12} \text{ parâmetros} \tag{C.40}$$

$$D^* = \left(\frac{10^{26}}{6}\right)^{0.50} \approx 4.1 \times 10^{12} \text{ tokens} \tag{C.41}$$

Performance esperada:

$$L_{pred} = 1.69 + \frac{406.4}{(4.1 \times 10^{12})^{0.34}} + \frac{410.7}{(4.1 \times 10^{12})^{0.28}} \tag{C.42}$$

$$\approx 1.69 + 0.012 + 0.018 = 1.72 \tag{C.43}$$

C.1.5 Implicações para Desenvolvimento de LLMs

C.1.5.1 Estratégia de Alocação de Recursos

Função objetivo multi-critério: Para budget limitado, otimizar:

$$\max_{N,D,C} \alpha \cdot \text{Performance}(N,D) - \beta \cdot \text{Cost}(C) - \gamma \cdot \text{Time}(C) \tag{C.44}$$

sujeito a:

$$6ND \leq C_{max} \tag{C.45}$$

$$N \geq N_{min} \text{ (threshold de emergência)} \tag{C.46}$$

$$D \geq D_{min} \text{ (qualidade mínima)} \tag{C.47}$$

Solução por programação dinâmica: Para orçamentos discretos $C_1 < C_2 < \ldots < C_k$:

$$V(C_i) = \max_{N,D} \{\text{Performance}(N,D) + \delta \cdot V(C_i - 6ND)\} \tag{C.48}$$

onde δ é o fator de desconto temporal.

C.1.5.2 Análise de ROI por Escala

Retornos marginais decrescentes: como as scaling laws seguem leis de potência, o ganho marginal por unidade de compute diminui à medida que o budget cresce. Em termos analíticos, a derivada $\frac{\partial L}{\partial C}$ decai rapidamente com C, implicando que dobrar compute tende a produzir melhorias menores na loss em regimes já altos.

Ponto de saturação: O ROI marginal se torna negativo quando:

$$\frac{d(\text{Performance})}{dC} \cdot \text{Value per unit performance} < \text{Cost per FLOP} \tag{C.49}$$

O ponto de "saturação" é altamente dependente do domínio e da forma como *valor* é definido (e.g., métricas extrínsecas, custo de erros, requisitos de latência).

Estratégia ótima escalonada:

1. **Fase 1** ($C < 10^{21}$): Maximizar learning rate, usar modelos pequenos

2. **Fase 2** ($10^{21} < C < 10^{23}$): Balancear N e D segundo Chinchilla
3. **Fase 3** ($C > 10^{23}$): Focar em qualidade de dados e fine-tuning

Esta análise rigorosa das scaling laws fornece uma base quantitativa sólida para tomada de decisões estratégicas no desenvolvimento de LLMs, permitindo otimização eficiente de recursos computacionais limitados.

C.2 Análise Técnica: Arquitetura Ótima vs. Restrições Orçamentárias

Esta seção apresenta uma análise quantitativa das trade-offs entre arquiteturas ideais para desenvolvimento de LLMs e as limitações práticas impostas por restrições orçamentárias, com foco em otimização de recursos computacionais.

C.2.1 Modelagem de Custos Computacionais

C.2.1.1 Função de Custo Total

O custo total para desenvolvimento de um LLM pode ser modelado como:

$$C_{total} = C_{compute} + C_{storage} + C_{network} + C_{personnel} + C_{data} \tag{C.50}$$

onde cada componente segue diferentes leis de escala em relação ao tamanho do modelo N e dataset D.

Custo Computacional: Baseado nas leis de Chinchilla, o compute ótimo é $C = 6ND$. O custo monetário é:

$$C_{compute} = \frac{6ND \times \text{cost\_per\_FLOP}}{\text{utilization\_factor}} \tag{C.51}$$

Para hardware moderno:

$$\text{cost\_per\_FLOP} = \frac{\text{hardware\_cost\_per\_hour}}{\text{peak\_FLOPS} \times 3600} \tag{C.52}$$

$$\text{utilization\_factor} \approx 0.3 - 0.5 \text{ (típico para treinamento)} \tag{C.53}$$

Custo de Armazenamento:

$$C_{storage} = D \times \text{bytes\_per\_token} \times \text{cost\_per\_GB} \times \text{replication\_factor} \tag{C.54}$$

Custo de Rede: Para treinamento distribuído com P nós:

$$C_{network} = f(P) \times \text{bandwidth\_cost} \times \text{training\_time} \tag{C.55}$$

onde $f(P)$ representa a topologia de rede (linear para ring, quadrática para all-to-all).

C.2.1.2 Análise de Pareto: Performance vs. Custo

Definindo eficiência como performance por unidade de custo:

$$\eta = \frac{\text{Performance}(N,D,C)}{\text{Cost}(N,D,C)} \tag{C.56}$$

A fronteira de Pareto é encontrada resolvendo:

$$\max_{N,D,C} \eta \quad \text{sujeito a} \quad C_{total} \leq B \tag{C.57}$$

onde B é o budget disponível.

Solução analítica para caso simplificado: Assumindo Performance $\propto (ND)^{\alpha}$ e Cost $\propto ND$:

$$N^* = \left(\frac{\alpha B}{2}\right)^{\frac{1}{1-\alpha}} \tag{C.58}$$

$$D^* = \left(\frac{\alpha B}{2}\right)^{\frac{1}{1-\alpha}} \tag{C.59}$$

Para $\alpha = 0.34$ (baseado em scaling laws empíricas):

$$N^* = D^* = (0.17B)^{1.52} \tag{C.60}$$

C.2.2 Arquiteturas de Referência

Os cenários Tier-1 a Tier-3 a seguir são **arquiteturas de referência** (valores aproximados) para contextualizar ordens de grandeza de custo e escala em treinamento de LLMs. Eles não correspondem ao ambiente experimental deste trabalho, que foi conduzido em GPU única de consumo e instâncias sob demanda (Vast.ai), conforme descrito no Capítulo 4.

C.2.2.1 Configuração Tier-1: Infraestrutura de Pesquisa Global

Especificações técnicas:

- **Compute**: 2048x H100 (80GB), 2.02 ExaFLOPS agregado
- **Interconnect**: NVLink Switch + InfiniBand NDR (400 Gb/s)
- **Storage**: 50 PB NVMe distribuído, 1 TB/s bandwidth agregado

- **Memory**: 163 TB HBM total, 6.8 PB/s bandwidth agregado

Análise de custos (5 anos):

$$C_{hardware} = 2048 \times \$30,000 + \text{infraestrutura} = \$80M \tag{C.61}$$
$$C_{energia} = 2048 \times 700W \times 24 \times 365 \times 5 \times \$0.10/kWh = \$62M \tag{C.62}$$
$$C_{cooling} = 0.3 \times C_{energia} = \$19M \tag{C.63}$$
$$C_{facilities} = \$50M \tag{C.64}$$
$$C_{total} = \$211M \tag{C.65}$$

Capacidade de treinamento: Modelo de 1T parâmetros com 20T tokens (budget Chinchilla-optimal):

$$C_{required} = 6 \times 10^{12} \times 20 \times 10^{12} = 1.2 \times 10^{26} \text{ FLOPs} \tag{C.66}$$
$$t_{training} = \frac{1.2 \times 10^{26}}{2.02 \times 10^{18} \times 0.4} = 148 \text{ dias} \tag{C.67}$$

C.2.2.2 Configuração Tier-2: Centro de Pesquisa Nacional

Especificações técnicas:

- **Compute**: 256x A100 (80GB), 80 PetaFLOPS agregado
- **Interconnect**: InfiniBand HDR (200 Gb/s)
- **Storage**: 5 PB NVMe, 100 GB/s bandwidth
- **Memory**: 20 TB HBM total, 410 TB/s bandwidth agregado

Análise de custos (3 anos):

$$C_{hardware} = 256 \times \$15,000 + \text{infraestrutura} = \$6M \tag{C.68}$$
$$C_{energia} = 256 \times 400W \times 24 \times 365 \times 3 \times \$0.12/kWh = \$3M \tag{C.69}$$
$$C_{operacional} = \$2M \tag{C.70}$$
$$C_{total} = \$11M \tag{C.71}$$

Capacidade de treinamento: Modelo de 70B parâmetros com 1.4T tokens:

$$C_{required} = 6 \times 70 \times 10^{9} \times 1.4 \times 10^{12} = 5.88 \times 10^{23} \text{ FLOPs} \tag{C.72}$$
$$t_{training} = \frac{5.88 \times 10^{23}}{80 \times 10^{15} \times 0.35} = 210 \text{ dias} \tag{C.73}$$

C.2.2.3 Configuração Tier-3: Laboratório Universitário

Especificações técnicas:

- **Compute**: 32x RTX 4090 (24GB), 2.6 PetaFLOPS agregado
- **Interconnect**: Ethernet 100 Gb/s
- **Storage**: 500 TB NVMe, 10 GB/s bandwidth
- **Memory**: 768 GB GDDR6X total, 26 TB/s bandwidth agregado

Análise de custos (2 anos):

$$C_{hardware} = 32 \times \$1,600 + \text{infraestrutura} = \$80K \tag{C.74}$$

$$C_{energia} = 32 \times 450W \times 24 \times 365 \times 2 \times \$0.15/kWh = \$19K \tag{C.75}$$

$$C_{operacional} = \$20K \tag{C.76}$$

$$C_{total} = \$119K \tag{C.77}$$

Capacidade de treinamento: Limitado a fine-tuning de modelos pré-treinados devido a restrições de memória.

C.2.3 Estratégias de Otimização para Recursos Limitados

C.2.3.1 Parameter-Efficient Fine-Tuning (PEFT)

Low-Rank Adaptation (LoRA): Em vez de atualizar toda a matriz de pesos $W \in \mathbb{R}^{d\times k}$, LoRA aproxima a atualização como:

$$W' = W + \Delta W = W + BA \tag{C.78}$$

onde $B \in \mathbb{R}^{d\times r}$, $A \in \mathbb{R}^{r\times k}$, e $r \ll \min(d,k)$.

Redução de parâmetros:

$$\text{Parâmetros originais} = dk \tag{C.79}$$

$$\text{Parâmetros LoRA} = r(d+k) \tag{C.80}$$

$$\text{Compression ratio} = \frac{dk}{r(d+k)} \tag{C.81}$$

Para $d = k = 4096, r = 64$:

$$\text{Compression ratio} = \frac{4096^2}{64 \times 8192} = 32x \tag{C.82}$$

Análise de performance: A capacidade de representação de LoRA é limitada pelo rank r. Para preservar ε-fração da informação:

$$r \geq \frac{\text{rank}(W)}{\varepsilon} \tag{C.83}$$

onde $\text{rank}(W)$ é o rank efetivo da matriz original.

Aplicação em adaptação (CPT/SFT): O mesmo mecanismo de PEFT pode ser aplicado tanto para *continued pretraining* (CPT), quando o objetivo é adaptar o modelo a uma distribuição linguística/domínio (texto bruto), quanto para *supervised fine-tuning* (SFT), quando o objetivo é seguir instruções a partir de pares instrução–resposta. Em ambos os casos, a diferença central está no tipo de dado e no objetivo experimental, e não no mecanismo de otimização em si (GURURANGAN *et al.*, 2020; OUYANG *et al.*, 2022).

QLoRA (Quantized LoRA): Em cenários onde o modelo base não cabe em FP16/BF16 na VRAM disponível, QLoRA combina LoRA com quantização dos pesos (tipicamente 4-bit, como NF4), mantendo o modelo base quantizado e treinando apenas os adaptadores em maior precisão. Na prática, essa combinação reduz drasticamente o consumo de memória e amplia a viabilidade de pós-treinamento em GPU única (DETTMERS *et al.*, 2023).

C.2.3.2 Quantização Adaptativa

Mixed-bit quantização: Diferentes camadas requerem diferentes precisões. A alocação ótima de bits minimiza:

$$L_{quant} = \sum_{i=1}^{L} w_i \cdot \text{MSE}_i(b_i) \tag{C.84}$$

sujeito a:

$$\sum_{i=1}^{L} b_i \cdot s_i \leq B_{total} \tag{C.85}$$

onde b_i são os bits para camada i, s_i é o tamanho da camada, e w_i é o peso de importância.

Solução via multiplicadores de Lagrange:

$$b_i^* = \arg\min_{b_i} [w_i \cdot \text{MSE}_i(b_i) + \lambda s_i b_i] \tag{C.86}$$

Heurística prática:

- **Attention weights**: 8-16 bits (alta sensibilidade)
- **Feed-forward weights**: 4-8 bits (menor sensibilidade)
- **Embeddings**: 8-16 bits (impacto direto na qualidade)

C.2.3.3 Gradient Checkpointing e Memory Optimization

Análise de trade-off tempo-memória: Gradient checkpointing reduz uso de memória de $O(L)$ para $O(\sqrt{L})$ ao custo de recomputação.

Para modelo com L camadas:

$$\text{Memory}_{standard} = L \times M_{activation} \tag{C.87}$$

$$\text{Memory}_{checkpointed} = \sqrt{L} \times M_{activation} \tag{C.88}$$

$$\text{Compute}_{overhead} = \frac{\sqrt{L}}{L} = \frac{1}{\sqrt{L}} \tag{C.89}$$

Estratégia ótima de checkpointing: Minimizar função objetivo:

$$J = \alpha \cdot \text{Memory} + \beta \cdot \text{Compute\_time} \tag{C.90}$$

A solução ótima para número de checkpoints é:

$$k^* = \sqrt{\frac{\beta \cdot T_{forward}}{\alpha \cdot M_{activation}}} \tag{C.91}$$

C.2.4 Análise Comparativa de Estratégias

C.2.4.1 Matriz de Decisão Técnica

Para um budget fixo B, comparamos estratégias baseadas em métricas quantitativas:

Estratégia	Performance	Custo	Tempo	Risco
Treinamento completo	$P_{baseline}$	$C_{baseline}$	$T_{baseline}$	Alto
Fine-tuning + LoRA	$0.85P_{baseline}$	$0.05C_{baseline}$	$0.1T_{baseline}$	Baixo
Quantização + PEFT	$0.80P_{baseline}$	$0.03C_{baseline}$	$0.08T_{baseline}$	Médio
Destilação	$0.75P_{baseline}$	$0.15C_{baseline}$	$0.3T_{baseline}$	Médio

Tabela 21 – Análise comparativa de estratégias (valores normalizados)

Função de utilidade multi-objetivo:

$$U = w_1 \cdot \text{Performance} - w_2 \cdot \text{Cost} - w_3 \cdot \text{Time} - w_4 \cdot \text{Risk} \tag{C.92}$$

Para pesos típicos $w = [0.4, 0.3, 0.2, 0.1]$:

$$U_{completo} = 0.4 \times 1.0 - 0.3 \times 1.0 - 0.2 \times 1.0 - 0.1 \times 0.8 = -0.18 \tag{C.93}$$

$$U_{LoRA} = 0.4 \times 0.85 - 0.3 \times 0.05 - 0.2 \times 0.1 - 0.1 \times 0.2 = 0.295 \tag{C.94}$$

C.2.4.2 Roadmap de Implementação Baseado em Recursos

Fase 1 - Prova de Conceito (orçamento baixo):

- Base: Modelo open-source (LLaMA-7B, Mistral-7B)
- Técnica: LoRA fine-tuning com dados específicos
- Hardware: poucas GPUs de consumo e/ou instâncias sob demanda
- Objetivo: Validar viabilidade técnica

Fase 2 - MVP Funcional (orçamento intermediário):

- Base: Modelo de 13B parâmetros
- Técnica: Full fine-tuning + quantização
- Hardware: dezenas de GPUs classe data-center (ou equivalente sob demanda)
- Objetivo: Produto mínimo viável

Fase 3 - Produto Nacional (orçamento alto):

- Base: Família de modelos (7B, 13B, 30B)
- Técnica: Treinamento híbrido + otimizações avançadas
- Hardware: infraestrutura multi-GPU em escala de dezenas/centenas (ambiente dedicado)
- Objetivo: Competitividade internacional

Observação sobre ROI: embora o conceito de retorno sobre investimento (ROI) ajude a discutir custo-benefício, seu valor depende fortemente do contexto (domínio, requisitos legais, custos de dados, riscos e operação). Por isso, este trabalho não estima ROI econômico; em vez disso, reporta custos observados (tempo de GPU e custo em USD) e ganhos mensuráveis nas avaliações propostas/executadas.

Esta discussão fornece uma base quantitativa para tomada de decisões sobre arquitetura e alocação de recursos em projetos de desenvolvimento de LLMs sob restrições orçamentárias, mantendo o foco em evidências reprodutíveis e medições diretas nos experimentos descritos.


ICMC
USP
SÃO CARLOS