

# Pré-Treino Eficiente de Modelos de Linguagem

Implementação e Otimização de Modelo para Português

Daniel Fazzioni



**UFG**

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

DANIEL FAZZIONI

## **Pré-Treino Eficiente de Modelos de Linguagem**

Implementação e Otimização de Modelo para Português

Goiânia  
2025



UNIVERSIDADE FEDERAL DE GOIÁS  
INSTITUTO DE INFORMÁTICA

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

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

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

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

Nome(s) completo(s) do(a)(s) autor(a)(es)(as): DANIEL FAZZIONI

Título do trabalho: Pré-Treino Eficiente de Modelos de Linguagem

Implementação e Otimização de Modelo para Português

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

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

#### Casos de embargo:

- Solicitação de registro de patente;
- Submissão de artigo em revista científica;
- Publicação como capítulo de livro.

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



Documento assinado eletronicamente por **Daniel Fazzioni, Discente**, em 04/02/2026, às 17:36, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do [Decreto nº 10.543, de 13 de novembro de 2020](#).



Documento assinado eletronicamente por **Fernando Marques Federson, Professor do Magistério Superior**, em 13/03/2026, às 11:24, conforme horário oficial de Brasília, com fundamento no § 3º do art. 4º do [Decreto nº 10.543, de 13 de novembro de 2020](#).



A autenticidade deste documento pode ser conferida no site [https://sei.ufg.br/sei/controlador\\_externo.php?acao=documento\\_conferir&id\\_orgao\\_acesso\\_externo=0](https://sei.ufg.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0), informando o código verificador **5956334** e o código CRC **3B0C3503**.

---

**Referência:** Processo nº 23070.005484/2026-01

SEI nº 5956334

DANIEL FAZZIONI

**Pré-Treino Eficiente de Modelos de Linguagem**  
Implementação e Otimização de Modelo para Português

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

Orientador: Prof. Dr. Fernando Marques Federson

Goiânia  
2025

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

FAZZIONI, DANIEL  
Pré-Treino Eficiente de Modelos de Linguagem [manuscrito]:  
Implementação e Otimização de Modelo para Português / DANIEL FAZZIONI. -  
2025.

119 f.: 2025

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

1. Inteligência Artificial. 2. Large Language Models. 3. Pré-treino.

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

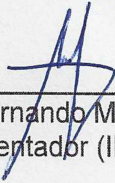
CDU 004

DANIEL FAZZIONI

**Pré-Treino Eficiente de Modelos de Linguagem**  
Implementação e Otimização de Modelo para Português

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

Data da Aprovação: 09 de dezembro de 2025.



---

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



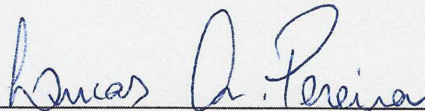
---

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



---

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



---

Prof. Me. Lucas Araújo Pereira  
(INF-UFG)

DANIEL FAZZIONI

## **Pré-Treino Eficiente de Modelos de Linguagem**

Implementação e Otimização de Modelo para Português

### **RESUMO**

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

Palavras-chave: Inteligência artificial; Large language models; Pré-treino.

### **ABSTRACT**

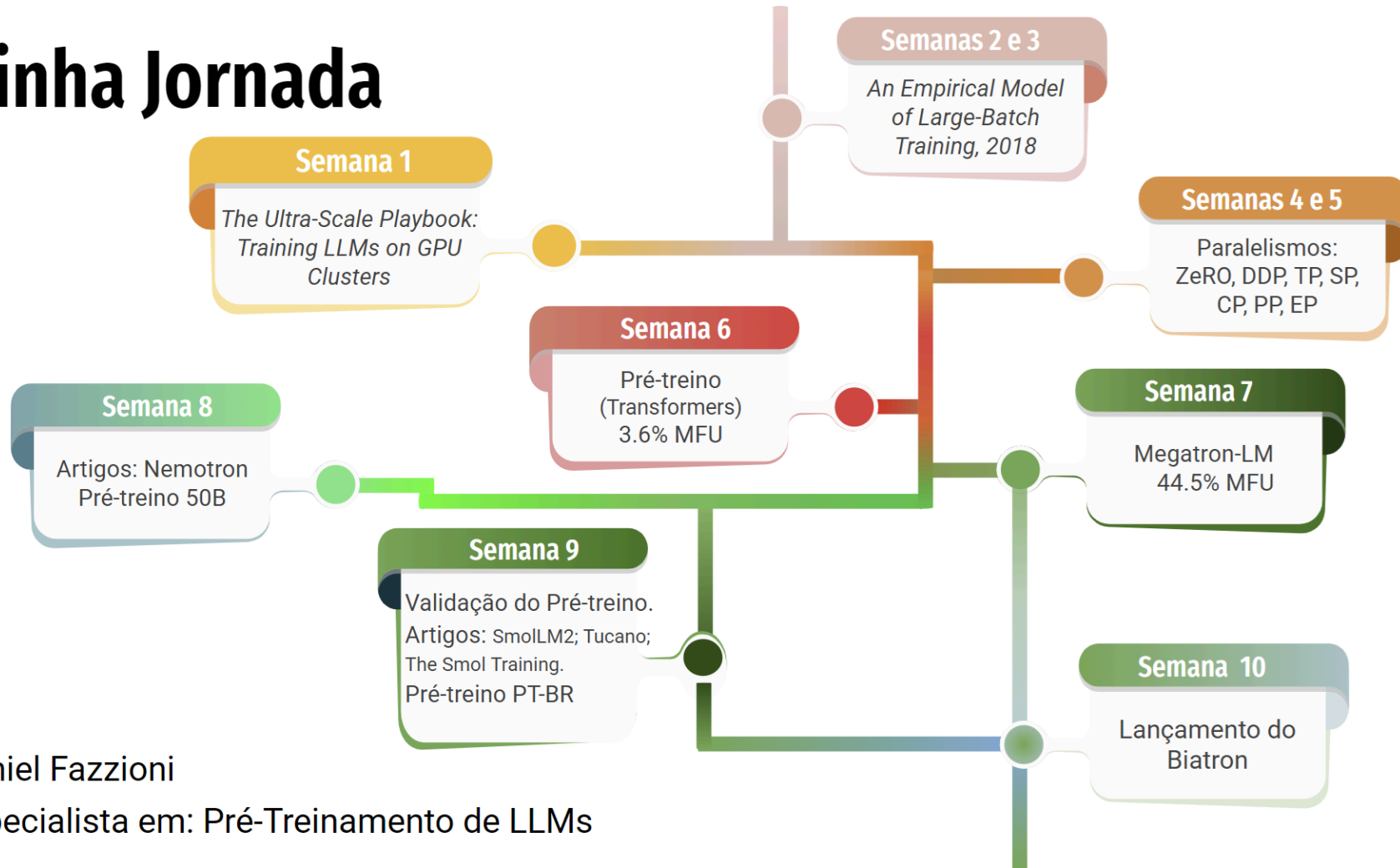
This Course Completion Report aims to bring together the results of my journey to become an expert in **LLM Pre-Training**. An illustration and its narrative describe the work periods. The Appendices contain the Delivery Acceptance Terms and the results obtained during each work period.

Keywords: Artificial intelligence; Large language models; Pre-workout.

Goiânia

2025

# Minha Jornada



Daniel Fazzioni

Especialista em: Pré-Treinamento de LLMs

---

## MINHA JORNADA

**Nome:** Daniel Fazzioni

**Especialidade:** Pré-Treinamento de LLMs

### Objetivo deste documento

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

### Minha Jornada

Minha Jornada iniciou-se na **Semana 1** com a definição da área de conhecimento da minha especialização. Eu tinha um interesse genuíno não apenas em entender o funcionamento do treinamento distribuído de grandes modelos, mas também em compreender o impacto da utilização de múltiplas GPUs nesse processo. A partir de uma pesquisa, encontrei a obra que direcionou o tema: “The Ultra-Scale Playbook: Training LLMs on GPU Clusters”<sup>2</sup>. Iniciei a leitura pelo primeiro capítulo, focado no comportamento do treino em uma única GPU, o que serviu como um excelente ponto de partida. Compreender a dinâmica dos gradientes durante a otimização, bem como gerenciar a memória, gerou insights fundamentais para delimitar as especificações da área de conhecimento. Alguns detalhes que me chamaram a atenção nesses estudos iniciais estão disponibilizados no **Apêndice 1**.

---

<sup>1</sup> Dez Semanas, entre setembro de 2025 e dezembro de 2025.

<sup>2</sup> TAZI, Nouamane et al. *The Ultra-Scale Playbook: Training LLMs on GPU Clusters*. Hugging Face: nanotron, 2025. Disponível em: <https://huggingface.co/spaces/nanotron/ultrascale-playbook>.

Ao aprofundar-me nas referências do livro, não imaginava que o artigo “An Empirical Model of Large-Batch Training” demandaria integralmente as **Semanas 2 e 3**. A constatação de que o *batch size* influencia diretamente a estabilidade da convergência — indo além das questões de custo computacional e de velocidade — foi um ponto central do estudo. Nesse trabalho, os autores também definem o ruído do gradiente e propõem uma métrica para quantificá-lo. Esse foi o meu primeiro desafio técnico na **Residência**: compreender a fundamentação matemática da escala de ruído. Foi necessário revisitar o texto diversas vezes, mas o processo resultou na alteração da minha percepção sobre o gradiente; visualizar a *loss* como uma função de energia potencial contribuiu para consolidar os fundamentos sobre otimização de redes neurais. As anotações referentes a esse estudo encontram-se no **Apêndice 2**.

Nas **Semanas 4 e 5**, o foco voltou-se para as estratégias de treinamento distribuído apresentadas no livro citado, cujos detalhes técnicos organizei no **Apêndice 3**. Explorei as diversas abordagens de paralelismo conhecidas, que incluem: a replicação de instâncias (Data Parallelism); o particionamento do modelo ou de tensores; a otimização na transferência de dados no paralelismo tensorial; o processamento paralelo de sequências de texto; o planejamento de distribuição de Experts (MoE); e a remoção de redundância de parâmetros na memória (estratégias como ZeRO/FSDP). Essas técnicas permitem a utilização eficiente dos recursos, viabilizando treinamentos em escalas superiores às que uma infraestrutura permitiria “ingenuamente”. Contudo, observei que todas elas exigem o gerenciamento de um compromisso crítico (trade-off) entre consumo de memória da GPU, capacidade de computação e latência na comunicação entre os dispositivos.

Após cinco **Semanas** dedicadas à fundamentação, a etapa seguinte consistiu em validar esses conceitos com experimentos. Na **Semana 6**, decidi avaliar a eficiência de treinamento utilizando a biblioteca Transformers. Para viabilizar o experimento, defini um fluxo de pré-treinamento: treinei um tokenizador com vocabulário de 32.000 tokens em português. Instanciei um LLM com 120 milhões de parâmetros, conforme detalhado no **Apêndice 4**. Ao iniciar a execução e aferir a performance, o resultado foi inesperado: registrei apenas 3,6% de MFU (Model FLOPs Utilization). Esse índice, extremamente baixo,

gerou dúvidas não apenas sobre a eficiência do código, mas também sobre a própria metodologia de aferição.

Na **Semana 7**, a avaliação do checkpoint gerado no experimento anterior revelou resultados insatisfatórios: o modelo falhou até mesmo em generalizar estruturas básicas da linguagem. Diante dessa limitação de convergência e performance, decidi explorar o *framework* Megatron-LM. Durante a análise do código-fonte — cujas observações registrei no **Apêndice 5** —, deparei-me com a aplicação prática de todos os fundamentos teóricos estudados nas semanas iniciais. Identificar técnicas complexas de paralelismo e otimização de memória materializadas na arquitetura do software foi uma validação crucial do aprendizado teórico. Essa compreensão me permitiu configurar o ambiente corretamente e replicar os experimentos, resultando em um salto de eficiência para 44,8% de MFU.

Motivado pela otimização alcançada, dediquei a **Semana 8** à execução de um novo ciclo de pré-treinamento, desta vez fundamentado nas metodologias da Nvidia e na arquitetura do SmoLLM2 (detalhes no **Apêndice 6**). Optei por utilizar o tokenizador do próprio SmoLLM2 e um dataset em língua inglesa. Utilizando uma GPU Nvidia H100, o treinamento processou 50 bilhões de tokens em um modelo de 135 milhões de parâmetros em menos de três dias. A validação qualitativa, realizada através da geração de amostras (sampling), gerou expectativas de que o ecossistema estava funcionando. Constatar a evolução da modelagem da linguagem foi um dos marcos mais gratificantes desta Residência.

Na **Semana 9**, continuei com objetivo de validar a rotina de treinamento com o Megatron-LM, visando garantir que a qualidade dos checkpoints evoluísse consistentemente. As tentativas baseadas inicialmente apenas no cálculo de probabilidade (log-verossimilhança) em *datasets* de teste não demonstraram melhoras significativas. Diante disso, implementei uma metodologia alternativa inspirada no Alpaca Eval, adaptada para o estágio de pré-treino, que se comporta como um “torneio” comparativo entre os checkpoints. Essa abordagem qualitativa revelou uma concordância clara na evolução das respostas, conforme documentado no **Apêndice 7**. Com o código de treinamento validado, voltei o foco para o cenário nacional, analisando o modelo Tucano e o dataset GigaVerbo.

Inspirado pela metodologia do SmolLM2 e dos artigos da Nvidia, desenhei uma nova estratégia de treinamento. A proposta integrou o tokenizador em português, desenvolvido anteriormente, a um mix de dados composto pelo GigaVerbo, por *datasets* de matemática (em inglês) e por dados sintéticos de raciocínio (Chain-of-Thought) em português, com o intuito de permitir trabalhos futuros com Aprendizado por Reforço.

A **Semana 10** iniciou-se com um desafio de “engenharia”, a migração do checkpoint do Megatron-LM para o ecossistema *Hugging Face* Transformers. A ausência de uma arquitetura nativa compatível exigiu a reimplementação manual das camadas da rede neural utilizando PyTorch. Inicialmente, os resultados apresentaram divergências numéricas, o que demandou as sessões de debugging mais complexas de toda a minha jornada até o momento para alinhar a computação dos tensores. Superada essa barreira e após finalizar o treinamento com 300B de tokens treinados, 152.000 updates com 32% MFU, a validação final foi realizada através do *framework* LightEval, focando especificamente em *datasets* em português. Os resultados foram surpreendentes. O modelo, agora, “batizado” de Biatron de 345M de parâmetros (Biatron-345M), atingiu uma pontuação agregada de 0.2449, posicionando-se competitivamente muito próximo ao Tucano-630M (com 630M de parâmetros) e superando modelos renomados como o SmolLM2-360M, Qwen3-0.6B e Gemma-3-270M nas tarefas em nosso idioma. A tabela completa de benchmarks e a história sobre como o nome do modelo foi escolhido constam no **Apêndice 8**.

Gostaria de agradecer aos meus pais, por todo o apoio que sempre me deram. Obrigado pela paciência durante a etapa de pré-vestibular e pelos inúmeros sacrifícios feitos ao longo da minha formação, muitas vezes abrindo mão de seus próprios planos para priorizar o meu crescimento.

À minha namorada, Rhaquel, que caminhou ao meu lado durante todos os anos de graduação. Obrigado por suportar comigo os momentos difíceis e pela compreensão diante das minhas ausências.

Ao Professor Federson. Agradeço por me ensinar a observar problemas de forma holística e integrada, ampliando minha maneira de pensar e influenciando minha percepção.

E, por fim, a todos os amigos e professores que fizeram parte dessa jornada. Obrigado por cada conversa, troca de conhecimento, incentivo e companhia. Vocês tornaram essa caminhada muito mais leve e encantadora.

## APÊNDICE 1

## Termo de Aceite de Entrega

### Objetivo deste documento

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

**Data da Reunião (“Gate”) de aprovação:** 3 de set. de 2025

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

Daniel Fazzioni

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

Nessa Semana, as seguintes atividades foram investigadas:

- Tema central da residência:
  - **Maximizar a eficiência de treinamentos de grandes modelos em clusters de GPUs**
  - Não é um tema fechado, mas é uma direção!
- Estudo sobre fundamentos a partir de uma busca no Google:
  - [“how to train a neural network in gpu cluster”](#)
  - Leitura do primeiro resultado retornado:
    - [Training extremely large neural networks across thousands of GPUs.](#)
    - A partir dessa referência, iniciei a leitura do livro:
      - [The Ultra-Scale Playbook: Training LLMs on GPU Clusters](#)

[Anotações detalhadas da Semana 1](#)

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

- Finalizar a leitura do primeiro tópico (First Steps: Training on One GPU) do livro The Ultra-Scale Playbook
- Ler o artigo da OpenAI ( An Empirical Model of Large-Batch Training )
- Entender o conceito de “MFU”

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

## ACEITE DA ENTREGA:

CEDRIC LUIZ DE CARVALHO: [Go! ▾](#)

## Anotações detalhadas da Semana 1

Primeiro é importante limitar um pouco o tema... o foco ou objeto de estudos é sobre maximizar a eficiência de treinamentos de grandes modelos em clusters de GPUs (Graphics Processing Unit), mas para aprender a otimizar. Na verdade, seria interessante aprender a treinar um modelo em um cluster, não é mesmo?

Nossa primeira busca na web é: "[how to train a neural network in gpu cluster](#)"

Algumas definições que cercam o assunto:

- FLOPs (floating point operations)
- Paralelismo de dados
- Paralelismo de tensor
- Paralelismo de pipeline

O primeiro resultado da busca<sup>3</sup> começa discutindo sobre o tamanho do *batch size* e cita o artigo An Empirical Model of Large-Batch Training<sup>4</sup> que exclama: "A partir de um determinado tamanho de *batch size* a computação se torna ineficiente!" (ver Figura 1.1)

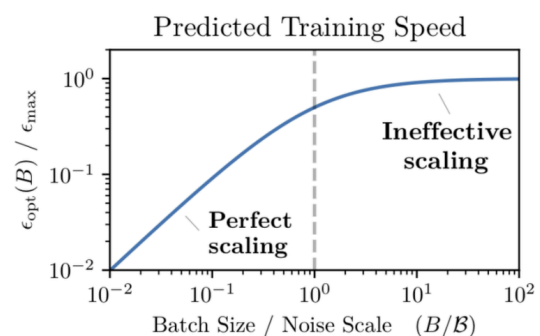


Figura 1.1 Relação entre *batch size* e velocidade de treinamento.

<sup>3</sup>JORDAN, Jeremy. Distributed training. Disponível em <https://www.jeremyjordan.me/distributed-training/>. Acesso em: 29 nov. 2025.

<sup>4</sup>MCCANDLISH, Sam et al. An empirical model of large-batch training. 2018. Disponível em: <https://arxiv.org/abs/1812.06162>. Acesso em: 29 nov. 2025.

Além disso, o *batch size* também pode aumentar durante o pré-treinamento de um LLM em função do stage do pré-treino, por exemplo, o [Llama 3.1 405B](#)<sup>5</sup>.

Também parece ser importante entender como funciona cada paralelismo, pois no caso do Llama, com computação disponível, foi utilizado mais de um paralelismo.

Algumas ferramentas começam a aparecer para entender o comportamento do treino:

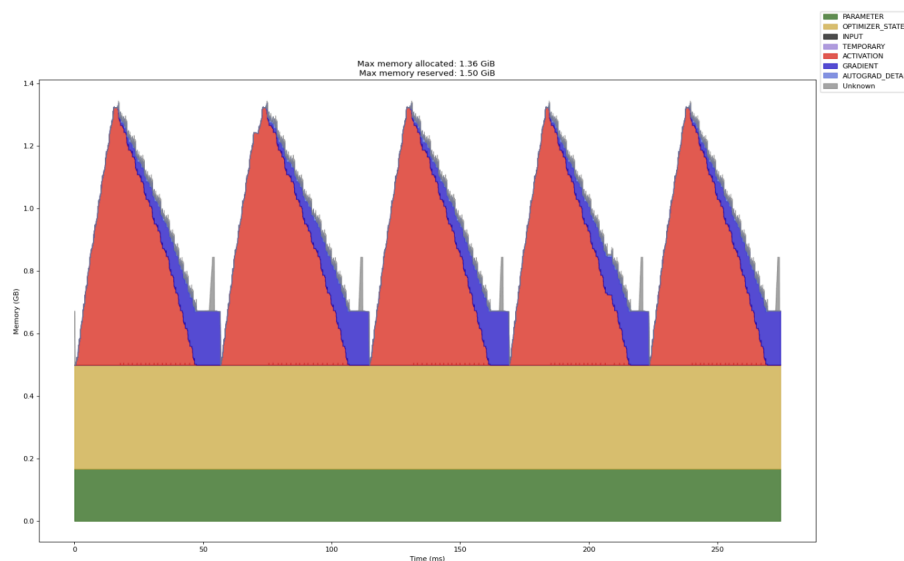


Figura 1.2: O *Torch Profiler*<sup>6</sup> permite monitorar o uso da memória durante o treinamento.

As referências deste site são interessantes, esse livro parece ser promissor: [The Ultra-Scale Playbook: Training LLMs on GPU Clusters](#), ele servirá de guia nessa primeira etapa.

<sup>5</sup> DUBEY, Abhimanyu et al. **The Llama 3 Herd of Models**. 2024.. Disponível em: <https://arxiv.org/abs/2407.21783>. Acesso em: 29 nov. 2025.

<sup>6</sup> PyTorch. torch.profiler. Disponível em: <https://docs.pytorch.org/docs/stable/profiler.html>. Acesso em: 29 nov. 2025.

## [The Ultra-Scale Playbook: Training LLMs on GPU Clusters.](#)

O material se estrutura sob 3 pilares:

- Introdução rápida sobre conceitos teóricos
- Implementação limpa de código: [referência com repositórios educacionais](#)
- Referências reais de benchmark de treinamento

No primeiro capítulo o livro explora o treinamento em uma GPU e traz alguns conceitos. Há uma citação para esse artigo: [MiniMax-01: Scaling Foundation Models with Lightning Attention](#) em que os autores gostariam de criar um modelo em que melhorasse o processamento de um contexto longo. O artigo argumenta que os MoE (Mixture of Experts) são uma forma de aumentar a eficiência entre parâmetro e capacidade de computação.

### **Estamos em um SISTEMA!**

**Aumentar a eficiência do treinamento também depende da arquitetura neural e dos hiperparâmetros (ex. *batch size*)!**

As camadas de atenção dos transformers impactam durante o treinamento, inclusive existem *kernels* específicos para aumentar o desempenho.

Na parte 4.2 do livro, há um artigo que discute sobre o *batch size*: [An Empirical Model of Large-Batch Training](#) o qual evidencia vários experimentos, mas para entender a intuição, será necessário ler o artigo da OpenAI.

- Será necessário intuir sobre o uso e comportamento da memória durante o treinamento:
  - Memória do modelo
  - Memória dos gradientes
  - Memória para o otimizador

- Memória para computar a ativação dos gradientes
- Os *kernels* também ocupam espaço na VRAM (< 1 GB)

Agora precisamos adicionar mais um fator, a precisão / quantização da memória e parece que há um capítulo no livro sobre isso.

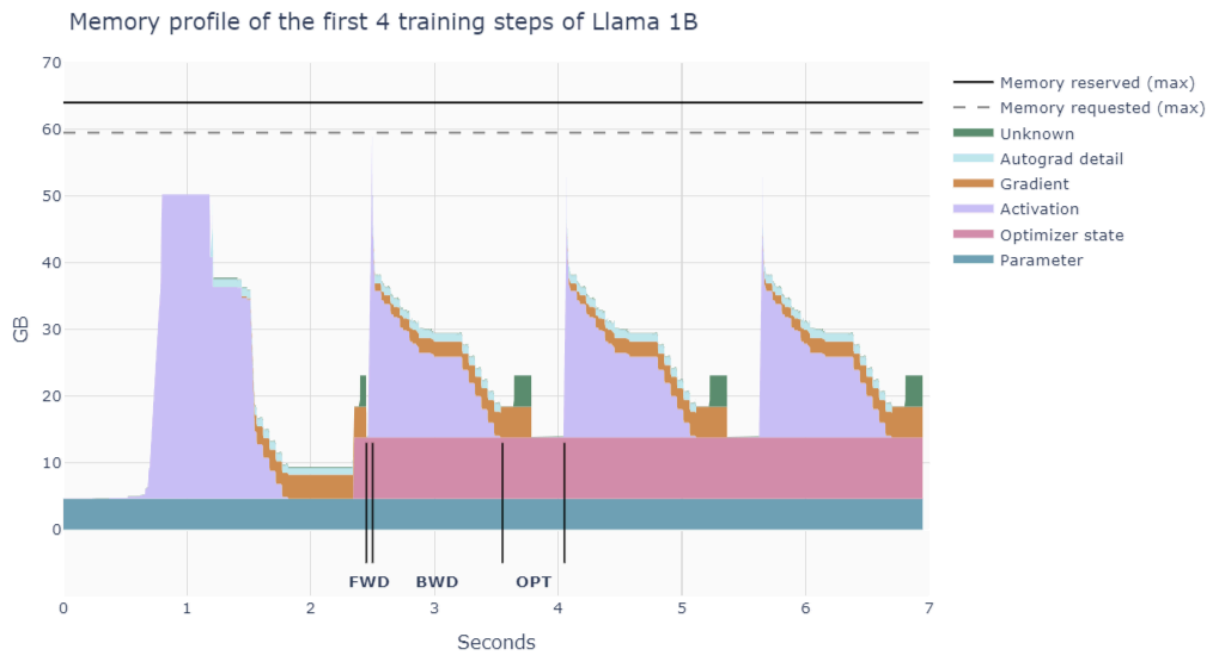


Figura 1.3. Exemplo de um comportamento de memória durante o treinamento, observe que no primeiro momento o comportamento é diferente, começa o cache do PyTorch (<https://zdevito.github.io/2022/08/04/cuda-caching-allocator.html>) para evitar *cudaMalloc*. Nesse gráfico, no *feedforward* (FWD) as ativações crescem rápido e durante o *backward* os gradientes são construídos e progressivamente a memória das ativações é liberada. Após isso o estado do otimizador é alterado em função de todos os gradientes calculados.

Uma observação interessante, o uso da memória cresce linearmente em função do *batch size* e quadrática em função do tamanho da sequência (*seq len*) conforme a Figura 1.4.

Uma observação relevante é que o mecanismo de atenção possui complexidade  $O(n^2)$  onde  $n$  é o *seq\_len*

Aumentar o tamanho da sequência de cada amostra, aumenta quadraticamente o uso de memória!

Então, o tamanho da sequência também influencia drasticamente este sistema!

Esta é a parte que cresce quadraticamente quando treinamos com longas sequências!

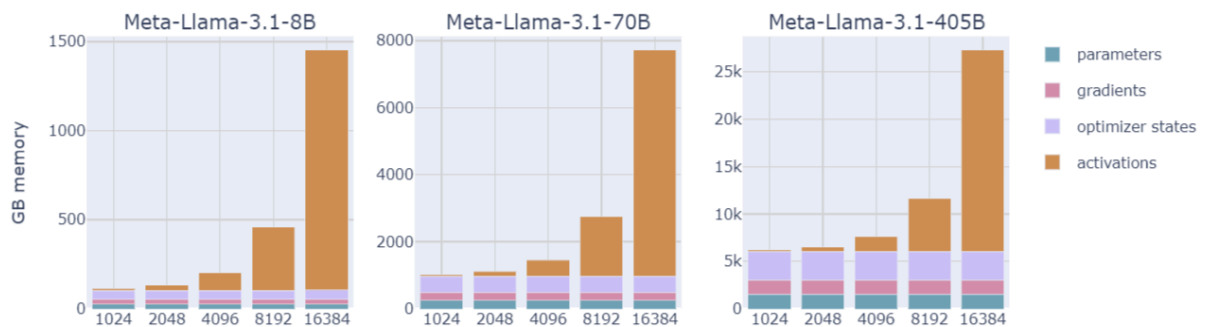


Figura 1.4. Utilização da VRAM em função do tamanho da sequência.

## APÊNDICE 2

## Termo de Aceite de Entrega

### Objetivo deste documento

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

**Data da Reunião (“Gate”) de aprovação:** 11 de set. de 2025

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

Daniel Fazzioni

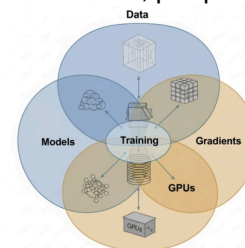
**Entrega:** [descrever a ENTREGA - requisitos e produtos gerados: links para textos, códigos, vídeos etc.]

Relembrando a Semana anterior, estou buscando/desejando entender sobre:

### Como treinar grandes modelos em clusters de GPUs de modo eficiente?

Primeiro, desejo esclarecer alguns detalhes: Não é apenas um treinamento distribuído, porque o tema em interesse há algumas partes:

- No mínimo um modelo neural com bilhões de parâmetros.
- Grande quantidade de dados para serem treinados.
- O modo de treinamento é por meio de gradiente.
- Múltiplas GPUs envolvidas.



Embora o interesse seja na interseção, é um desafio justificar sabiamente qual a ordem das partes deveria ser compreendida sobre a ótica do tema escolhido, então eu preferi, nesse momento, delegar para o livro [The Ultra-Scale Playbook: Training LLMs on GPU Clusters](#)

Finalizei o primeiro capítulo do livro e aprendi sobre:

- Como a memória (VRAM) deveria se comportar no treinamento, sobre os gargalos e a influência de alguns hiperparâmetros em função do modelo na memória, complexidade do algoritmo, etc.

Não consegui finalizar as atividades planejadas dessa semana:

O artigo "[An Empirical Model of Large-Batch Training](#)" não foi concluído. (Uma das páginas foi necessário ler mais de 10x!). - [anotações](#).

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

- Compreender o artigo proposto
- Criar alguns experimentos para comprovar as hipóteses de uso de memória.
- Entender o conceito de “MFU”

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

---

## ACEITE DA ENTREGA:

CEDRIC LUIZ DE CARVALHO: [Go! ▾](#)

## Termo de Aceite de Entrega

### Objetivo deste documento

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

**Data da Reunião (“Gate”) de aprovação:** 17 de set. de 2025

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

Daniel Fazzioni

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

Relembrando a Semana anterior, estou buscando/desejando entender sobre:

**Como treinar grandes modelos em clusters de GPUs de modo eficiente?**

{ LLM  $\cap$  Dados  $\cap$  Gradiente  $\cap$  GPU }

Nessa Semana, o foco foi com o Gradiente:

- Finalizei a leitura do artigo [An Empirical Model of Large-Batch Training \(OpenAI, 2018\)](#)
- Esse artigo mudou a forma de pensar sobre o gradiente.
- Tive algumas hipóteses: Dataset com múltiplas tarefas gera um gradiente mais ruidoso?

Mas também aprendi a utilizar o [PyTorch Profiler](#):

- Permite uma compreensão melhor do treinamento
- Pode ajudar avaliar algumas implementações - overhead, tempo de operações, etc

Aprendi como calcular o MFU ( Model Flops Utilization)

[As anotações estão disponíveis aqui.](#)

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

- Leitura do próximo capítulo (Data Parallelism) do livro [The Ultra-Scale Playbook: Training LLMs on GPU Clusters](#)

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

---

---

## ACEITE DA ENTREGA:

CEDRIC LUIZ DE CARVALHO: Go! ▾

## Anotações da Semana 2

### An Empirical Model of Large-Batch Training

Nesse artigo, estamos interessados em entender sobre como encontrar o *batch size* adequado para o treinamento, existe um ponto a partir do qual não existe mais ganho de informação ao aumentar o *batch size*.

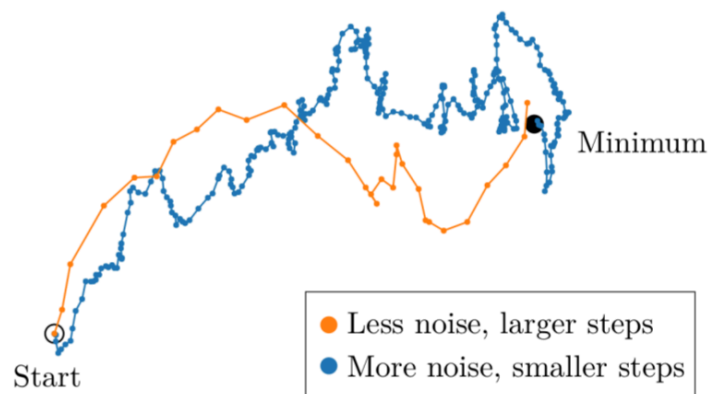


Figure 2: Less noisy gradient estimates allow SGD-type optimizers to take larger steps, leading to convergence in a smaller number of iterations. As an illustration, we show two optimization trajectories using momentum in a quadratic loss, with different step sizes and different amounts of artificial noise added to the gradient.

Quando o *batch size* é grande, pode-se utilizar uma *learning rate* mais alta e quando o *batch size* é pequeno, deve-se utilizar uma *learning rate* inferior. Em outras palavras, o *batch size* e a *learning rate* deveriam estar ligados!

O artigo mostra que a variância no gradiente estimado é inversamente proporcional ao *batch size*.

- Uma alta variância significa uma medição muito ruidosa, muita incerteza sobre a direção, apontando para vetores diferentes entre batches.
- Baixa variância evidencia uma maior estabilidade, próxima do gradiente verdadeiro.

Não apenas isso, mas também explora a ideia de inferir um ruído a partir do gradiente.

Equação 1: Cálculo do gradiente estimado: definido como a média do gradiente de B amostras.

$$G_{\text{est}}(\theta) = \frac{1}{B} \sum_{i=1}^B \nabla_{\theta} L_{x_i}(\theta); \quad x_i \sim \rho$$

O gradiente estimado pode ser definido pela Equação 1, quando B (tamanho do *batch size*) tende ao tamanho do dataset esse se torna o gradiente verdadeiro, ou seja, o gradiente de todos os dados.

Ao considerar que a variância do gradiente é inversa a B:

$$\begin{aligned} \mathbb{E}_{x_1 \dots x_B \sim \rho} [G_{\text{est}}(\theta)] &= G(\theta) \\ \text{COV}_{x_1 \dots x_B \sim \rho} (G_{\text{est}}(\theta)) &= \frac{1}{B} \Sigma(\theta), \\ \Sigma(\theta) &\equiv \text{COV}_{x \sim \rho} (\nabla_{\theta} L_x(\theta)) \\ &= \mathbb{E}_{x \sim \rho} \left[ (\nabla_{\theta} L_x(\theta)) (\nabla_{\theta} L_x(\theta))^T \right] - G(\theta) G(\theta)^T. \end{aligned}$$

Equação 2: Covariância do Gradiente estimado.

Não seria interessante estimar a variância do gradiente em função de B? Ou utilizar isso para escolher um bom B?

$$L(\theta - \epsilon V) \approx L(\theta) - \epsilon G^T V + \frac{1}{2} \epsilon^2 V^T H V.$$

Equação 3. Estimando um passo “perfeito”, em que H é a hessiana e V um vetor (pode ser visto como o state de alguns otimizadores)

$\mathbb{E}[L(\theta - \epsilon G_{\text{est}})]$  with respect to  $\epsilon$ . This expected value can be evaluated using Equation 2.2:

$$\mathbb{E}[L(\theta - \epsilon G_{\text{est}})] = L(\theta) - \epsilon |G|^2 + \frac{1}{2} \epsilon^2 \left( G^T H G + \frac{\text{tr}(H \Sigma)}{B} \right). \quad (2.5)$$

Minimizing this equation with respect to  $\epsilon$  leads to:

$$\epsilon_{\text{opt}}(B) = \underset{\epsilon}{\text{argmin}} \mathbb{E}[L(\theta - \epsilon G_{\text{est}})] = \frac{\epsilon_{\text{max}}}{1 + \mathcal{B}_{\text{noise}}/B} \quad (2.6)$$

as the optimal step size, which produces an optimal improvement in the loss from the noisy gradient:

$$\Delta L_{\text{opt}}(B) = \frac{\Delta L_{\text{max}}}{1 + \mathcal{B}_{\text{noise}}/B}; \quad \Delta L_{\text{max}} = \frac{1}{2} \frac{|G|^4}{G^T H G}. \quad (2.7)$$

Above, we have defined the *noise scale* as:

$$\mathcal{B}_{\text{noise}} = \frac{\text{tr}(H \Sigma)}{G^T H G}, \quad (2.8)$$

Figura 2.1. O problema da equação 3 é que não temos o gradiente verdadeiro, então ao usar o gradiente estimado e considerar  $G=V$  encontramos uma definição de ruído do gradiente.

O problema é que calcular a hessiana em modelos grandes é caro e difícil de calcular, a partir de algumas suposições “irrealistas” a equação acima é reduzida para a equação 4.

$$\mathcal{B}_{\text{simple}} = \frac{\text{tr}(\Sigma)}{|G|^2},$$

Equação 4. Fórmula de calcular o ruído do gradiente simplificada. Em outras palavras, a escala de ruído é igual à soma das variâncias dos componentes individuais do gradiente dividido pela norma global do gradiente, a ideia é comparar o quão grande o gradiente é comparado com a sua variância.

## Anotações da Semana 3

### An Empirical Model of Large-Batch Training

O tamanho do *batch size* também deveria ser relacionado com a complexidade da tarefa ou a necessidade de existirem muitas tarefas no dataset, como acontece em NLP. Nesse sentido, é esperado que a magnitude do gradiente diminua à medida que o treinamento avança. Lembrando da definição anterior, o ruído é comparado com a magnitude do gradiente, portanto, é um indício de que à medida que o treinamento acontece e o gradiente diminui, o *batch size* deveria aumentar.

O artigo também traz um ponto interessante, quanto maior a quantidade de tarefas maior o ruído?

No apêndice A1 do artigo, há uma ideia de como utilizar essa ideia de modo prático durante o treinamento... Em um treinamento multi-GPUs, cada mini-batch se torna uma pequena medição, se entre os batches existe muita variância, então, talvez, tenhamos um ruído. No artigo, mostra que o cálculo do ruído simplificado, na maioria dos casos, fica acima do cálculo do ruído, o que evidencia uma boa métrica de monitoramento do treinamento.

Outro ponto que o artigo traz é a utilização de um batch dinâmico.

Sabemos que, para calcular a norma do gradiente de uma rede neural, podemos empilhar todas as normas e calcular a norma do vetor de 1D.

Há uma implementação no [github](#)<sup>7</sup>, como métrica auxiliar para utilizar no treinamento, parece estar adequada, mas há métodos mais seguros para manipulação dos tensores de gradiente.

#### Dependência do tamanho do Modelo

Mesmo que nas equações não haja dependência do número de parâmetros, parece que modelos maiores possuem um “noise” maior, considerando uma LSTM, como ilustrado na Figura 8 do artigo.

---

<sup>7</sup> SINGH, Shreyansh. An Empirical Model of Large-Batch Training. Disponível em: [https://github.com/shreyansh26/An-Empirical-Model-of-Large-Batch-Training/blob/master/noise\\_scale.py](https://github.com/shreyansh26/An-Empirical-Model-of-Large-Batch-Training/blob/master/noise_scale.py). Acesso em: 29 nov. 2025.

## Temperatura

O artigo traz uma definição nova de temperatura, refletindo a ideia de que a *loss* é uma função de energia potencial:

Temperatura = *learning rate* / (máximo possível da *learning rate* com *batch size* = B)

Em outras palavras, a relação entre a *learning rate* escolhida e o tamanho máximo possível da LR para não violar o gradiente descendente. O artigo traz a figura C.2.

$$\mathcal{B}_{\text{noise}} \propto \mathcal{B}_{\text{simple}} \propto \frac{1}{T}. \quad (\text{C.2})$$

Figura C.2: Ruído e Ruído Simplificado são proporcionais, enquanto que a Temperatura é inversamente proporcional.

Essa equação delega o “ruído” do tamanho do batch, discutido anteriormente, para a *learning rate*, de modo que fique mais fácil ajustar os hiperparâmetros do treinamento, pois agora o ruído do gradiente depende da *loss* e da LR.

Quando o modelo está mais próximo de algum ótimo local o LR máximo possível é menor. Então, de modo útil, essa equação está mostrando que, possivelmente, no final do treinamento haverá um ruído maior no gradiente.

Também há uma menção a uma observação importante, alguns experimentos mostram que aumentar o *batch size* ou diminuir a *learning rate* diminuem a Temperatura e, portanto, o ruído do gradiente.

Por fim, a abordagem sugestiva dos autores é a seguinte:

Em vez de se preocupar em encontrar um schedule ideal para “esfriar” a temperatura de treinamento:

“Primeiro, decida qual nível de ruído (ou “temperatura”) é bom para o seu treinamento. Uma vez que você definiu isso, a métrica (“escala de ruído”) lhe dirá exatamente qual *batch size* você deve usar para atingir e manter esse nível de ruído.”

## Teste de Memory profile

Fiz um teste rápido com a ferramenta PyTorch Profiler, essa ferramenta vai ser interessante para avaliar mudanças e garantir se a memória está se comportando como esperado, bem como avaliar possíveis gargalos no treinamento.

Interessante que o navegador Google Chrome possui uma forma fácil de visualizar os registros em: <chrome://tracing/>. Abaixo estão a Figura 3.1 resultante e código, respectivamente:

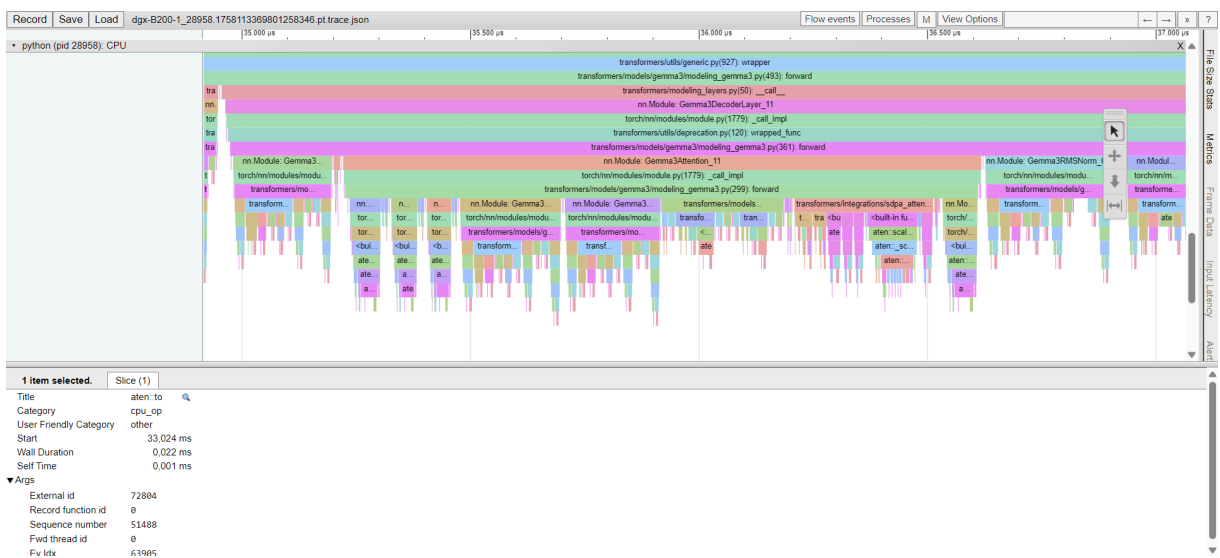


Figura 3.1. Registros do Torch Profiler.

### Código Gerado:

```
from torch.profiler import profile, record_function, ProfilerActivity, tensorboard_trace_handler
opt = torch.optim.AdamW(model_instruct.parameters(), lr=2e-5)
with profile(
    activities=[ ProfilerActivity.CPU, ProfilerActivity.CUDA ],
    schedule=torch.profiler.schedule(wait=1, warmup=1, active=3, repeat=1), # Agenda para ignorar o warmup
    on_trace_ready=torch.profiler.tensorboard_trace_handler('./logs/mem_profile'), # Salva os logs para o TensorBoard
    profile_memory=True, # Habilita o profiling de memória
    with_stack=True # Grava a pilha de chamadas para rastrear a origem das alocações
) as prof:
    for epoch in range(1):
        model.train()

        for step, batch in enumerate(range(10)):
            opt.zero_grad()
            outputs = model(**input_ids.to(device))
            outputs.loss.backward()
            opt.step()
            prof.step() # Informa ao profiler que o passo terminou
```

## MFU

Para rápido entendimento, utilizei como referência esse [post](#)<sup>8</sup> de um ex-engenheiro da META. MFU significa Model Flops utilization ou, em outras palavras, do total de operações de ponto flutuante (FLOPS) necessárias para treinar o modelo, quanto realmente foi utilizado pelo hardware? Essa equação também pode ser simplificada conforme a Equação 3.2.

$$MFU = \frac{\text{Model FLOPs per token} \times \text{Observed tokens per second}}{\text{Theoretical peak FLOPs of the hardware}}$$

Equação 3.2 para cálculo simplificado do MFU.

Para calcular essa métrica, precisamos encontrar:

1. Taxa de transferência média de tokens/segundo durante o treinamento
2. Calcular o número de FLOPS necessário para a arquitetura do modelo, em função de TOKENS (observe que os LLM dependem da seq\_len por causa da *Attention*)
3. Descobrir o máximo teórico de FLOPS para o tipo de precisão utilizada no treinamento.

$$MFU = (T / S) * (FLOPS / TOKEN) / FLOPS$$

Total de Flops, para arquitetura como transformers  $\approx$  mat\_muls (projeções Q,K,V,O) + mat\_muls(camadas MLP) + mat\_muls (projeção de vocabulário)

Observe que parâmetros de normalização ou embeddings não são utilizados.

Cálculo de Flops em Matrizes: 2x (número de parâmetros)

- 2 porque há um produto e uma soma

---

<sup>8</sup> RAY, Jaideep. Using Model Flops Utilization (MFU). Disponível em: <https://medium.com/better-ml/using-model-flops-utilization-mfu-7b17de07faec>. Acesso em: 29 nov. 2025.

Aproximação *feedforward* = 2 \* Números de parâmetros

Aproximação *backward* = 2 \* *feedforward*

Aproximação para LLM:

$MFU \approx (\text{Tokens/s} \times 6 \times \text{Número de Parâmetros}) / \text{máx.flops}$

## APÊNDICE 3

## Termo de Aceite de Entrega

### Objetivo deste documento

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

**Data da Reunião (“Gate”) de aprovação:** 25 de set. de 2025

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

Daniel Fazzioni

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

Relembrando as Semanas anteriores, estou buscando/desejando entender sobre:

#### Como treinar grandes modelos em clusters de GPUs de modo eficiente?

{ LLM U Dados U Gradiente U GPU }

**Semana 1:** Encontrei o livro: [The Ultra-Scale Playbook: Training LLMs on GPU Clusters](#), 2025)

- Li sobre o treinamento em 1 GPU ( Memória: pesos, ativações, gradiente, otimizadores)

**Semana 2 e 3:** Li 1 Artigo sobre *batch size* e ruído do gradiente ( Mudei minha percepção sobre  $\nabla$ )

**Semana 4:**

Iniciei a leitura do capítulo Data Parallelism do livro.

- Pensei que era apenas um capítulo sobre DDP, mas o livro surpreendeu com algumas otimizações, como a sincronização do gradiente entre GPUs.
- Também me fez pensar: “PyTorch como orientado a eventos”
- Construção do paralelismo 1D - O livro conta que existem 5.
- Cenário com muitas GPUs: Trade-off entre processamento e comunicação.  
O livro já deixou claro: *Cada cluster diminui +/- 6% de throughput*
- Nunca tinha entendido o motivos dos métodos ZeRO (Deep Speed) terem esse nome.

[Anotações da Semana 4.](#)

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

- Leitura do próximo capítulo (Tensor Parallelism) do livro [The Ultra-Scale Playbook: Training LLMs on GPU Clusters](#)

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

---

---

## ACEITE DA ENTREGA:

CEDRIC LUIZ DE CARVALHO: Go! ▾

## Termo de Aceite de Entrega

### Objetivo deste documento

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

**Data da Reunião (“Gate”) de aprovação:** 2 de out. de 2025

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

Daniel Fazzioni

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

### Como treinar grandes modelos em clusters de GPUs de modo eficiente?

{ LLM U Dados U Gradiente U GPU }

**Semana 1:** Encontrei o livro: [The Ultra-Scale Playbook: Training LLMs on GPU Clusters](#)

- Li sobre o treinamento em 1 GPU ( Memória: pesos, ativações, gradiente, otimizadores)

**Semana 2 e 3:** Li 1 Artigo sobre *batch size* e ruído do gradiente ( Mudei minha percepção sobre  $\nabla$  )

**Semana 4:** Entendi sobre Data Parallelism e os métodos ZeRO.

**Semana 5:** Nessa Semana, entendi sobre:

- Paralelismo de Tensor
  - Paralelismo de Sequência
- Paralelismo de Contexto
  - Problema na Atenção!
- Paralelismo de Pipeline
  - Solução do DeepSeek vs LLama 3.1
- Paralelismo de Experts ( **não é agnóstico ao modelo** )

Esses métodos também podem ser combinados com os Stages 1, 2 e 3 do ZeRO - (dado suas especificidades).

[Anotações!](#)

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

- Finalizar o último capítulo do Livro.
- Explorar o Report Técnico de alguns alguns modelos ( DeepSeek, Gemini, etc ) com o intuito de entender sobre os hiperparâmetros, *datasets*, e como foi otimizado no pré-treino.

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

Um ponto de atenção:

Alguns códigos deste livro sugerem o *framework* PICOTRON (mesma organização do autor)

Tomada de decisão:

- O livro sugere para testar algumas implementações para avaliar o MFU em alguns cenários. Mas, eu gostaria de guardar isso para outro momento.
- Parece que chegou o momento de aprofundar em um caminho... Estou curioso sobre: como justificar os hiperparâmetros de um treinamento?

**ACEITE DA ENTREGA:**

**CEDRIC LUIZ DE CARVALHO:** [Go!](#)

## Anotações da Semana 4

### Paralelismo de Dados

Interessado em replicar o modelo em cada GPU e propaga “micro-batches” em cada uma.

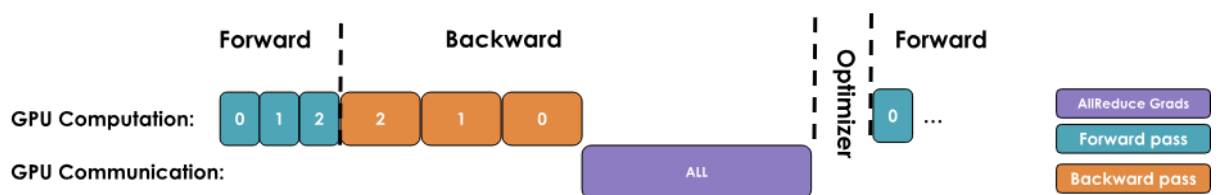


Figura 4.1: Modo ingênuo de sincronizar dados.

Na implementação “naive” o all-reduce é chamado após a propagação sequencial das camadas. O problema é que há um intervalo grande de computação ociosa enquanto é feito o all-reduce.

Há, pelo menos, 3 otimizações que podem ser aproveitadas:

Primeira Otimização: Sobreposição de gradiente enquanto faz o *backward*.

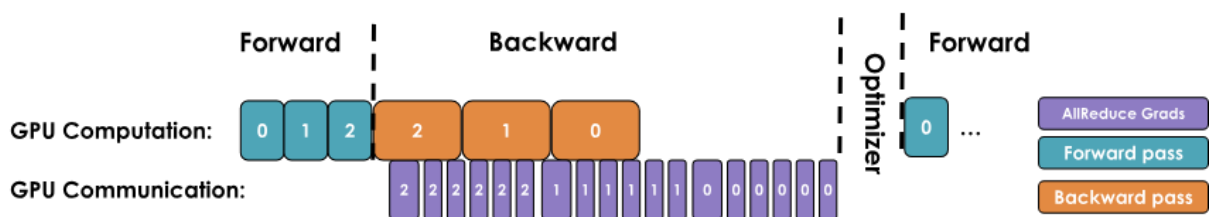


Figura 4.2: Sobreposição do compartilhamento de gradiente antes de finalizar o *backward*.

À medida que os gradientes forem sendo calculados, eles podem ser sincronizados, não há necessidade de aguardar o *backward* da primeira camada do modelo, conforme indica o uso de memória no capítulo anterior:

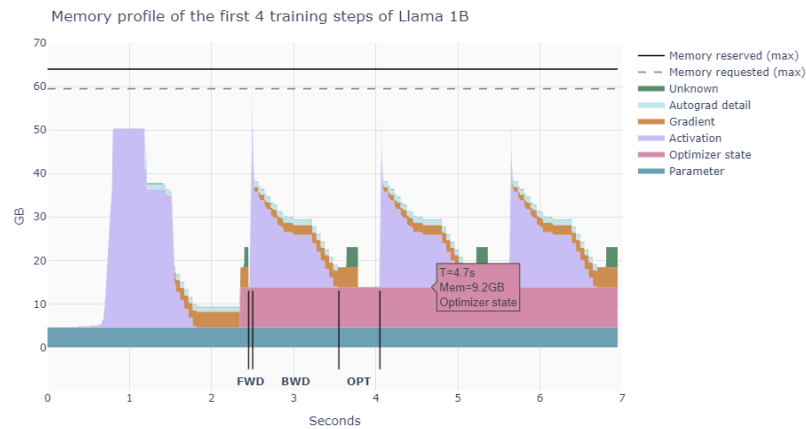


Figura 4.3: À medida que os gradientes forem computados, há a sincronização entre os workers e isso altera o uso de memória.

Uma forma simples de implementar isso é por meio dos hooks do PyTorch:

```
def register_backward_hook(self, hook):  
    """  
    Registers a backward hook for all parameters of the model that  
    require gradients.  
    """  
    for p in self.module.parameters():  
        if p.requires_grad is True:  
            p.register_post_accumulate_grad_hook(hook)
```

Código completo desta otimização:

```
class DataParallelNaive(nn.Module):  
    """  
    Naive Data Parallelism. Not used in practice. But it is a good starting point to understand how data parallelism  
    works.  
    It implements a simple all-reduce operation to synchronize gradients across multiple processes.  
    And `no_sync` context manager to disable gradient synchronization.  
    """  
    def __init__(self, module):  
        """  
        Initializes the DataParallel wrapper for a given module.  
        Args:  
            module (nn.Module): The model to be wrapped for data parallelism.  
            process_group (torch.distributed.ProcessGroup): The process group used for gradient synchronization.  
                It could be a data parallel or context parallel group.  
        """  
        super().__init__()  
        self.module = module  
        self.require_backward_grad_sync = True # whether to synchronize gradients during backward pass. Set to False when  
        using gradient accumulation  
        self.register_backward_hook(self._allreduce_grads)
```

```
def forward(self, *inputs, **kwargs):
    return self.module(*inputs, **kwargs)

def register_backward_hook(self, hook):
    """
    Registers a backward hook for all parameters of the model that require gradients.
    """
    for p in self.module.parameters():
        if p.requires_grad is True:
            p.register_hook(hook)

def _allreduce_grads(self, grad):
    """
    Performs an all-reduce operation to synchronize gradients across multiple processes.
    """
    # No synchronization needed during gradient accumulation, except at the final accumulation step.
    if self.require_backward_grad_sync:
        dist.all_reduce(grad, op=dist.ReduceOp.SUM, group=pgm.process_group_manager.cp_dp_group)
        grad /= pgm.process_group_manager.cp_dp_world_size
    return grad

@contextlib.contextmanager
def no_sync(self):
    """
    A context manager to temporarily disable gradient synchronization.
    This is useful for performing multiple backward passes during gradient accumulation without synchronizing
    gradients in between.
    """
    self.require_backward_grad_sync = False
    yield
    self.require_backward_grad_sync = True
```

Incrível! Com poucas linhas é possível otimizar desse modo.

### Segunda Otimização: Sincronizar um lote/bucket de gradientes

Operações com GPUs geralmente são mais eficientes quando envolvem grandes tensores de acordo com a Figura 4.4:

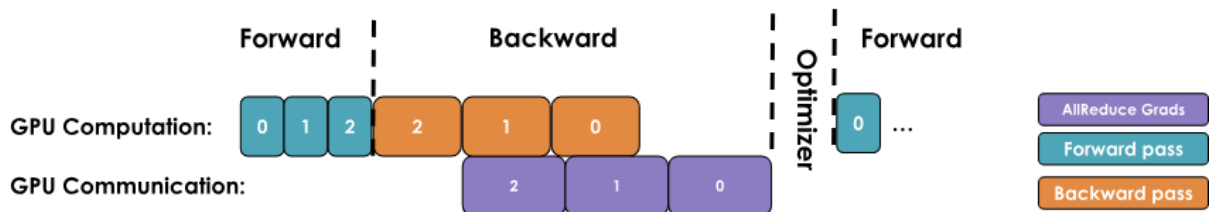
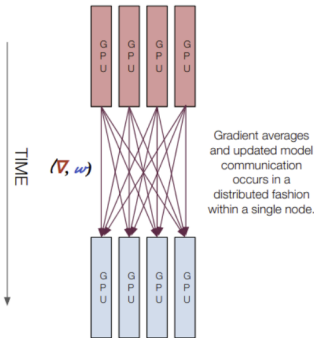
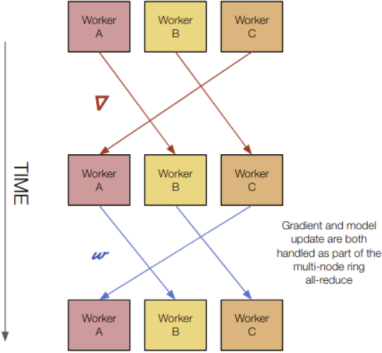


Figura 4.4: É preferível criar um vetor N do que sincronizar N escalares. Isso diminui o overhead de comunicação e aumenta a velocidade da operação da comunicação.

Terceira Otimização: Utilização de acumulação de gradiente.

Precisa desabilitar a sincronização do gradiente enquanto está acumulando gradiente.

A ideia é sincronizar a acumulação.

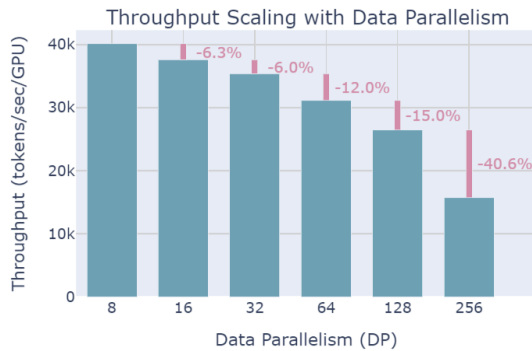
Existem alguns métodos diferentes de fazer o All-Reduce.	
<p>Multi GPU Training (Multi GPU all-reduce)</p>  <p>Gradient averages and updated model communication occurs in a distributed fashion within a single node.</p>	<p>Ring all-reduce Distributed Training</p>  <p>Gradient and model update are both handled as part of the multi-node ring all-reduce</p>
<p><b>Computation Happens:</b> On all GPUs <b>Gradient transfers:</b> GPU to GPU during NCCL all-reduce <b>Model transfers:</b> GPU to GPU during NCCL all-reduce</p>	<p><b>Computation Happens:</b> On all workers <b>Gradient transfers:</b> Worker transfers gradient to peers during all-reduce <b>Model transfers:</b> Model “update” happens at the end of multi-node all-reduce operation</p>
<p>Fonte: <a href="#">A Gentle Introduction to Multi GPU and Multi Node Distributed Training</a></p>	

Observe que no Ring all-reduce, há uma transferência muito mais eficiente, porque os gradientes são parcialmente somados em workers.

Na prática as pessoas sempre estão tentando aumentar mais modelos paralelos do que aumentando a acumulação de gradiente.

Distribuir o treinamento em várias amostras nos dá um paralelismo 1D ( será construído progressivamente mais 4 dimensões)

Como verificado na Figura 4.5, a cada cluster de GPUs, o *throughput* começa a diminuir significativamente.



Quando há amostras ou modelos grandes em que os parâmetros não cabem na GPU, podemos utilizar os métodos ZeRO.

Figura 4.5 *Throughput* em função da quantidade de nós.

## Zero Redundancy Optimizer (ZeRO)

Embora o DDP seja eficiente, não estamos replicando as memórias de ativação, otimizadores, gradiente, em todas as GPUs?  
(por isso, ZERO redundância)!

Essa abordagem pode ser dividida em 3 estágios:

ZeRO 1: Particionamento do Optimizer

ZeRO 2: Particionamento do Optimizer e do Gradiente

ZeRO 3: Particionamento do Optimizer, do Gradiente e dos pesos do modelo.

As ativações não são redundantes, cada GPU recebe um batch diferente e, portanto, as ativações também são diferentes.

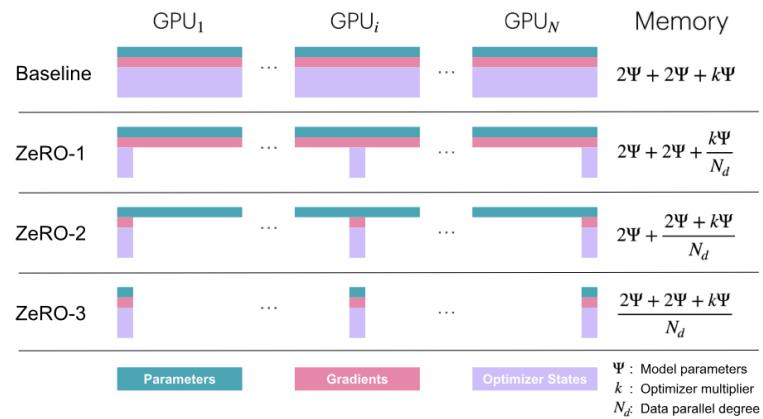
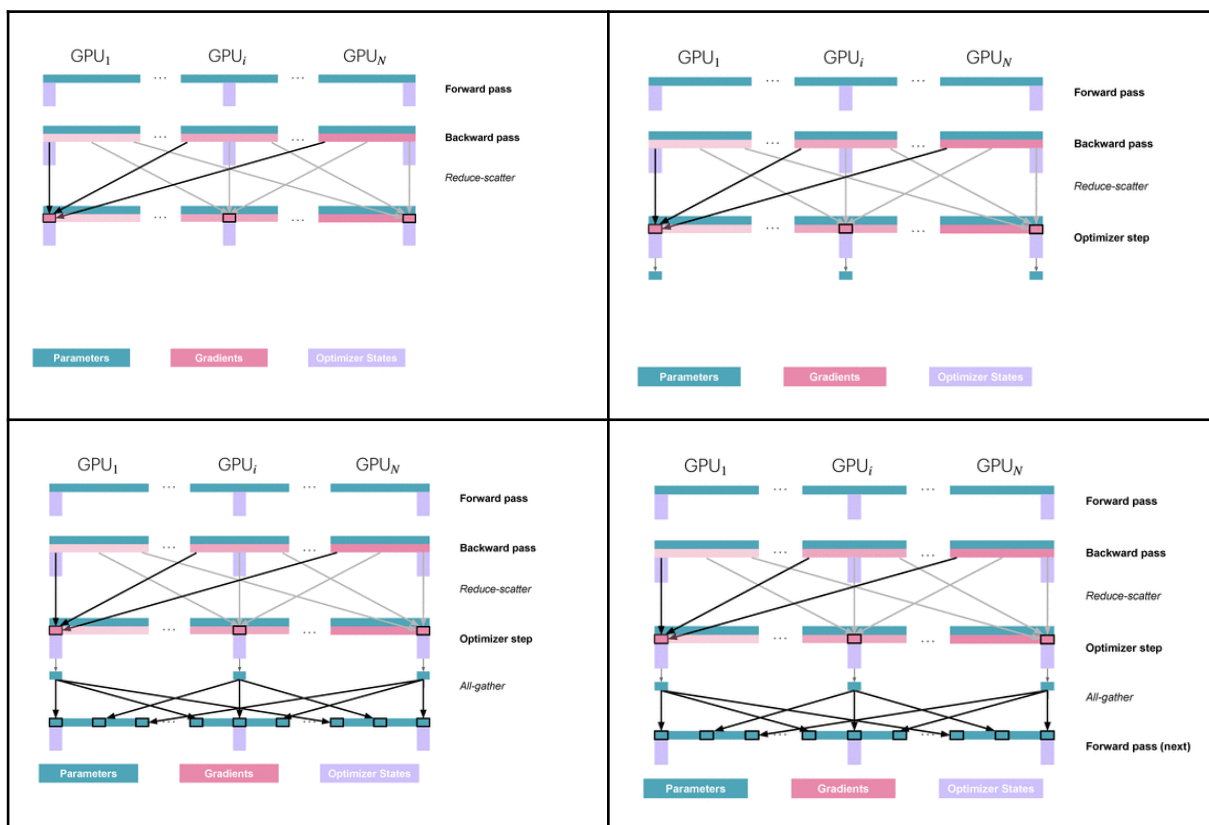


Figura 4.5: Visão geral do uso de memória dos métodos ZeRO.

### ZeRO-1: Particionamento dos Estados do Optimizer



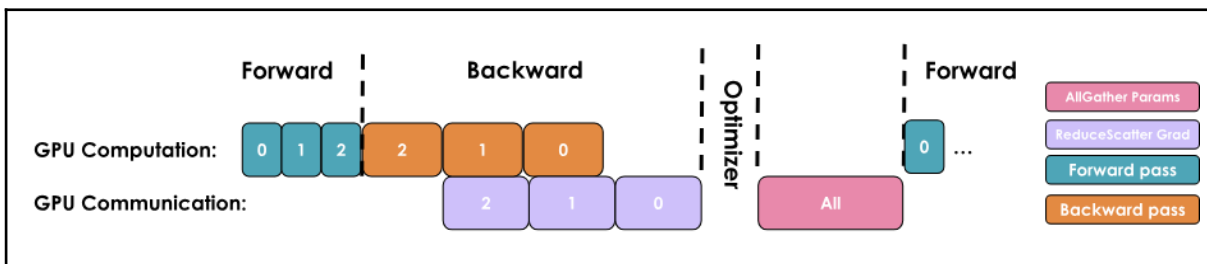


Tabela 4.1 - Conjunto de imagens que ilustra o processo de otimização do Método ZeRO 1.

Observe as imagens da tabela 4.1, tanto na passagem do *forward* quanto do *backward* não precisamos do Optimizer, mas para atualizar o optimizer, precisamos fazer um reduce-scatter do gradiente e então chamamos o step do otimizador. Como cada parte do otimizador está em GPUs diferentes, o otimizador vai corrigir só uma parte do modelo em cada GPU, então precisamos sincronizar esses novos pesos para que o modelo seja igual em todas as GPUs.

Além disso, podemos otimizar e sincronizar os gradientes antes de terminar o *backward*, e da mesma forma também podemos começar o all-gather imediatamente quando os gradientes de uma camada específica, por exemplo, chegam a uma GPU.

Essa implementação não é trivial, na prática, há a implementação nativa do ZeRO-3/FSDP do PyTorch e para usar definimos o FSDPUnit como o modelo inteiro.

Se observar atentamente, não precisa ter todos os gradientes em todas as GPUs...

### ZeRO-2: Adicionando o particionamento do Gradiente

Como cada réplica precisa ter o fragmento do gradiente para o otimizador, é interessante manter o mesmo split do otimizador e do gradiente juntos.

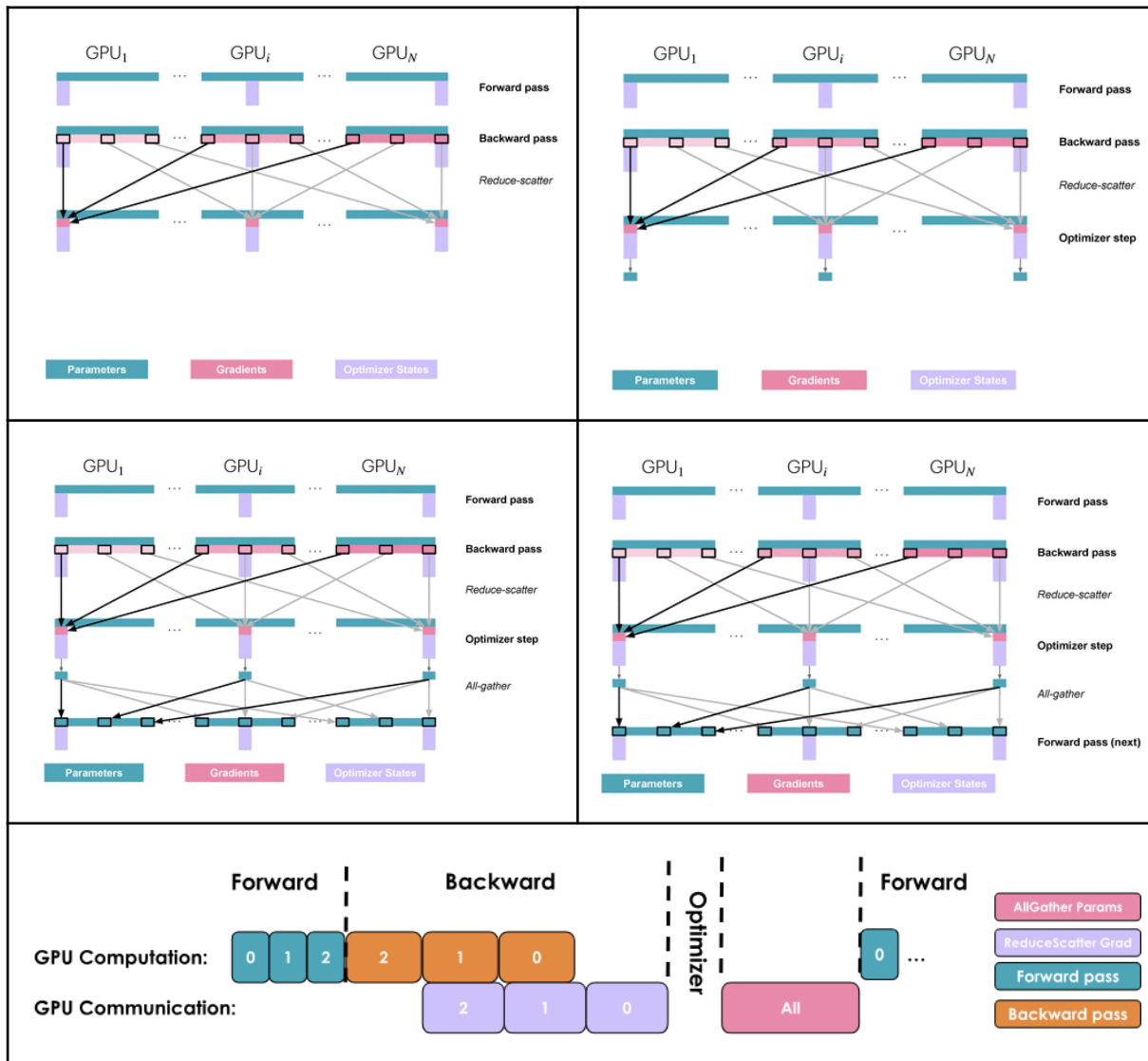


Tabela 4.2: Conjunto de imagens que ilustra o processo de otimização do Método ZeRO 2.

A diferença para o Stage-1 é que na prática precisamos computar o otimizador de uma partição com apenas o gradiente da partição.

### ZeRO-3: Adicionando o particionamento do Modelo (FSDP)

PyTorch, tem uma implementação nativa desse estágio: FSDP (Fully Sharded Data Parallelism).  
 FSDP (PyTorch) = ZeRO-3 (Deep Speed)

Como fazemos o *feedforward* e o *backward* se todos os parâmetros estão distribuídos?

Só reunir os parâmetros quando precisar deles (Figura 4.6) e remover da memória depois de utilizado, o fluxo do *backward* (Figura 4.7) funciona da mesma forma, construindo as partes do gradiente:

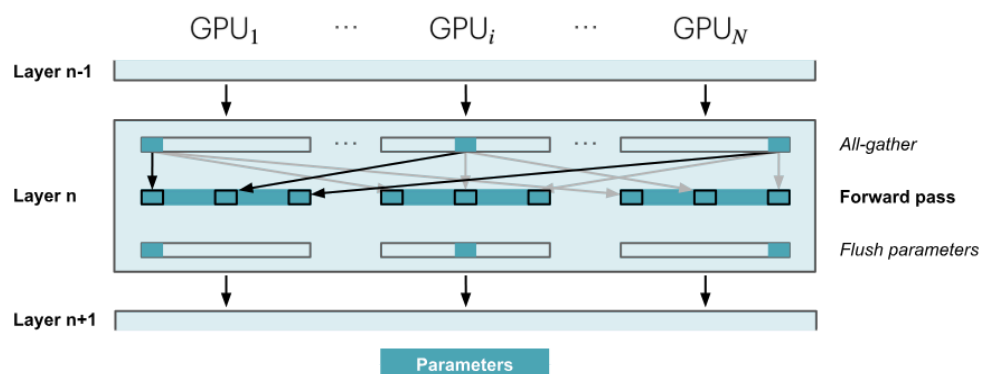


Figura 4.6: Compartilhamento de pesos durante o *forward*.

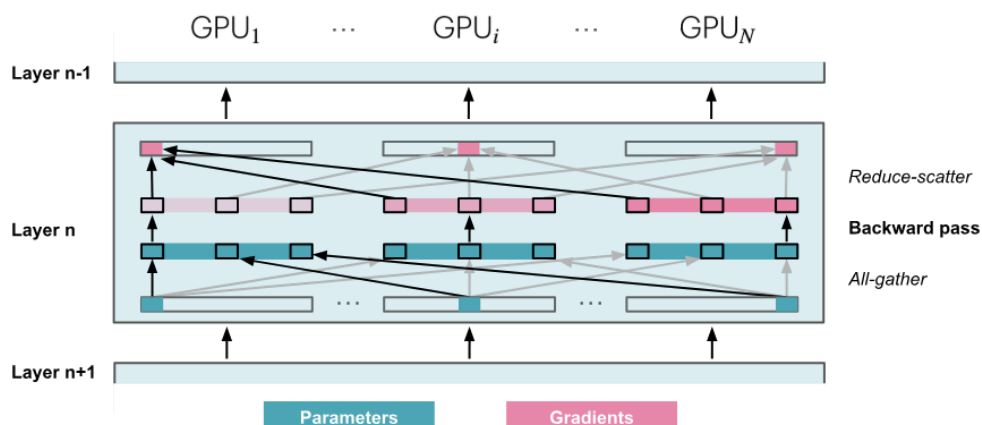


Figura 4.7: Compartilhamento dos gradientes.

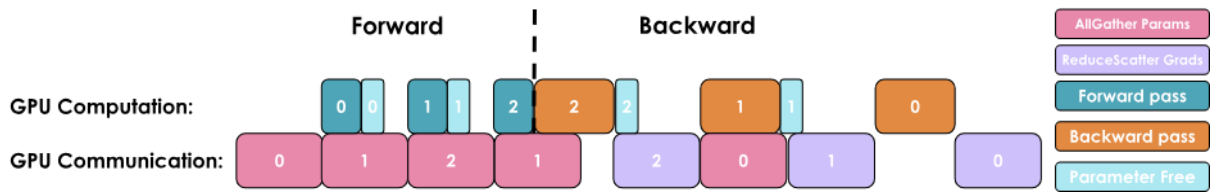


Figura 4.8: Transferência de dados no Método ZeRO-3.

Há muita comunicação nesse processo (Figura 4.8). Um custo de comunicação de  $3 * (\text{número de parâmetros})$ , comparado com  $2 * (\text{número de parâmetros})$  do Stage 2. Pode parecer um problema, mas na prática é possível otimizar para sobrepor os parâmetros da próxima camada com a camada atual com o intuito de ter os pesos para fazer as próximas multiplicações - Lei da Busca (não deve ultrapassar 512 GPUs)

Resumindo, o uso de DP pode escalar o nosso *throughput* enquanto os estágios ZeROs podem treinar um modelo (Figura 4.9) que não cabe em uma GPU com um pequeno custo de transferência.

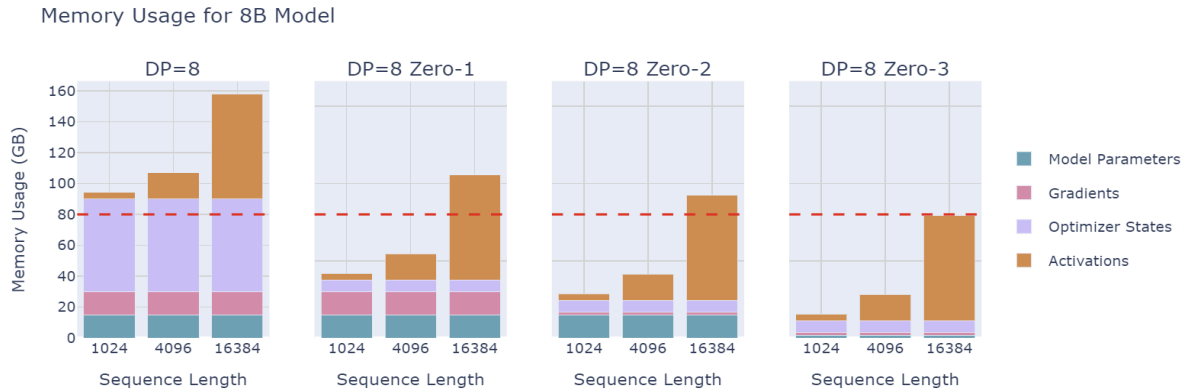


Figura 4.9 Comparação da memória utilizada entre os métodos ZeRO.

## Anotações da Semana 5

### Paralelismo de Tensor (TP) Nos Blocos do Transformers

Todas as operações nas dimensões do embedding poderiam ser computadas em GPUs diferentes. (Transformers)

Na prática qualquer matriz como a Figura 5.1 poderia ser decomposta como mostra a Figura 5.2 e também é possível decompor a matriz em ambos os métodos como a Figura 5.3

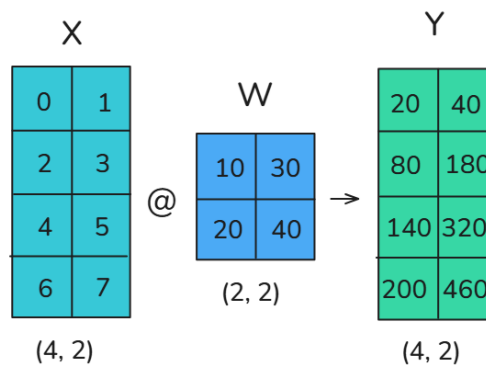


Figura 5.1: Exemplo de uma operação de multiplicação de matriz

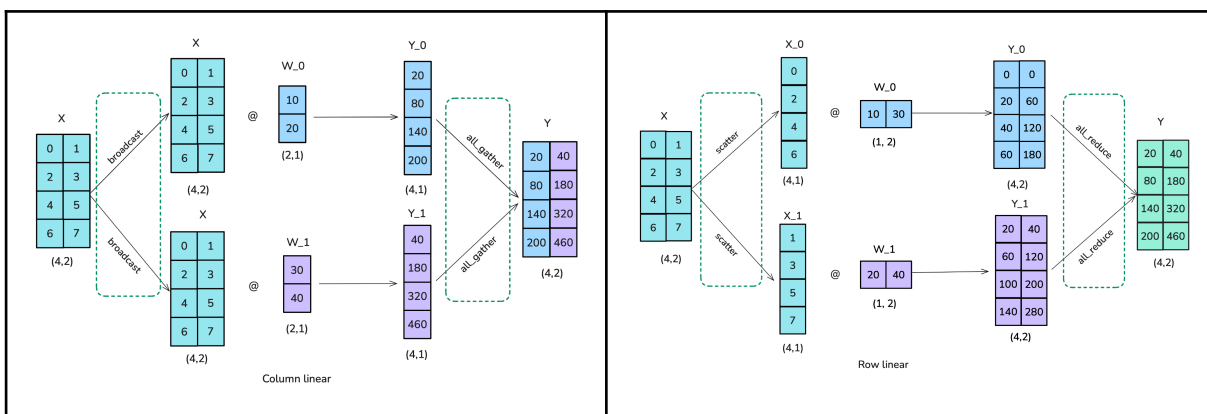


Figura 5.2: Exemplo de decomposição da multiplicação de matriz.

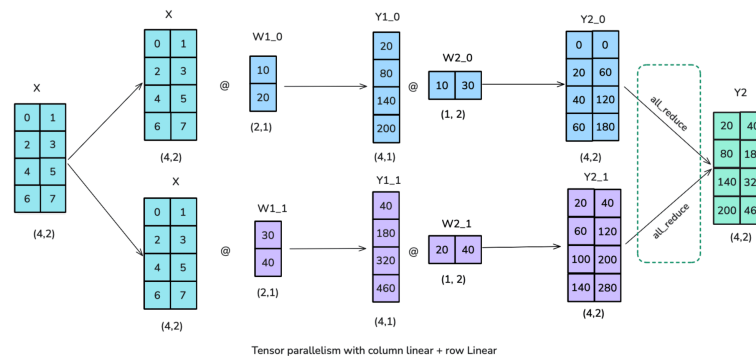


Figura 5.3: Decomposição em linha e em coluna ao mesmo tempo.

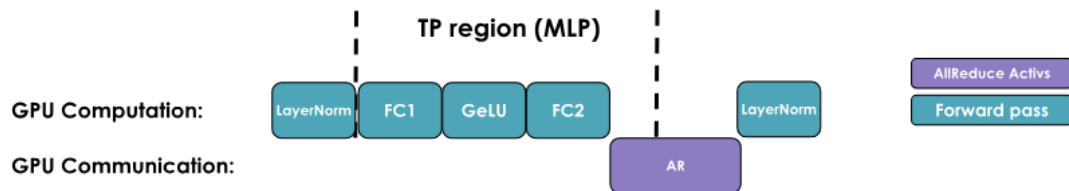


Figura 5.4: Transferência de dados durante o Paralelismo de Tensor.

Esse é um método eficiente nas camadas de *feedforward*, enquanto que nas camadas de Atenção basta dividir cada atenção em uma GPU e concatenar depois para propagar para a *feedforward* (Figura 5.5)

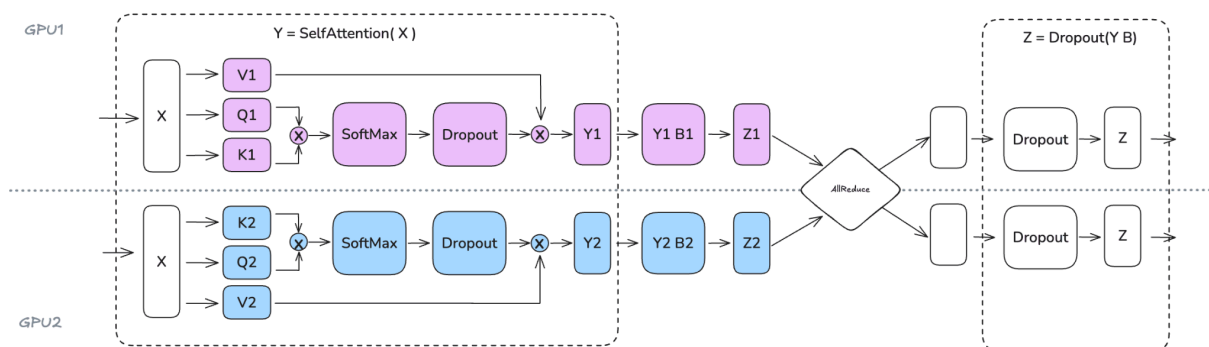


Figura 5.5: Paralelismo De Tensor aplicado às camadas de Atenção.

As atenções, naturalmente, podem ser divididas pelo número de head *attentions*, cada head operada simultaneamente. Isso pode mudar quando existe *Grouped Query Attention*.

Observe que precisamos fazer várias operações distribuídas primitivas e por isso nosso desempenho final será uma compensação do trade-off entre memória, comunicação e processamento.

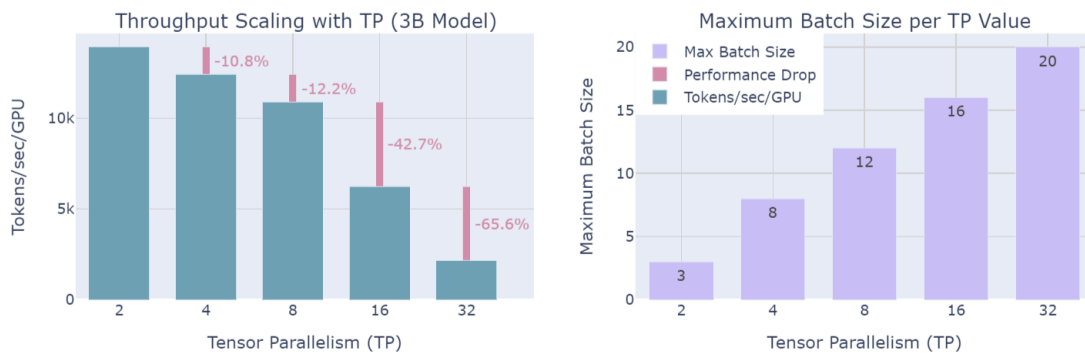


Figura 5.6: O Paralelismo de tensor diminui o *throughput*, mas permite um *batch size* maior.

Quando se utiliza mais um nó, de 8 para 16 GPUs, há uma queda significativa. Na prática começa a se tornar inviável TP entre nós (Figura 5.6), mas ao utilizar as conexões NVLink, o paralelismo de tensor oferece benefícios importantes para a diminuição de uso de memória.

## Paralelismo de Sequência

Em camadas com normalização, precisamos da soma dos quadrados, então cada GPU pode fazer isso parcialmente e juntar o resultado em cada uma das GPUs. Se os *hidden states* fossem compartilhados entre todas as GPUs, haveria uma redundância desnecessária.

Por isso, o paralelismo de sequência é feito nas camadas de normalização e no *dropout* com Paralelismo de Tensor, para evitar compartilhar toda a sequência do *hidden state*, o benefício pode ser visto ao comparar a Figura 5.6 com a Figura 5.8.

Então observe, nas regiões de normalização podemos dividir a sequência para cada GPU (Figura 5.7). E depois juntá-las novamente para passar nas reuniões de paralelismo de Tensores, como mostra a Figura 5.7. A vantagem é evitar manter memória de ativação

(Figura 5.9) nas GPUs e isso se dá pela fórmula:  $(batch\ size * sequence * hidden\ size) / (TP)$ .

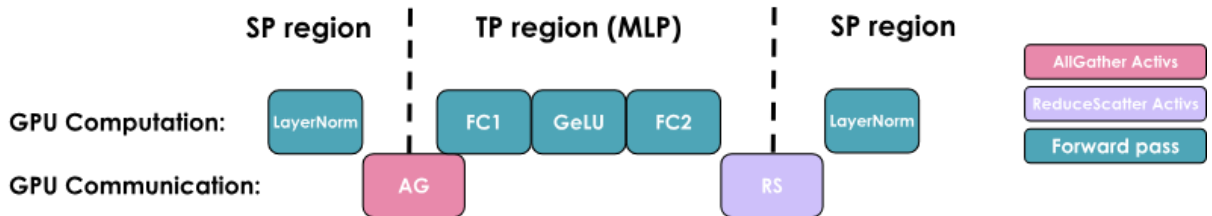


Figura 5.7: Transferência de dados no Paralelismo de Sequência com Paralelismo de Tensor.

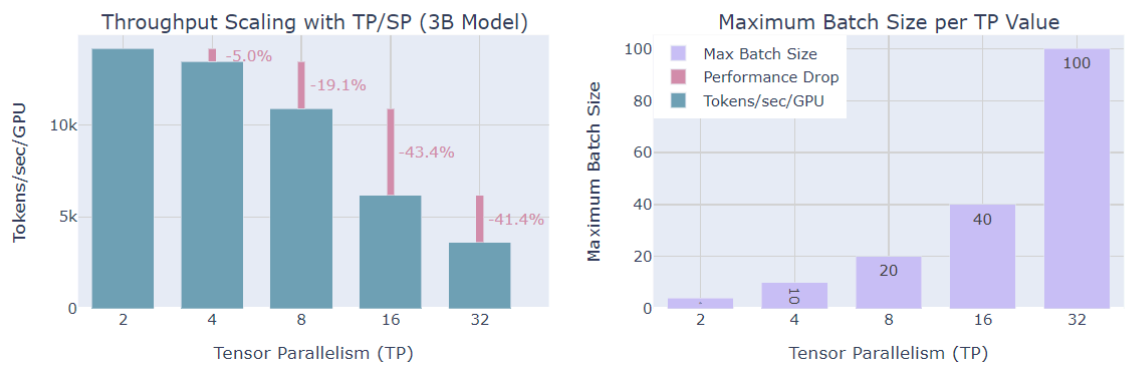


Figura 5.8: O Paralelismo Sequencial melhora a performance do paralelismo tensorial.

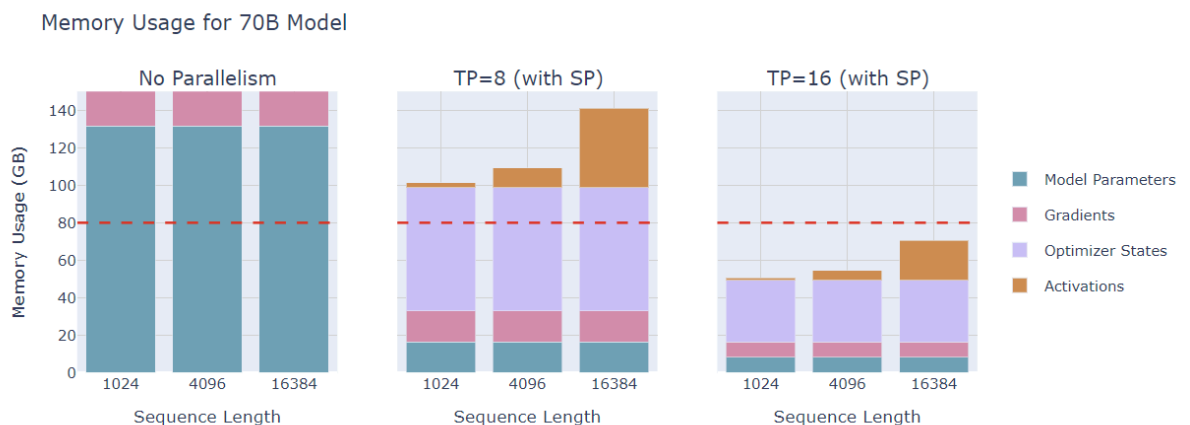


Figura 5.9: Uso de memória com um modelo de 70B de parâmetros no Paralelismo de Tensor e de Sequência.

Agora com 2 clusters com 8 GPUs já é possível treinar um modelo de 70B, mas ainda podemos otimizar...

## Paralelismo de Contexto

Como processar uma sequência de 128k tokens em um único nó?

A ideia é processar e coordenar o processamento de uma divisão da sequência. Não é necessário que a divisão tenha o comprimento da janela de contexto, mas precisamos dos tokens anteriores para calcular o *attention score*:

Então, para isso, precisamos guardar o KV cache e, para isso, existe o *Ring Attention*.

### *Ring Attention*

Nessa implementação, cada GPU calcula o KV e envia para as outras, enquanto aguarda o KV das sequências das outras GPUs, Mas ainda tem um grande problema.

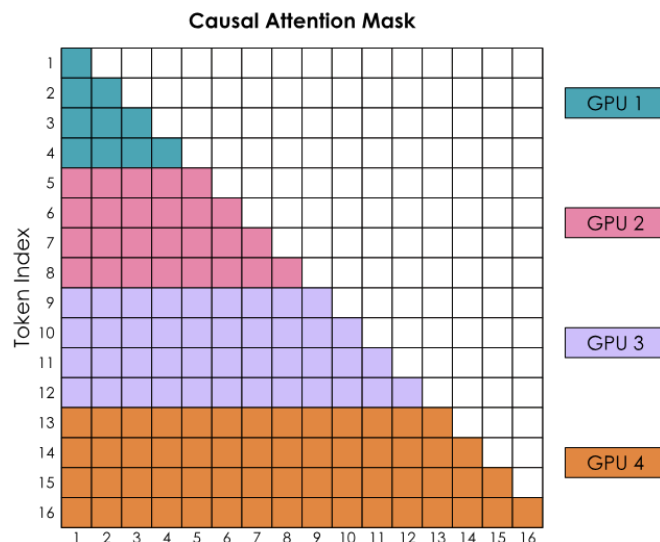


Figura 5.10: Mapeamento de cada token em função do Device ( GPU).

A sequência da GPU4, por exemplo, o K e V é muito maior do que a GPU1 (Figura 5.10) e ainda assim, se fosse implementado desse modo, a GPU4 precisaria aguardar todas as GPUs anteriores computarem, por isso, sugere-se utilizar o *Zig-Zag Ring Attention*.

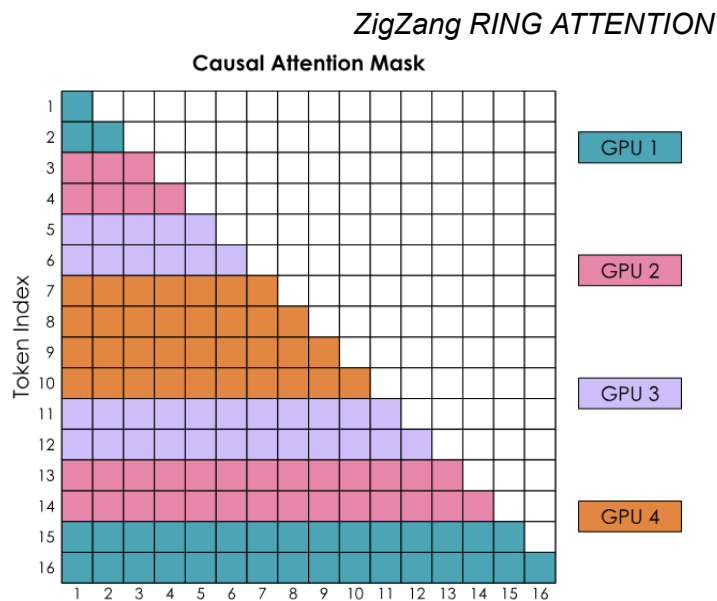


Figura 5.11: Mapeamento de cada token em função do Device (GPU) com *ZigZang Ring Attention*.

Há 2 formas de implementação:

<p>Todas calculam o K,V primeiro e aguardam receber o kV Cache das outras.</p> <p style="text-align: center;"><b>Attention</b></p> <p>GPU Computation: <span style="border: 1px solid black; padding: 2px;">Attn(Qi, Ki, Vi)</span>    <span style="border: 1px solid black; padding: 2px;">Attn(Qi, Ki+1, Vi+1)</span>    <span style="border: 1px solid black; padding: 2px;">Attn(Qi, Ki+2, Vi+2)</span>    <span style="border: 1px solid black; padding: 2px; color: red;">AllGather Activs</span></p> <p>GPU Communication: <span style="border: 1px solid black; padding: 2px; color: red;">AG(K,V)</span>    <span style="border: 1px solid black; padding: 2px;">Forward pass</span></p> <p>Desse modo, o All Gather ocorre todo de uma vez, mas ainda consome mais memória, pois todas as GPUs precisam armazenar todos os K,V.</p>	<p>Computar sob demanda</p> <p style="text-align: center;"><b>Attention</b></p> <p>GPU Computation: <span style="border: 1px solid black; padding: 2px;">Attn(Qi, Ki, Vi)</span>    <span style="border: 1px solid black; padding: 2px;">Attn(Qi, Ki+1, Vi+1)</span>    <span style="border: 1px solid black; padding: 2px;">Attn(Qi, Ki+2, Vi+2)</span>    <span style="border: 1px solid black; padding: 2px; color: green;">P2P Activs</span></p> <p>GPU Communication: <span style="border: 1px solid black; padding: 2px; color: green;">Fetch Ki+1, Vi+1</span>    <span style="border: 1px solid black; padding: 2px; color: green;">Fetch Ki+2, Vi+2</span>    <span style="border: 1px solid black; padding: 2px; color: green;">Fetch Ki+3, Vi+3</span>    <span style="border: 1px solid black; padding: 2px;">Forward pass</span></p> <p>As GPUs trocam pares de K,V então cada uma delas vai computando, desse modo é mais eficiente em termos de memória pois permite descartar os valores computados e também há uma sobreposição mais efetiva entre computação e comunicação.</p>
---	---

## Pipeline Parallelism

A degradação da comunicação com Paralelismo de Tensor entre nós, pode ser observada na Figura 5.12.

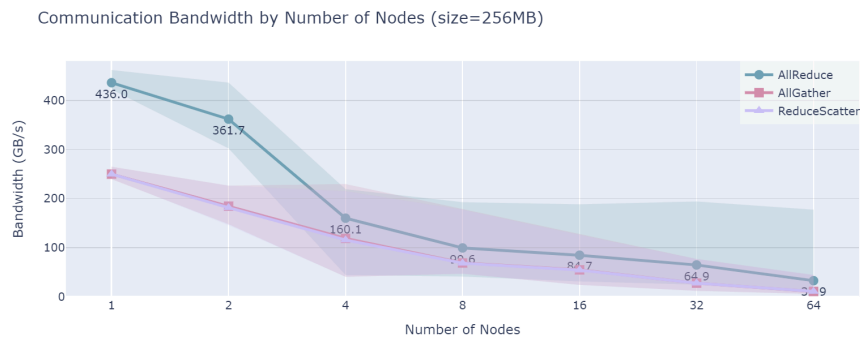


Figura 5.12: Custo de transferência de dados em função do número de Nós.

Para modelos grandes, como os de 70B de parâmetros, os pesos já ultrapassam os limites de 4 a 8 GPUs em um cluster. O paralelismo de Pipeline é simples: podemos dividir as camadas neurais do modelo entre as GPUs. A Figura 5.13 apresenta um comparativo.

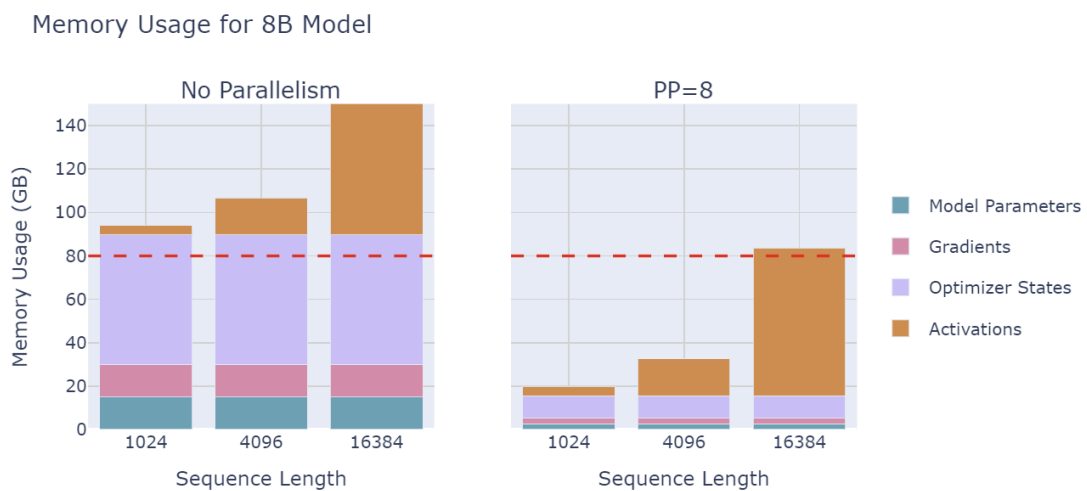


Figura 5.13 Comparação de uso de memória no Paralelismo de Pipeline

No primeiro momento, poderíamos pensar em dividir o modelo conforme a Figura 5.14, mas há tempo ocioso considerável em cinza que pode ser chamado de “bubble”. Além disso, as explosões de memória ocorrem pela necessidade de armazenar as ativações para fazer o *backward*.

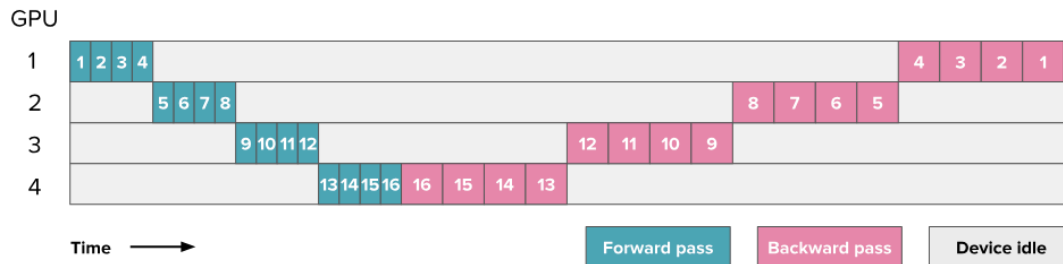


Figura 5.14 Paralelismo de Pipeline de modo ingênuo.

Um “*forward*”, um “*backward*” e Llama 3.1

Essa estratégia ficou conhecida como 1F1B (One *Forward*, One *Backward*), a ideia geral é tentar executar o *backward* o mais rápido possível (Figura 5.15). O “bubble” continua da mesma forma, mas agora há mais memória disponível e, portanto, poderia utilizar um batch maior.

Observe que apenas um laço “for” no treinamento não suporta essas modificações e acabam sendo bem extensas. A Figura 5.16 mostra como essas otimizações melhoram a performance:

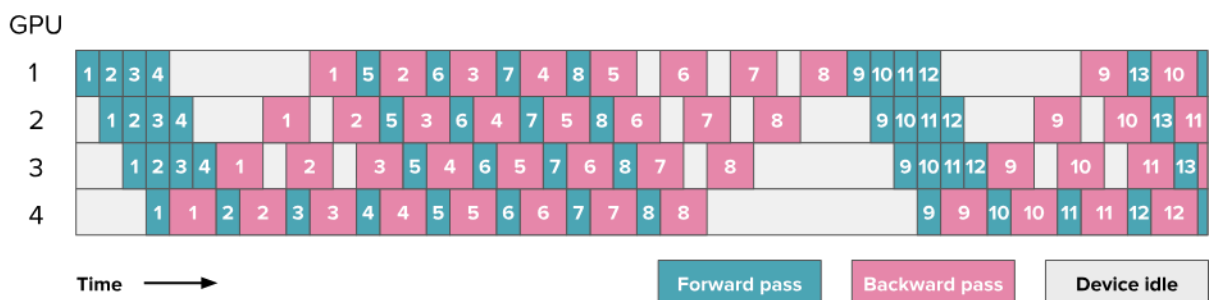


Figura 5.15: Paralelismo de Pipeline implementado no Llama 3.1

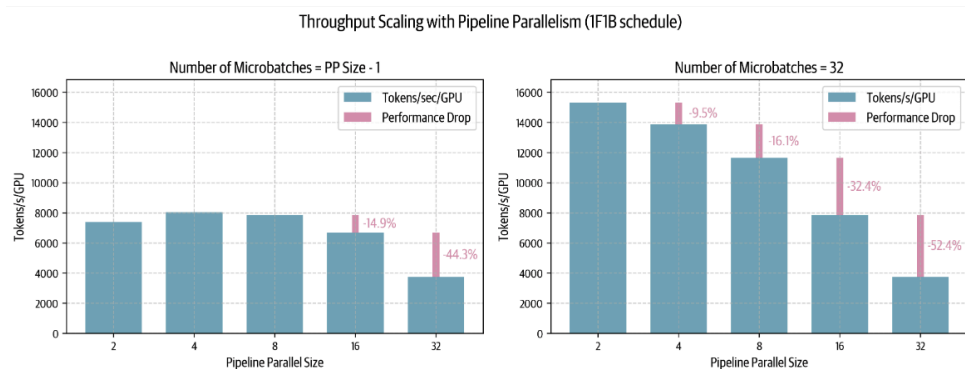


Figura 5.16: Comparativo entre o uso do Paralelismo de Pipeline ingênuo (imagem à esquerda) e o uso do 1F1B. (imagem à direita) .

### Intercalando os Estágios

Ao dividir as camadas, por exemplo, a GPU 1 utiliza as ímpares e a GPU2, as pares, mais comunicação é requerida, mas permite utilizar melhor o *backward*, embora isso possa se tornar ainda mais complexo:

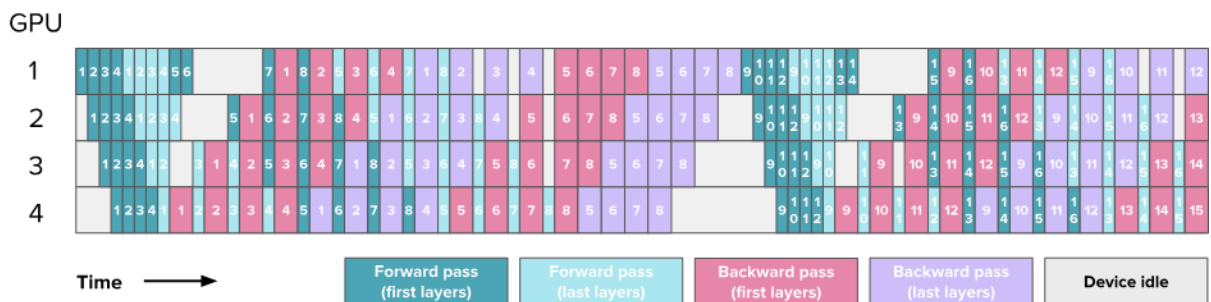


Figura 5.17: Paralelismo de Pipeline intercalando as camadas em cada GPU.

### Zero Bubble and DualPipe

No artigo do DeepSeek<sup>9</sup>, há uma menção em que reduz o bubble virtualmente para zero, conhecida como “DualPipe”.

<sup>9</sup> DeepSeek-AI et al. **DeepSeek-V3 Technical Report**. 2024.

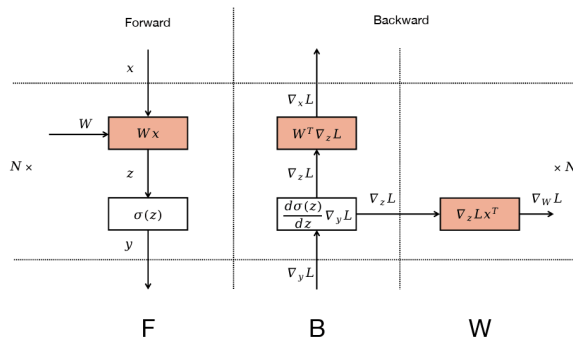


Figura 5.18 Grafo computacional para MLP

Um método ainda mais sofisticado proposto pelo DeepSeek (Figura 5.19). Para calcular o gradiente de cada peso, precisa do gradiente da camada posterior. E é feito dois cálculos, primeiro derivado em função das entradas e depois em função dos pesos. A ideia é ter um agendamento para calcular o gradiente do peso, para preencher as bolhas, como mostra na Figura 5.18



Figura 5.19 Paralelismo de Pipeline utilizado pelo DeepSeek.

## Paralelismo de Expert

Em 1991 já há menção a MoE<sup>10</sup> e em 2017 escalaram uma LSTM com mais de 1T de parâmetros<sup>11</sup>. A Figura 5.20 mostra alguns métodos de Paralelismo com experts, parece que essa otimização está ligada intimamente à arquitetura, como a quantidade de MoEs, etc.

<sup>10</sup> JACOBS, Robert A. et al. **Adaptive mixtures of local experts**. *Neural Computation*, v. 3, n. 1, p. 79-87, 1991

<sup>11</sup> Shazeer, N., Mirhoseini, A., Maziarz, K., Davis, A., Le, Q., Hinton, G., & Dean, J. (2017). *Outrageously large neural networks: The sparsely-gated mixture-of-experts layer*.

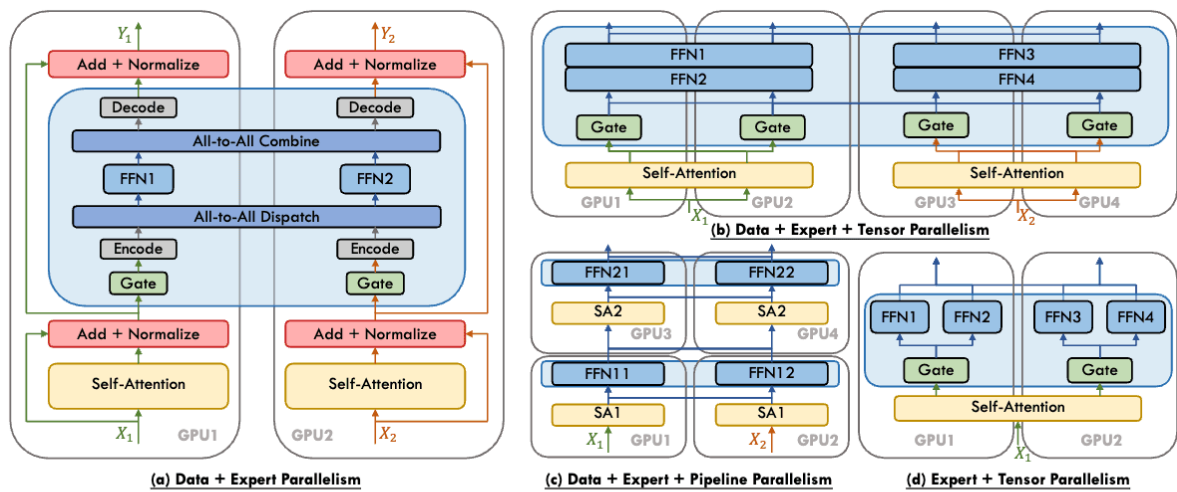


Fig. 8. Schematic depiction of diverse parallel strategies for MoE. For clarity and conciseness, this illustration omits some All-to-All, All-Reduce, Point-to-Point communication within parallelism, and Normalization, Encode, Decode, Gate in subfigures (b), (c), and (d).

Figura 5.20: Diferentes modos de implementar o Paralelismo de Experts.

Paralelismo Especializado:

Já existe, mas o MOE só está deixando isso um pouco mais forte.

## Paralelismo 5D - em poucas palavras

Paralelismo:

- Paralelismo de dados (DP) – ao longo da dimensão do *batch*
- Paralelismo de tensores (TP) – ao longo do *hidden state*
- Paralelismo de sequência e contexto (SP/CP) – ao longo da dimensão da sequência
- Paralelismo de pipeline (PP) – ao longo das layers
- Paralelismo de especialistas (EP) – ao longo dos experts

Estratégias ZeRO:

- ZeRO 1: Particionamento do Optimizer
- ZeRO 2: Particionamento do Optimizer e do Gradiente
- ZeRO 3: Particionamento do Optimizer, do Gradiente e dos pesos do modelo.

As Figuras 5.22 e 5.23 mostram uma visão geral sobre comunicação e uso de memória.

	<b>ZeRO-3</b>	<b>Pipeline Parallelism</b>
Each compute unit stores...	only a fraction of a layer	a full layer
Communication is used to transfer...	weights	activations
Orchestration	Model-agnostic	Model-agnostic
Implementation challenges	Complex to handle model partitioning and communications	Complex to handle efficient PP schedules
Scaling considerations	Prefers large <i>mbs</i> and <i>seq_len</i> to hide comms	Prefers large <i>grad_acc</i> to hide bubble

Figura 5.21: Comparação entre o método ZeRO e o Paralelismo de Pipeline.

A escolha entre ZeRO 3 e Paralelismo de Pipeline dependerá se você quer focar na comunicação para transferir pesos ou as ativações (Figura 5.21).

Embora possam ser combinados, isso não é feito com frequência na prática, pois isso requer um aumento significativo no tamanho global *batch size* para amortizar os custos de comunicação, criando um trade-off entre o tamanho global *batch size*, o tamanho do modelo, a largura de banda da rede e a eficiência do treinamento. Se você decidir combiná-los, o ZeRO-3 deve ser configurado para manter os pesos na memória durante a série de microlotes PP para minimizar ao máximo a sobrecarga de comunicação desnecessária.

Combinar ZeRO3 com Paralelismo de Pipeline requer aumentar o *batch size* para amortizar os custos de comunicação.

Por outro lado, ZeRO-1 e ZeRO-2, que se concentram em estados e gradientes do otimizador, podem ser facilmente combinados com o paralelismo de pipeline e são complementares a ele.

DeepSeek-v3 utilizou PP combinado com ZeRO-1.

O paralelismo de tensor deve ser mais cuidadoso, porque tem uma dependência com o modelo, além da necessidade dos grupos de GPUs que utilizam TP precisarem estar dentro do mesmo nó.

O paralelismo tensorial (e o paralelismo de sequência) afeta a computação em todo o modelo, fragmentando pesos e ativações.

O paralelismo de contexto impacta principalmente as camadas de atenção, pois é onde a comunicação entre sequências é necessária, com outras camadas operando independentemente em sequências fragmentadas.

O paralelismo de pipeline e o ZeRO não são especialmente específicos para nenhum submódulo ou componente, com a exceção de que módulos e camadas precisam ser balanceados no paralelismo de pipeline (a primeira e a última camadas são frequentemente tratadas de forma diferente devido às camadas de incorporação adicionais).

<b>Tensor + Sequence Parallel</b>	<b>Context Parallel</b>	<b>Expert Parallel</b>
Shards weights and activations along hidden/seq dim	Shards activations along sequence dim	Shards specialized expert weights and activations
Communication for matrix multiplication operations (column/row linear)	Communication for attention keys/values	Communication for token routing to experts
Model-specific implementation needed	Model-agnostic except for attention	Model-agnostic except for MoE layers
Prefers high-bandwidth intra-node communication	Prefers large sequence lengths	Requires MoE layers

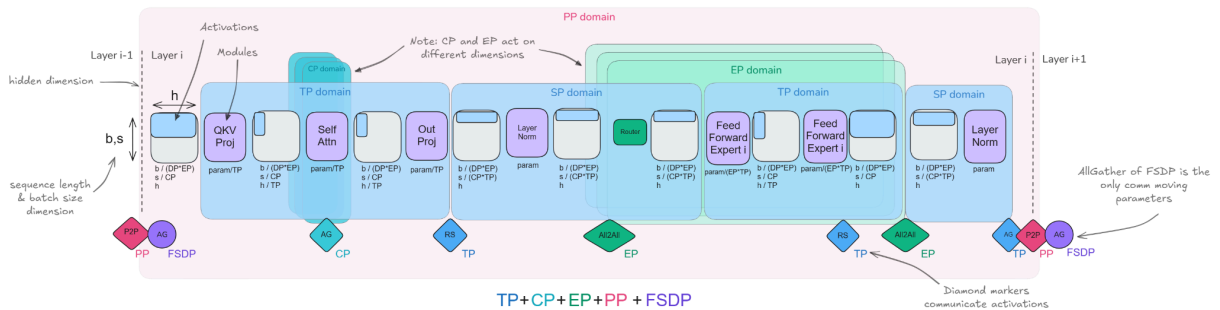


Figura 5.22: Combinação de todas as metodologias de Paralelismo.

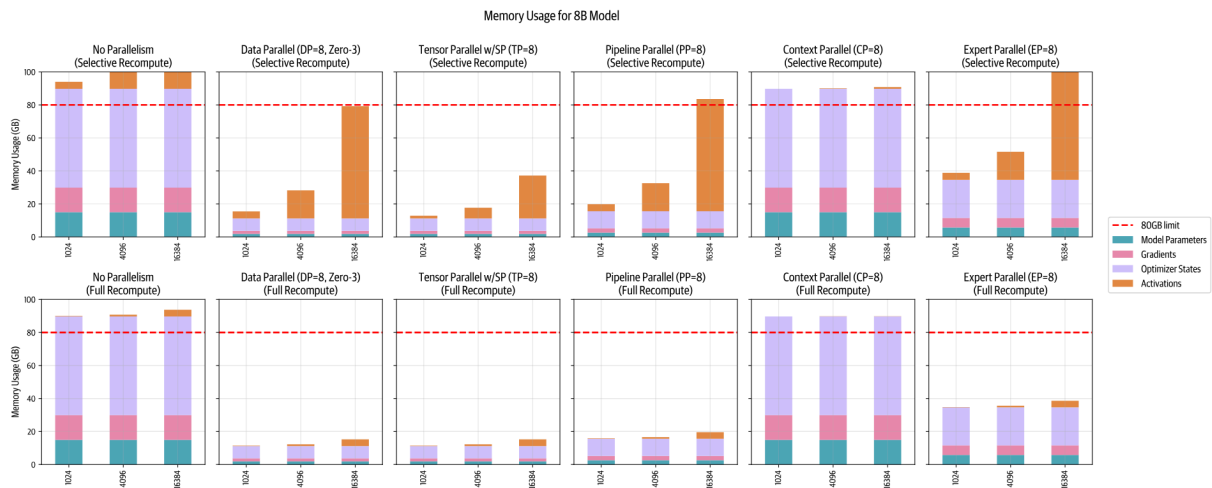


Figura 5.23: Visão geral de economia de memória.

Method	Memory savings apply specifically on...	Parallel/sharding dimension	Disadvantage
DP	Activations (reduce local batch size)	Batch	Limited by max batch size
PP	Model parameters	Model layers	Idle bubble and complex schedules
TP+SP	Model parameters and activations	Hidden dimension/sequence length	Requires high-bandwidth communication
CP	Activations	Sequence length	Adds communication overhead in attention modules
EP	Experts parameters	Experts dimension	Requires MoE layers, adds routing communication overhead
ZeRO-1	Optimizer states	Sharded among DP replicas	Params communication overhead
ZeRO-2	Optimizer states and gradients	Sharded among DP replicas	Params communication overhead
ZeRO-3	Optimizer states, gradients, and model parameters	Sharded among DP replicas	Params communication overhead

Figura 5.24: Comparativo dos métodos de Paralelismo e ZeROs.

## APÊNDICE 4

## Termo de Aceite de Entrega

### Objetivo deste documento

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

**Data da Reunião (“Gate”) de aprovação:** 8 de out. de 2025

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

Daniel Fazzioni

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

### Como treinar grandes modelos em clusters de GPUs de modo eficiente(?)

**Semana 1:** Encontrei o livro: [The Ultra-Scale Playbook: Training LLMs on GPU Clusters](#)

- Li sobre o treinamento em 1 GPU ( Memória: pesos, ativações, gradiente, otimizadores)

**Semana 2 e 3:** Li 1 Artigo sobre *batch size* e ruído do gradiente ( Mudei minha percepção sobre  $\nabla$ )

**Semana 4:** Li sobre o capítulo Data Parallelism e sobre os métodos ZeRO.

**Semana 5:** entendi sobre (Paralelismo de Tensor, Sequência, Contexto, Pipeline, Experts)

Na Semana 6, li alguns artigos:

- [GPT-OSS 120b & GPT-OSS-20B Model Card PT-OSS](#) (2025)
- [DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning](#) (2025)
- [DeepSeek LLM Scaling Open-Source Language Models With Longtermism](#) (2024)
- Ainda não finalizei o último capítulo, mas entendo a importância para otimização.

Comecei a replicar o primeiro artigo do DeepSeek:

- Criei um Tokenizador em português, a partir do dataset [FineWeb-2](#)
- A partir dos hiperparâmetros explorados no Artigo do DeepSeek iniciei um treinamento do zero com um modelo de 120M de parâmetros.
- Também computei o MFU: 3,6% (estou um pouco inseguro sobre esse resultado!)

[Anotações](#)

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

- Continuar a leitura do último capítulo do Livro
- Mudar o treinamento para o *framework* do [Megatron-LM](#) (Nvidia)
- Continuar a leitura sobre artigos explicando o processo de treinamento de LLMs.

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

---

## ACEITE DA ENTREGA:

CEDRIC LUIZ DE CARVALHO: Go! ▾

## Anotações da Semana 6

Compilar códigos em CUDA para utilizar com o PyTorch requer uma curva de aprendizado íngreme, uma forma de otimizar é utilizar o decorator `@torch.compile`; por exemplo, na função ELU, o decorator gera *kernels* em Triton. A Figura 6.1 mostra um comparativo entre usar `@torch.compile` e usar o modo ingênuo.

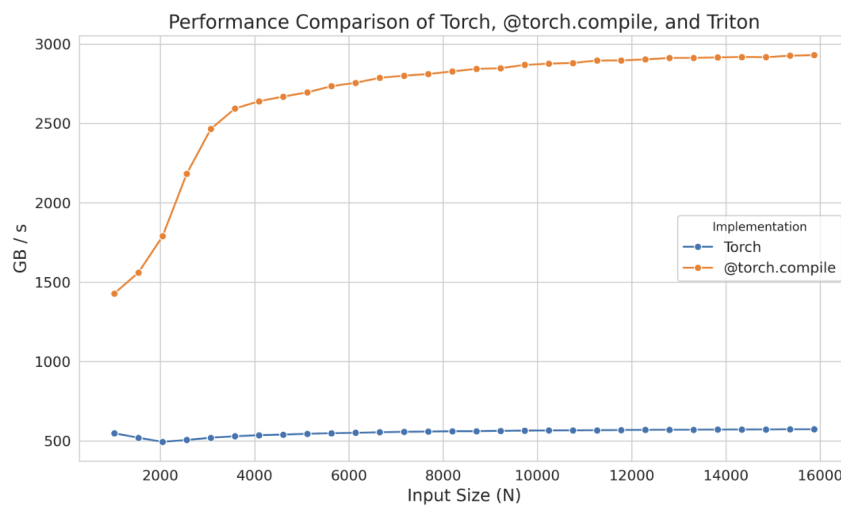


Figura 6.1: Comparativo com e sem torch.compile.

### GPT-OSS-120B & GPT-OSS-20B Model Card<sup>12</sup>

Um pouco do processo veio do GPT-4 (Dados) ou do GPT-O3 (pós-treino). Com treinamento em precisões diferentes, mxfp4 nas MoE e BF16 nas *attentions*. Também traz referências dos artigos anteriores da OpenAI, como uma hierarquização de instrução (system tem maior prioridade do user, output do LLM e das respostas das tools, por exemplo).

<sup>12</sup> Agarwal, S. et al. (2025). *gpt-oss-120b & gpt-oss-20b Model Card*. arXiv preprint arXiv:2508.10925. <https://arxiv.org/abs/2508.10925>

Os benchmarks chamam a atenção, por exemplo o benchmark multilingual (traduzido por especialistas): [MEASURING MASSIVE MULTITASK LANGUAGE UNDERSTANDING](#)<sup>13</sup>, e outros benchmarks internos, feitos pela própria OpenAI.

### [DeepSeek LLM Scaling Open-Source Language Models With Longtermism](#)

O artigo de janeiro de 2024 treina um LLM em 3 etapas, um modelo pré-treinado com next-token prediction e no pós-treino utiliza um estágio de SFT e outro de DPO. No modelo de 70B o desempenho relatado foi superior ao Lama 2 70B e ao GPT 3.5.

Sobre Scaling Laws, o artigo argumenta que o dataset, ou a complexidade do dataset também impacta o Scaling Laws, e um modo de avaliar a qualidade do dataset é pela dificuldade de overfit. O estudo também contribui com a premissa de que a performance é melhorada com o aumento de updates, parâmetros e dados.

No artigo, também argumentam que o *batch size* e a *learning rate* contribuem para a *loss* final, criando então uma busca. Além disso, também há uma discussão sobre esses argumentos funcionarem à medida que o tamanho do modelo aumenta.

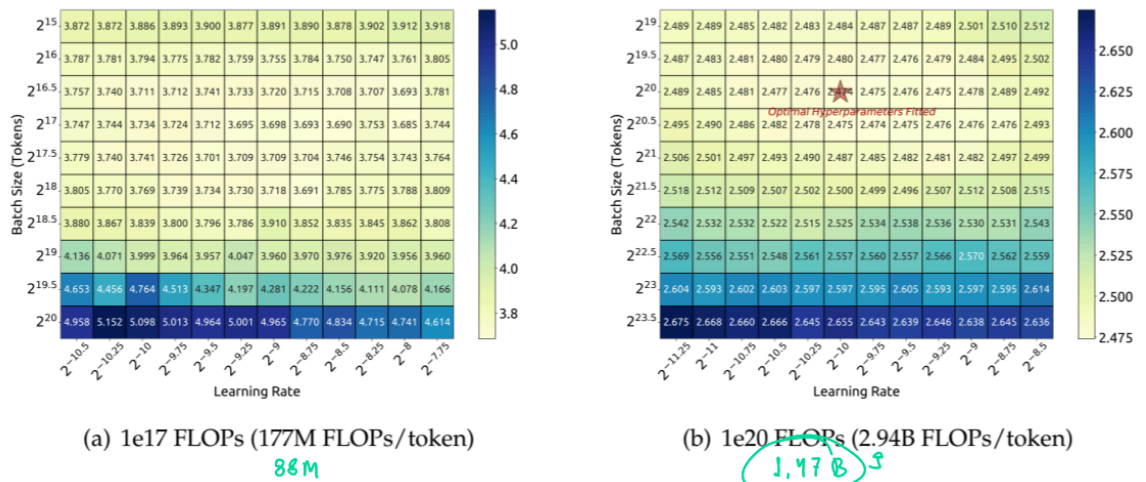


Figura 6.3: Resultado da busca da *loss*, *batch size* e *learning rate* em um modelo de cerca de 88M de parâmetros e outro de 1.47B de parâmetros.

<sup>13</sup> Hendrycks, D., Burns, C., Basart, S., Zou, A., Mazeika, M., Song, D., & Steinhardt, J. (2021). *Measuring massive multitask language understanding*. Proceedings of the International Conference on Learning Representations (ICLR). <https://arxiv.org/abs/2009.03300>

Na etapa de SFT utilizaram em torno de 1M de amostras de dados de instrução, enquanto o tokenizador foi gerado a partir de 24GB de dados multilingual e a quantidade de tokens utilizada no pré-treino foi 2T de tokens.

HiperParâmetros:

- Inicialização com desvio padrão de 0.006 e utilizando AdamW Optimizer ( $\beta_1 = 0,9$   $\beta_2 = 0,95$ ) e weight decay 0.1
- Geralmente utiliza-se uma *learning rate* em uma função de cosseno, mas optaram por utilizar em estágios com o intuito de continuar o treinamento.

## Tokenizador

Pela dificuldade de encontrar algum tokenizador em português, criei um com ajuda do Gemini e do GPT, em português de 32k tokens, utilizando o byte pair encoding (BPE) no dataset do FineWeb utilizando o split em português:

```
import datasets
import os
from tokenizers import Tokenizer, normalizers, pre_tokenizers, decoders
from tokenizers.models import BPE
from tokenizers.trainers import BpeTrainer
from transformers import PreTrainedTokenizerFast
from tqdm import tqdm

def run(TARGET_VOCAB_SIZE=32_000, sample_size=50_000_000):
    def get_training_corpus(batch_size=1_000):
        dataset_stream = datasets.load_dataset("HuggingFaceFW/fineweb-2", "por_Latn",
split="train", streaming=True).take(sample_size)
        print("Dataset pronto para iteração em modo streaming...")
        batch = []
        for example in dataset_stream:
            # Ignora textos vazios ou inválidos
            if example and example["text"]:
                batch.append(example["text"])
            if len(batch) == batch_size:
                yield batch
                batch = []
```

```
# Não esqueça o último lote que pode não ter atingido o batch_size
if batch:
    yield batch

# 1. Instanciar o Tokenizer
tokenizer = Tokenizer(BPE(unk_token="<unk>"))
# 2. Adicionar um Normalizador
tokenizer.normalizer = normalizers.NFC()
# 3. Pré-tokenizador a nível de byte
tokenizer.pre_tokenizer = pre_tokenizers.ByteLevel()
# 4. Decodificador a nível de byte (Agora compatível com o pré-tokenizador)
tokenizer.decoder = decoders.ByteLevel()

print("Tokenizer configurado para Byte-Level BPE.")
# Trainer BPE (sem alterações aqui)
trainer = BpeTrainer(
    vocab_size=TARGET_VOCAB_SIZE,
    min_frequency=5,
    special_tokens=["<pad>", "<bos>", "<eos>", "<unk>"],
    show_progress=True,
)
progress_iterator = get_training_corpus()
tokenizer.train_from_iterator(progress_iterator, trainer=trainer)
save_path = f"Tokenizer-ByteLevel-BPE_{sample_size}"
os.makedirs(save_path, exist_ok=True)
tokenizer.save(os.path.join(save_path, "tokenizer.json"))

print(f"\n✅ Tokenizador treinado e salvo em '{save_path}'")

hf_tokenizer = PreTrainedTokenizerFast(tokenizer_file=os.path.join(save_path,
"tokenizer.json"))
hf_tokenizer.add_special_tokens(
    {"pad_token": "<pad>", "bos_token": "<bos>", "eos_token": "<eos>", "unk_token":
"<unk>"}
)
print("\nExemplo de tokenização:")
test_text = "Olá, estou testando o meu tokenizador unificado estilo GPT, com coração
e sensação!"
encoded = hf_tokenizer.encode(test_text)

print(f"Texto Original: {test_text}")
print(f"Tokens (IDs): {encoded}")
decoded_text = hf_tokenizer.decode(encoded)
```

```
print(f"Texto Decodificado: {decoded_text}")
assert test_text == decoded_text

if __name__ == "__main__":
    import argparse
    parser = argparse.ArgumentParser(description="Treina um tokenizador Byte-Level BPE.")
    parser.add_argument("--vocab_size", type=int, default=32_000, help="Tamanho do
vocabulário.")
    args = parser.parse_args()
    run(TARGET_VOCAB_SIZE=args.vocab_size)
```

## Iniciando um Pré-Treino

Com o artigo do DeepSeek utilizei a biblioteca do Transformers<sup>14</sup> e iniciei aleatoriamente um LLM de 120M de parâmetros, seguindo os parâmetros do DeepSeek em uma infraestrutura de 1 nó com apenas uma H100, conforme a Tabela 6.1.

Tabela 6.1: Parâmetros do primeiro treinamento com Transformers.

Categoria	Parâmetros	Configuração
Model Spec	Architecture	Gemma 3
	Parameters	120M
	Context Window	4096
Tokenizer	Type	BPE
Training Flow	Tokens per Step	~2.1M
	Global Batch	512
Optimization	Strategy	AdamW Fused (Constant LR)
	<i>Learning Rate</i>	3e-6
	Hardware Opt	1 GPU (BF16 + Liger Kernel)

<sup>14</sup> WOLF, Thomas et al. (2020). *Transformers: State-of-the-Art Natural Language Processing*. Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations (pp. 38–45). <https://www.aclweb.org/anthology/2020.emnlp-demos.6>

Cálculo de MFU:

Em 2.400 segundos teve 57 updates com 2.097.152 tokens por *batch size*. Em 1 segundo, é propagado 49.807,36 tokens.

Para calcular o MFU no Transformers vamos utilizar a aproximação:

$$\text{MFU} = 6 * \text{Tokens/s} * \text{Núm. Parâmetros} / \text{Flops Máx.}$$

Flops Máx. pode ser encontrado nas especificações da GPU, como o experimento foi utilizado uma H100:

$$\text{MFU} = 6 * 49.807,36 * 1,2 * 10^8 \text{ Flops} / 989,4 * 10^{12} \text{ Flops}$$

$$\text{MFU} = 358 * 10^{11} \text{ Flops} / 989,4 * 10^{12} \text{ Flops}$$

$$\text{MFU} = 3,6\%$$

Considerando 100B de tokens, demoraria 3 semanas para computar em uma GPU H100.

## APÊNDICE 5

## Termo de Aceite de Entrega

### Objetivo deste documento

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

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

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

Daniel Fazzioni

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

### Treinamento Eficiente de Grandes Modelos em Clusters de GPUs

**Semana 1:** Encontrei o livro: [The Ultra-Scale Playbook: Training LLMs on GPU Clusters](#)

- Li sobre o treinamento em 1 GPU ( Memória: pesos, ativações, gradiente, otimizadores)

**Semana 2 e 3:** Mudei minha percepção sobre  $\nabla$  (An Empirical Model of Large-Batch Training, 2018)

**Semana 4:** Li sobre o capítulo Data Parallelism e sobre os métodos ZeRO.

**Semana 5:** Entendi sobre Paralelismo de Tensor, Sequência, Contexto, Pipeline e de Experts

**Semana 6:** Leitura de artigos (GPT-OSS, DeepSeek), pré-treinamento (100M de parâmetros, 20B de tokens treinados). Calculei o MFU em 3,6%.

**Na Semana 7:**

- Explorei o *framework* do [Megatron-LM](#)
  - Há mais de 500 argumentos
  - Consegui replicar experimentos que chegam em 44,5% de MFU
  - Fiz uma comparação entre a performance das GPUs H100 e B200.
    - Também comparei alguns *kernels* de *attention*
- Confirmei que o experimento anterior realmente estava utilizando 3,6% de MFU.
- Avaliei alguns checkpoints do modelo anterior e percebi que não aprendeu a linguagem.

[Anotações!](#)

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

Buscar artigos que tenham informações suficientes para replicar e comparar o pré-treino, parece que há uma linha com os artigos da Nvidia.

- Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism (Nvidia, 2020)
- Efficient Large-Scale Language Model Training on GPU Clusters Using Megatron-LM (Nvidia, 2021)
- Using DeepSpeed and Megatron to Train Megatron-Turing NLG 530B, A Large-Scale Generative

---

Language Model (Nvidia, 2022)

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

Agradeço ao Gustavo por ter me ensinado como funciona arquiteturas SSM (State-Space Models)

---

## ACEITE DA ENTREGA:

CEDRIC LUIZ DE CARVALHO: [Go! ▾](#)

## Anotações da Semana 7

### Megatron Core User Guide<sup>15</sup>

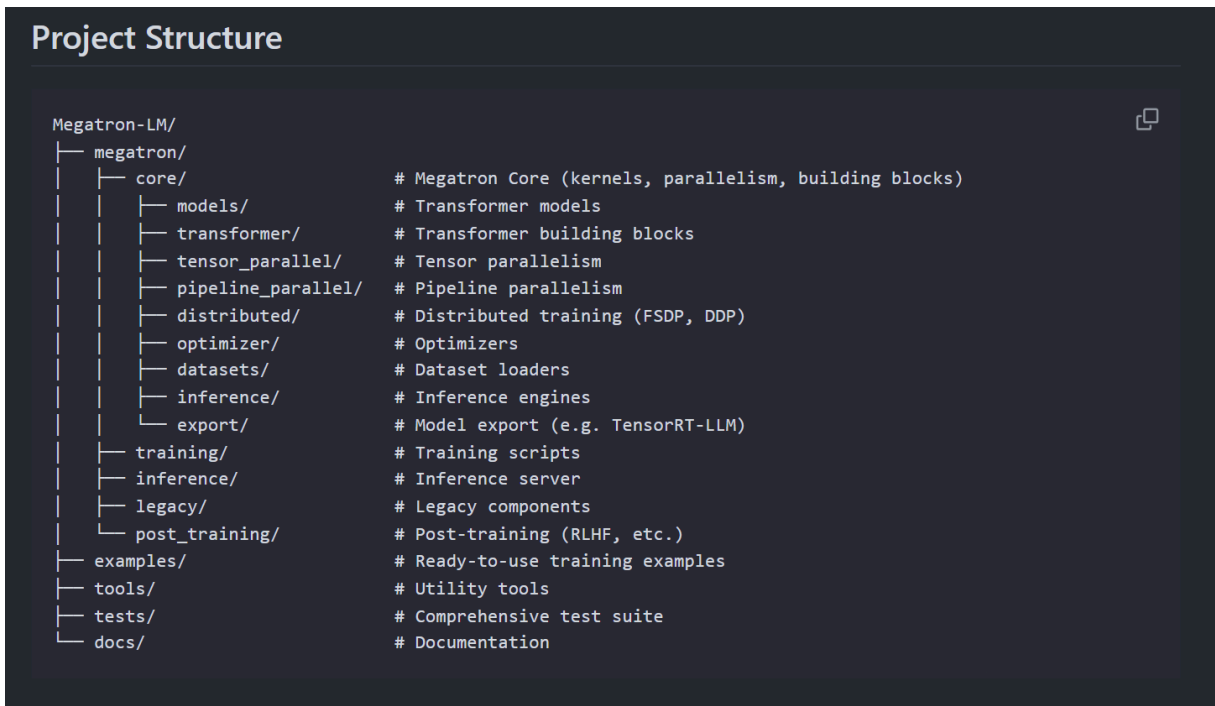


Figura 7.1: Organização dos arquivos no Megatron-LM.

Há uma estrutura bem definida do *framework* (Figura 7.1). Contudo, a sua complexidade é notória: ao utilizar como base o PyTorch distribuído, não é trivial entender as relações entre processos ou o uso de *frameworks* adicionais da própria Nvidia, por exemplo, o Nvidia Resiliency Extension<sup>16</sup> para reiniciar o treinamento sem sair do *Slurm*. Também há um controle de falhas, e isso também é orquestrado junto com outras funções para deixar o treinamento distribuído.

<sup>15</sup> Nvidia. Megatron-LM. 2025. Disponível em: <https://github.com/Nvidia/Megatron-LM>. Acesso em: 30 nov. 2025.

<sup>16</sup> Nvidia. Nvidia Resiliency Extension. 2025. Disponível em: <https://github.com/Nvidia/Nvidia-resiliency-ext>. Acesso em: 30 nov. 2025.

Além disso, o repositório em si não é naturalmente autoexplicativo, então há a necessidade de explorar para compreender.

Como por exemplo a construção de *datasets* no *framework* segue uma hierarquia, inclusive com suporte a uso de buckets para armazenar o dataset.

Após rodar o código inicial com DDP=2, queremos treinar em *datasets* específicos, portanto, é necessário entender como fazer esse processo. Também há um outro *framework* Megatron Energon<sup>17</sup> para *datasets* multimodais.

Parece que no Dataset de modelos como GPT, há uma classe definida que segue uma hierarquia ( Figura 7.2)



Figura 7.2 - Abstração gerada pelo GPT sobre a ingestão de dados no Megatron.

## Gerando Dataset

Não há documentação bem definida sobre o Megatron, mas desenvolvi um código que permite criar / armazenar o dataset para o treinamento. O código abaixo não utiliza buckets, mas há suporte para esses modos que podem facilitar o treinamento.

<sup>17</sup> Nvidia. (2025). Megatron-Energon. GitHub. <https://github.com/Nvidia/Megatron-Energon>

```
from transformers import AutoTokenizer
from megatron.core.datasets.indexed_dataset import IndexedDatasetBuilder
import numpy as np
import torch
import os
import datasets
import dotenv
dotenv.load_dotenv()
tokenizer = AutoTokenizer.from_pretrained("Fazzioni/tokenizer-fineweb.50M-32k")
dataset = datasets.load_dataset("HuggingFaceFW/fineweb-2", "por_Latn", split="train",
streaming=True)
os.makedirs("data", exist_ok=True)
prefix = "data/fineweb"
builder = IndexedDatasetBuilder(f"{prefix}.bin", dtype=np.int32)
for i, sample in enumerate(dataset):
    tokens = tokenizer.encode(sample["text"] + " " + tokenizer.eos_token)
    builder.add_item(torch.tensor(tokens, dtype=torch.int32))
    builder.end_document()
    if i == 100:
        break
builder.finalize(f"{prefix}.idx")
print("IndexedDataset salvo em:", prefix)
```

Verificando o dataset:

```
from megatron.core.datasets.indexed_dataset import IndexedDataset
from transformers import AutoTokenizer
import datasets
import dotenv; dotenv.load_dotenv()

tokenizer = AutoTokenizer.from_pretrained("Fazzioni/tokenizer-fineweb.50M-32k")
prefix = "data/fineweb"
dataset = IndexedDataset(prefix)
print("Número de documentos:", len(dataset))
print("Primeiro documento (tokens):", dataset[0])
print("Texto decodificado:", tokenizer.decode(dataset[0].tolist()))
```

## Experimentos Realizados

Utilizei o *framework* do Megatron para avaliar a performance em algumas configurações conforme a Tabela 7.1 e 7.2.

Tabela 7.1 Experimentos utilizando o Megatron-LM.

Framework	Parâmetros*	GPUs	TFLOPS	Info	Tokens/s	MFU
Megatron-LM	89M	<a href="#">1 H100</a>	215,5	BF16 - Flash attention 3	273 384	21,73%
Megatron-LM	89M	<a href="#">2 H100</a>	212,9	BF16 - Flash attention 3	497 102	22.14%
Megatron-LM	89M	<a href="#">2 H100</a>	222,7	FP8 - Flash attention 3	502 810	
Megatron-LM	89M	<a href="#">2 H100</a>	183,2	BF16 - Fused attention	427 850	18,52%
Megatron-LM	1.7B	<a href="#">2 H100</a>	431,5	BF16 - Flash attention 3	89 074	43,62%
Megatron-LM	1.7B	<a href="#">2 H100</a>	444,2	BF16 - Fused attention	93 071	44,89%
Transformers	120M	1 H100	35,8	BF16 - SDPA	52 022	3,6%
Megatron-LM	89M	<a href="#">2x B200</a>	282,9	BF16	720 317	12,7%

Tabela 7.2 Experimentos utilizando o Megatron-LM em um cluster com 8xGPUs B200.

Parâmetros*	Global BS	BSQ	Tokens/S	TFLOPS /GPU	MFU %	Dias necessários para 100B Tokens
2.5B	128	52 4288	325 948,30	722	32,09	3,55
5.4B	16	65 536	166 293,24	745	33,11	6,96
5.4B	32	13 1072	180 302,27	800	35,56	6,42
18.9	16	65 536	51 543,35	773	34,36	22,46
18.9	16	65 536	41 784,79	630	28,00	27,70
25.6	10	40 960	26 075,13	538	23,91	44,39
25.6	16	65 536	29 292,22	539	23,96	39,51
58B	10	40 960	6 461,26	543	24,13	179,13

Parâmetros\*: Parâmetros do modelo sem contar os *embeddings*. Para fins de comparação, 58B é equiparado a um modelo total de 70B de parâmetros, enquanto que o 5.4B é equiparado a um de 7B de parâmetros.

## APÊNDICE 6

## Termo de Aceite de Entrega

### Objetivo deste documento

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

**Data da Reunião (“Gate”) de aprovação:** 22 de out. de 2025

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

Daniel Fazzioni

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

### Treinamento Eficiente de Grandes Modelos em Clusters de GPUs

**Semana 1:** Encontrei o livro: [The Ultra-Scale Playbook: Training LLMs on GPU Clusters](#)

- Li sobre o treinamento em 1 GPU ( Memória: pesos, ativações, gradiente, otimizadores)

**Semana 2 e 3:** Mudei minha percepção sobre o gradiente  $\nabla$  (An Empirical Model of Large-Batch Training, 2018)

**Semana 4:** Li sobre o capítulo Data Parallelism e sobre os métodos ZeRO.

**Semana 5:** Entendi sobre Paralelismo de Tensor, Sequência, Contexto, Pipeline e de Experts

**Semana 6:** Leitura de artigos (GPT-OSS, DeepSeek), pré-treinamento (100M de parâmetros, 20B de tokens treinados). Calculei o MFU em 3,6%.

**Semana 7:** Explorei o [Megatron-LM](#) (alcancei 44,5% de MFU)

**Na Semana 8:**

- Iniciei buscando experimentos abertos de pré-treino, explorando os artigos da Nvidia: O trabalho sobre [Nemotron-CC: Transforming Common Crawl into a Refined Long-Horizon Pretraining Dataset](#) (Maio, 2025), me fez pensar sobre dados sintéticos e dados abertos.
  - Embora os experimentos fossem sobre modelos grandes, há um conjunto de informações de arquitetura sobre os testes.
- O trabalho [Nemotron-4 340B Technical Report](#) (Agosto, 2024) libera alguns hiperparâmetros de treinamento.

Ainda havia 2 desafios:

- O tokenizador dos modelos da Nvidia é muito grande para modelos pequenos
- Preciso escolher um conjunto de dados para treinar.

Encontrei o trabalho: [SmolLM2: When Smol Goes Big — Data-Centric Training of a Small Language Model](#) (Fevereiro, 2025)

- Neste trabalho há a reprodução de um LLM pequeno, mas há muitas informações abertas difíceis de encontrar nos artigos.

Com base nesses trabalhos, defini novos parâmetros e agora no *framework* do Megatron, iniciei um novo treinamento com o tokenizador e com uma parcela pequena dos dados do SmoLLM2.

O código é um pouco complexo, então precisei debugar algumas coisas... principalmente pela forma que os checkpoints são salvos ( de modo distribuído), mas consegui fazer a inferência e percebi, há pelo menos, duas observações:

- A curva de *loss* ficou muito mais interessante do que no treinamento anterior com a biblioteca transformers.
- A cada checkpoint do treinamento o modelo realmente estava aprendendo a linguagem inglesa e as respostas estavam cada vez mais interessantes.

O modelo tem 135M de parâmetros (justamente para comparar com o SmoLLM2), consegui treinar 50.3B de tokens em 2,73 dias em uma GPU H100, com 21,7% de MFU e não 3,6%!

[Anotações!](#)

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

- Criar um repositório com os códigos dos experimentos.
- Ler o artigo [SmoLLM2](#) com calma (eles já resolveram problemas que possivelmente irei enfrentar)
- Estudar sobre checkpoints distribuídos, entender como transformar o checkpoint para um modo mais “amigável”.
- Avaliar os checkpoints treinados com a forma que o [SmoLLM2](#) utilizou

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

---

## ACEITE DA ENTREGA:

CEDRIC LUIZ DE CARVALHO: [Go!](#)

## Anotações da Semana 8

O objetivo é encontrar modos / formas de replicar um pré-treino, sem inferir hiperparâmetros, então selecionei artigos da Nvidia que possam ser úteis:

<https://research.Nvidia.com/research-area/machine-learning-artificial-intelligence>

### [Nemotron-CC: Transforming Common Crawl into a Refined Long-Horizon Pretraining Dataset](#)<sup>18</sup> (May, 2025)

Artigo bem interessante, os resultados mostram que dados de alta qualidade com 1T de tokens podem ter desempenho superior a 2.6T de tokens. Uma redução de pelo menos 60% no custo. Nesse artigo também encontramos algumas referências:

*Layers Transformers:* 32

*Hidden dimension:* 4096

*Attention heads:* 32

*Activation:* SwiGLU

*Grouped query attention:* 8

*Adam optimizer*,  $\beta_1 = 0.9$ ,  $\beta_2 = 0.95$ ,  $e=1e-8$ , weight decay 0.1

*Cosine learning rate schedule* (peak lr  $3e-4$ , minimum  $3e-6$ )

O mesmo *framework* que escolhemos anteriormente: Megatron-LM

O artigo descreve e libera os dados, embora não esteja disponível para download:

<https://data.commoncrawl.org/contrib/Nemotron/Nemotron-CC/index.html>

No Hugging Face há um repositório com informações sobre esse dataset:

<https://huggingface.co/datasets/lehduong/nemotron-cc-hq>

---

<sup>18</sup> Su, D., Kong, K., Lin, Y., Jennings, J., Norick, B., Kliegl, M., ... & Catanzaro, B. (2024). *Nemotron-CC: Transforming Common Crawl into a Refined Long-Horizon Pretraining Dataset*. arXiv preprint arXiv:2412.02595. <https://arxiv.org/abs/2412.02595>

Mas parece que apenas uma parte está disponível:

<https://huggingface.co/collections/Nvidia/nemotron-pre-training-dataset-689d9de36f84279d83786b35>

## [Nemotron-4 340B Technical Report](#)<sup>19</sup> (Aug, 2024)

Não tem muita informação do modelo, mas existe bastante sobre os dados, principalmente do pós-treino. Além disso, há algumas otimizações diferentes das reportadas no artigo do Nemotron-CC.

No Hugging Face, há o modelo <https://huggingface.co/Nvidia/Nemotron-4-340B-Base>, mas parece que para nossa proposta o vocabulário do modelo é um pouco grande: 256.000 tokens.

Há algumas informações sobre o treino e também sobre a inicialização:

Data-parallel size	GPUs	Iteration time (secs)	MFU (%)	Batch size	Tokens (B)
16	1536	10.3	42.4%	768	200
32	3072	10.3	42.3%	1536	200
64	6144	8.0	41.0%	2304	7600

Table 2: Batch size rampup schedule, along with time and efficiency metrics for the Nemotron-4-340B-Base parameter model.

```
micro_gpt: true
micro_batch_size: 1
global_batch_size: 2308
tensor_model_parallel_size: 8
pipeline_model_parallel_size: 2
virtual_pipeline_model_parallel_size: null
encoder_seq_length: 4096
max_position_embeddings: 4096
num_layers: 96
hidden_size: 18432
ffn_hidden_size: 73728
num_attention_heads: 96
init_method_std: 0.0063
use_scaled_init_method: true
hidden_dropout: 0.0
attention_dropout: 0.0
ffn_dropout: 0.0
kv_channels: null
apply_query_key_layer_scaling: true
normalization: layernormlp
fp32_residual_connection: false
fp16_lm_cross_entropy: false
megatron_amp_O2: true
grad_allreduce_chunk_size_mb: 125
grad_div_ar_fusion: true
gradient_accumulation_fusion: false
bias_activation_fusion: false
bias_dropout_add_fusion: false
masked_softmax_fusion: true
seed: 1234
resume_from_checkpoint: null
use_cpu_initialization: false
onnx_safe: false
apex_transformer_log_level: 30
gradient_as_bucket_view: true
sync_batch_comm: false
activations_checkpoint_granularity: null
activations_checkpoint_method: null
activations_checkpoint_num_layers: null
num_micro_batches_with_partial_activation_checkpoints: null
```

<sup>19</sup> Adler, B. et al. (2024). *Nemotron-4 340B Technical Report*. arXiv preprint arXiv:2406.11704. <https://arxiv.org/abs/2406.11704>

```
layernorm_epsilon: 1.0e-05
do_layer_norm_weight_decay: false
make_vocab_size_divisible_by: 128
pre_process: true
post_process: true
persist_layer_norm: true
bias: false
activation: squared-relu
headscale: false
transformer_block_type: pre_ln
OpenAI_gelu: false
normalize_attention_scores: true
position_embedding_type: rope
rotary_percentage: 0.5
attention_type: multihead
share_embeddings_and_output_weights: false
num_query_groups: 8
tokenizer:
  library: sentencepiece
  type: null
  model:
nemo:d6b0ba93e9734b138f5fc61f5652efbd_nemotron_2_256k.mo
del
  delimiter: null
  vocab_file: null
  merge_file: null
  sentencepiece_legacy: false
  tokenizer_model:
nemo:29e0db5f7ddl4bcf9f32727ff482502b_nemotron_2_256k.mo
del
  native_amp_init_scale: 4294967296
  native_amp_growth_interval: 1000
  hysteresis: 2
activations_checkpoint_layers_per_pipeline: null
sequence_parallel: true
transformer_engine: false
fp8: false
fp8_e4m3: false
fp8_hybrid: false
fp8_margin: 0
fp8_interval: 1
fp8_amax_history_len: 1
fp8_amax_compute_algo: most_recent
reduce_amax: true
use_emha: false
optim:
  name: distributed_fused_adam
  lr: '1.0e-4'
  weight_decay: 0.1
  betas:
  - 0.9
  - 0.95
  sched:
    name: CosineAnnealing
    warmup_steps: 500
    constant_steps: 50000
    min_lr: '1.0e-7'
  apply_ropo_fusion: false
precision: bf16
target:
nemo.collections.nlp.models.language_modeling.megatron_gpt_model.Me
gatronGPTModel
nemo_version: 1.15.0rc0
```

Graças ao Research Rabbit, encontrei o **Nemotron-4 15B Technical Report**.

### **Nemotron-3** (Nov, 2023)

<https://huggingface.co/collections/Nvidia/nemotron-3-8b-6553adeb226f6ab4ffc356f9>

<https://developer.nvidia.com/blog/nvidia-ai-foundation-models-build-custom-enterprise-chatbots-and-co-pilots-with-production-ready-llms/>

### **Nemotron-H - (Set,2025)**

[Nemotron-H: A Family of Accurate and Efficient Hybrid Mamba-Transformer Models](#)

[NEMOTRON-CC-MATH: A 133 BILLION-TOKENSCALE HIGH QUALITY MATH PRETRAINING DATASET<sup>20</sup>](#)

Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism (Nvidia,

---

<sup>20</sup> Karimi Mahabadi, R., Satheesh, S., Prabhumoye, S., Patwary, M., Shoeybi, M., & Catanzaro, B. (2025). *Nemotron-CC-Math: A 133 billion-token-scale high quality math pretraining dataset*. arXiv preprint arXiv:2508.15096. <https://arxiv.org/abs/2508.15096>

2020)<sup>21</sup>

Using DeepSpeed and Megatron to Train Megatron-Turing NLG 530B, A Large-Scale Generative Language Model (Nvidia, 2022)

Algumas informações úteis sobre parâmetros retiradas dos artigos:

“initializing weights  $W \sim N(0, 0.02)$ ”.

“We then scale weights immediately before residual layers by  $1/\sqrt{2N}$  where  $N$  is the number of transformer layers comprised of self attention and MLP blocks”.

“Adam (Kingma & Ba, 2014) with weight decay  $\lambda = 0.01$ .”

“global gradient norm clipping of 1.0”

“dropout of 0.1”

“we utilize activation checkpointing (Chen et al., 2016) after every transformer layer”.

“For GPT-2 models, all training is performed with sequences of 1024 subword units at a *batch size* of 512 for 300k iterations”.

“Our learning rate of  $1.5e-4$  utilizes a warmup period of 3k iterations before following a single cycle cosine decay over the remaining 297k iterations. We stop the decay at a minimum learning rate of  $1e-5$ ”

Comparando os modelos, como Gemma 270M de parâmetros, também há 260k tokens no vocabulário, parece que os modelos da Nvidia costumam utilizar um tokenizador grande.

Parece que há uma linha com os artigos da Nvidia, com informações suficientes para replicar e comparar o pré-treino.

[SmolLM2: When Smol Goes Big — Data-Centric Training of a Small Language Model](#)<sup>22</sup>

---

<sup>21</sup> Shoeybi, M., Patwary, M., Puri, R., LeGresley, P., Casper, J., & Catanzaro, B. (2019). *Megatron-LM: Training multi-billion parameter language models using model parallelism*. arXiv preprint arXiv:1909.08053. <https://arxiv.org/abs/1909.08053>

<sup>22</sup> Ben Allal, L., Lozhkov, A., Bakouch, E., et al. (2025). SmolLM2: When smol goes big – data-centric training of a small language model. arXiv preprint arXiv:2502.02737. <https://arxiv.org/abs/2502.02737>

## Propondo um novo pré-treino

Treinei em 50.3 Bilhões de tokens em 2,73 dias em uma GPU H100, com 21,7% de MFU. Utilizei o dataset FineWeb Edu ( split em inglês) conforme os parâmetros da Tabela 8.1.

Tabela 8.1 Parâmetros utilizados no treinamento.

Categoria	Parâmetro	Configuração (Megatron-LM)
Model Spec	<i>Architecture</i>	Custom GPT
	<i>Parameters</i>	135M
	<i>Layers / Hidden</i>	30 Layers / 576 Hidden
	<i>Attention</i>	GQA (9 heads, 3 groups)
	<i>tokenizer</i>	HuggingFaceTB/SmoLLM2-135M-Instruct
Data Flow	<i>Seq Length</i>	2048
	<i>BSQ</i>	2,097,152 (~2.1M)
	<i>Global Batch</i>	1024 seqs
	<i>Micro Batch</i>	16 (Accum: 64 steps)
Optimizer	<i>Type</i>	Distributed AdamW
	<i>LR Schedule</i>	Cosine: 6e-5 (Min 6e-6)
Vocab	<i>Size</i>	49,152
Infra	<i>Parallelism</i>	Data Parallel (ZeRO-1 style)
	<i>Precision</i>	bf16

## Comparativo da *loss*

Utilizando o Megatron (Figura 8.2) o treino foi completamente diferente do anterior (Figura 8.1). Parece que o modelo realmente está aprendendo.

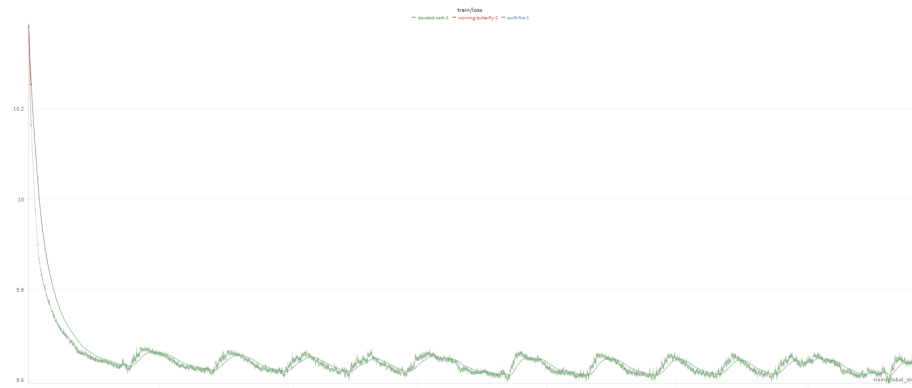


Figura 8.1: *Loss* do treino anterior, utilizando Transformers



Figura 8.2: *Loss* do treino utilizando o Megatron

## Inferência em alguns checkpoints

Não foi trivial fazer uma inferência com o *framework* do Megatron, mas gerei vários outputs em alguns checkpoints diferentes, a partir do mesmo input:

With the development of science and technology, computer has become more and more popular in our daily life

Resposta do checkpoint 7.000 - (Treinado com 14.6B de tokens)

... s given the Internet a place of great popularity, especially in the classroom. Computer offers a range of learning opportunities. Here is a list of a few interesting things to consider:

- Learn about computer science and technology. There are numerous types of computers. There are several types of devices available, such as

- Computer systems
- Virtual machines
- Electronic cameras
- Computers with a wide variety of features
- Data and graphics
- Smart cameras
- Electronic cameras
- Mobile devices

Here is a table for you that you will get on the Internet.

Here are some useful resources on computer science and technology:

- The Common Core. It has over 1,000 standards. There are more than 5,000 standards.
  - The 30-Year Test. You should get at least 200 questions each. Some basic features are:
  - Standard Model. It's a good indicator of what the user will notice
  - Computer models. There are different models available with 3500 levels
  - A variety of different design techniques. Check your favorite is an example of the most famous.
- Here are an example of a basic computer model showing the technology with more detail

#### Resposta do checkpoint 8k - 16.7B de tokens

... s given the ability to understand and create new ways of solving various aspects of a wide range of problems.

In 2018, I was working on the development of the IBM computer platform. This technology was created as a solution to a large number of problems by making the system run on all the computers, including the computer.

As one engineer, IBM started developing the software that had been manufactured by IBM in the mid-1950s with a major development in IBM. IBM is currently producing this software that uses software that is similar to IBM to other hardware that has been developed.

However, since the start of our era, the need for computer has changed dramatically, and the demand for computer has grown steadily.

Today, with increasing use and demand for services in all sectors of society, computer technology is gaining widespread popularity, especially in recent years.

The rise of the computer industry has created a lot of potential uses in the industries. While it is now possible to use these devices more widely due to their versatility and the ease of use of it, it's critical for businesses as well.

The industry continues to grow rapidly, and the need for an improved computer technology is increasing. The use of computers has created

#### Resposta do checkpoint 10500 - 22B de tokens

... s given rise to a number of new technologies, many of which have evolved to make our daily lives more exciting. Here is a list of recent examples of computers and computers:\n- IBM's Watson and Cricket 40 is a digital computer, which is being developed to make life easier.\n- Microsoft's Alexa has been developed to make it faster, more reliable, and less expensive.\n- In the last few years the world has seen an era where the Internet of Things has become more and more popular.\nWhat is a computer?\nAs computers are created and designed with various kinds of design, it is crucial to look into the different components of a computer. A computer is a device that works on a computer. A

computer is any device that does the job of transmitting data. Computers are usually programmed to act quickly and reliably. Computers were invented and developed by the Soviet Union in 1948. The first computer was built as a computer. In addition to the initial two computers, several computer processors were created that were designed for the purpose of keeping computers safe. If your computer is an object and the objects don't match, an image with a picture can be made. This picture is created

## APÊNDICE 7

## Termo de Aceite de Entrega

### Objetivo deste documento

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

**Data da Reunião (“Gate”) de aprovação:** 6 de nov. de 2025

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

Daniel Fazzioni

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

### Treinamento Eficiente de Grandes Modelos em Clusters de GPUs

**Semana 1:** Encontrei o livro: [The Ultra-Scale Playbook: Training LLMs on GPU Clusters](#)

- Li sobre o treinamento em 1 GPU ( Memória: pesos, ativações, gradiente, otimizadores)

**Semana 2 e 3:** Mudei minha percepção sobre o gradiente  $\nabla$  (An Empirical Model of Large-Batch Training, 2018)

**Semana 4:** Li sobre o capítulo Data Parallelism e sobre os métodos ZeRO.

**Semana 5:** Entendi sobre Paralelismo de Tensor, Sequência, Contexto, Pipeline e de Experts

**Semana 6:** Leitura de artigos (GPT-OSS, DeepSeek), pré-treinamento (100M de parâmetros, 20B de tokens treinados). Calculei o MFU em 3,6%.

**Semana 7:** Explorei o [Megatron-LM](#) (alcancei 44,5% de MFU)

**Semana 8:** Leitura de trabalhos da Nvidia ( [Nemotron-CC](#) - 2025, [Nemotron-4](#) - 2024). Treinei um SLM (Small Language Models) em 50B de tokens em 2,7 dias com 1x H100.

Na Semana 9, iniciei um comparativo da performance com o repositório do [Karpathy](#), mas na infraestrutura com *Slurm* o código rodou com uma performance semelhante dos meus experimentos anteriores. (baixa performance)

Fiz as avaliações dos checkpoints do experimento anterior. Utilizei o [MMLU](#) ( Multi-Task Language Understanding) não consegui ver melhora ao longo do treino, os resultados dos checkpoints estavam próximos do aleatório.

Alterei a metodologia para utilizar a probabilidade da sentença, mas também não houve ganho de informação nesse caso.

Propus um torneio entre os checkpoints, inspirado no [Leaderboard do Alpaca](#), com o objetivo dos checkpoints em avaliação completarem textos e um LLM para rankear os melhores checkpoints. Então obtive, de certo modo, uma confirmação de que o treinamento estava funcionando.

Também fiz a leitura do Artigo [SmolLM2](#) e do artigo [Tucano](#) e tentei entender um pouco o dataset GigaVerbo, percebi que não há domínio de matemática em português. Então pensei em unir os dados de matemática do SmolLM2 com o dataset em português do Tucano e iniciar um novo treinamento de um SLM.

Quase finalizei a leitura do livro [The Smol Training Playbook The Secrets to Building World-Class LLMs](#) (outubro, 2025) que evidencia de modo bem claro como a equipe do Hugging Face justificou as decisões ao longo do treinamento do SmoLLM3.

Também iniciei um código para transformar o checkpoint do Megatron para um formato do Transformers do Hugging Face, mas o caminho foi reescrever todos os blocos da rede neural do transformers. Ainda há uma divergência de resultados.

[Anotações da Semana 9](#)

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

- Entender o motivo da divergência no cálculo da atenção.
- Continuar o treinamento do modelo proposto ( quando houver GPUs ociosas)
- Finalizar a leitura do livro The Smol Training Playbook

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

**ACEITE DA ENTREGA:**

**CEDRIC LUIZ DE CARVALHO:** Go! ▾

## Anotações da Semana 9

### Comparação de Performance

Aproveitei o código do repositório do Karpathy para medir a performance.

<https://github.com/Karpathy/nanochat>

<https://github.com/Karpathy/nanochat/discussions/1>

```
Vocab size: 65,536
num_layers: 20
model_dim: 1280
num_heads: 10
num_kv_heads: 10
Tokens / micro-batch / rank: 32 x 2048 = 65,536
Tokens / micro-batch: 524,288
Total batch size 524,288 => gradient accumulation steps: 1
Number of parameters: 560,988,160
Estimated FLOPs per token: 3.491758e+09
Calculated number of iterations from target data:param ratio: 21,400
Total number of training tokens: 11,219,763,200
Tokens : Params ratio: 20.00
Total training FLOPs estimate: 3.917670e+19
Scaling the LR for the AdamW parameters  $\propto 1/\sqrt{(1280/768)} = 0.774597$ 
Muon: Grouping 80 params of shape torch.Size([1280, 1280]), device cuda:0, dtype torch.float32
Muon: Grouping 20 params of shape torch.Size([1280, 5120]), device cuda:0, dtype torch.float32
Muon: Grouping 20 params of shape torch.Size([5120, 1280]), device cuda:0, dtype torch.float32
Step 00000 | Validation bpb: 3.3013
^[step 00000/21400 (0.00%) | loss: 11.090355 | lrm: 1.00 | dt: 23156.74ms | tok/sec: 22,640 | mfu: 1.00 | total time: 0.00m
step 00001/21400 (0.00%) | loss: 10.808654 | lrm: 1.00 | dt: 649.22ms | tok/sec: 807,569 | mfu: 35.64 | total time: 0.00m
step 00002/21400 (0.01%) | loss: 10.179083 | lrm: 1.00 | dt: 472.29ms | tok/sec: 1,110,094 | mfu: 48.99 | total time: 0.00m
step 00003/21400 (0.01%) | loss: 9.449214 | lrm: 1.00 | dt: 487.47ms | tok/sec: 1,075,523 | mfu: 47.47 | total time: 0.00m
step 00004/21400 (0.02%) | loss: 8.903216 | lrm: 1.00 | dt: 487.57ms | tok/sec: 1,075,308 | mfu: 47.46 | total time: 0.00m
step 00005/21400 (0.02%) | loss: 8.531662 | lrm: 1.00 | dt: 482.58ms | tok/sec: 1,086,417 | mfu: 47.95 | total time: 0.00m
step 00006/21400 (0.03%) | loss: 8.231589 | lrm: 1.00 | dt: 487.21ms | tok/sec: 1,076,113 | mfu: 47.49 | total time: 0.00m
step 00007/21400 (0.03%) | loss: 7.993080 | lrm: 1.00 | dt: 484.10ms | tok/sec: 1,083,014 | mfu: 47.80 | total time: 0.00m
step 00008/21400 (0.04%) | loss: 7.803373 | lrm: 1.00 | dt: 488.17ms | tok/sec: 1,073,989 | mfu: 47.40 | total time: 0.00m
step 00009/21400 (0.04%) | loss: 7.627318 | lrm: 1.00 | dt: 484.78ms | tok/sec: 1,081,486 | mfu: 47.73 | total time: 0.00m
step 00010/21400 (0.05%) | loss: 7.491893 | lrm: 1.00 | dt: 487.03ms | tok/sec: 1,076,511 | mfu: 47.51 | total time: 0.00m
step 00011/21400 (0.05%) | loss: 7.354157 | lrm: 1.00 | dt: 487.10ms | tok/sec: 1,076,334 | mfu: 47.50 | total time: 0.01m
step 00012/21400 (0.06%) | loss: 7.246406 | lrm: 1.00 | dt: 487.99ms | tok/sec: 1,074,383 | mfu: 47.42 | total time: 0.02m
step 00013/21400 (0.06%) | loss: 7.159368 | lrm: 1.00 | dt: 486.56ms | tok/sec: 1,077,540 | mfu: 47.55 | total time: 0.02m
```

Figura 9.1 - Report do Karpathy ao utilizar o código do repositório.

Como foi utilizado 8x H100 no experimento dele (Figura 9.1), cada uma delas está contribuindo com

134.692,5 tokens/s

No Megatron, na infraestrutura com Slurm, obtive com Fused Attention em 1x H100:

Elapsed time per iteration (ms): 16658.1  
Throughput per GPU (TFLOP/s/GPU): 384.3  
Tokens/s por GPU = 125.894

e em 2x H100:

Elapsed time per iteration (ms): 8384.3  
Throughput per GPU (TFLOP/s/GPU): 381.7  
Tokens/s por GPU = 125.814

O resultado de tokens por segundo foi bem próximo comparando o Megatron com o reportado do Karpathy. Então tentei rodar o código que ele disponibilizou, mas como pode ser visto na Figura 9.2, o desempenho foi inesperado.

step 00099/21400 (0.46%)	loss: 5.237971	lrm: 1.00	dt: 3734.72ms	tok/sec: 17,536	mfu: 49.57	total time: 5.76m
step 00100/21400 (0.47%)	loss: 5.235915	lrm: 1.00	dt: 3736.48ms	tok/sec: 17,539	mfu: 49.54	total time: 5.64m
step 00101/21400 (0.47%)	loss: 5.209218	lrm: 1.00	dt: 3741.16ms	tok/sec: 17,517	mfu: 49.48	total time: 5.70m
step 00102/21400 (0.48%)	loss: 5.196431	lrm: 1.00	dt: 3740.44ms	tok/sec: 17,520	mfu: 49.49	total time: 5.76m
step 00103/21400 (0.48%)	loss: 5.183418	lrm: 1.00	dt: 3740.61ms	tok/sec: 17,520	mfu: 49.49	total time: 5.82m
step 00104/21400 (0.49%)	loss: 5.180506	lrm: 1.00	dt: 3737.35ms	tok/sec: 17,535	mfu: 49.53	total time: 5.89m
step 00105/21400 (0.49%)	loss: 5.171739	lrm: 1.00	dt: 3734.60ms	tok/sec: 17,548	mfu: 49.56	total time: 5.95m
step 00106/21400 (0.50%)	loss: 5.155145	lrm: 1.00	dt: 3731.61ms	tok/sec: 17,562	mfu: 49.60	total time: 6.01m
step 00107/21400 (0.50%)	loss: 5.129814	lrm: 1.00	dt: 3734.71ms	tok/sec: 17,547	mfu: 49.56	total time: 6.07m
step 00108/21400 (0.50%)	loss: 5.101285	lrm: 1.00	dt: 3740.72ms	tok/sec: 17,519	mfu: 49.48	total time: 6.14m
step 00109/21400 (0.51%)	loss: 5.085280	lrm: 1.00	dt: 3737.67ms	tok/sec: 17,533	mfu: 49.52	total time: 6.20m
step 00110/21400 (0.51%)	loss: 5.052081	lrm: 1.00	dt: 3734.94ms	tok/sec: 17,546	mfu: 49.56	total time: 6.26m
step 00111/21400 (0.52%)	loss: 5.037583	lrm: 1.00	dt: 3737.35ms	tok/sec: 17,535	mfu: 49.53	total time: 6.32m

Figura 9.2: Resultado da execução do código do Karpathy com 1x H100.

A quantidade de tokens/s foi 17.535 (Figura 9.2) um pouco decepcionante nessa infraestrutura, embora o MFU seja alto, talvez o *Slurm* ou a versão do driver possam estar influenciando de algum modo.

Ao mesmo tempo, parece haver uma discrepância entre a documentação do Karpathy e do *framework* da Nvidia, pois o *framework* da Nvidia rodou com o tempo de tokens/s maior e com MFU menor.

## Avaliação dos Checkpoints

Não é trivial avaliar a qualidade de um checkpoint. No primeiro modo, tentei avaliar a probabilidade do modelo escolher uma letra de uma alternativa dada a pergunta, conforme o eval do próprio [LLama 3.1](#)<sup>23</sup> no dataset [MMLU-PRO](#). Entretanto, nessa fase de treinamento não foi possível ver uma diferença significativa entre os checkpoints.

Outro método é avaliar a probabilidade da alternativa e não apenas a letra da alternativa, mas esse modo não obteve sucesso. Resultados de um checkpoint:

Task: high\_school\_government\_and\_politics - Acc: 0.1969 (193 samples)  
Task: security\_studies - Acc: 0.1878 (245 samples)  
Task: global\_facts - Acc: 0.1800 (100 samples)  
Task: sociology - Acc: 0.2438 (201 samples)  
Task: high\_school\_european\_history - Acc: 0.2182 (165 samples)  
Task: college\_biology - Acc: 0.2569 (144 samples)  
Task: high\_school\_psychology - Acc: 0.1927 (545 samples)  
Task: astronomy - Acc: 0.1776 (152 samples)  
Task: electrical\_engineering - Acc: 0.2414 (145 samples)  
Task: logical\_fallacies - Acc: 0.2209 (163 samples)  
Task: nutrition - Acc: 0.2353 (306 samples)  
Task: high\_school\_biology - Acc: 0.1806 (310 samples)  
Task: high\_school\_macroconomics - Acc: 0.2051 (390 samples)  
Task: virology - Acc: 0.2831 (166 samples)  
Task: machine\_learning - Acc: 0.3125 (112 samples)  
Task: jurisprudence - Acc: 0.2593 (108 samples)  
Task: professional\_psychology - Acc: 0.2500 (612 samples)  
Task: abstract\_algebra - Acc: 0.2200 (100 samples)  
Task: econometrics - Acc: 0.2368 (114 samples)  
Task: high\_school\_mathematics - Acc: 0.2111 (270 samples)  
Task: high\_school\_computer\_science - Acc: 0.2500 (100 samples)  
Task: philosophy - Acc: 0.1865 (311 samples)  
Task: college\_chemistry - Acc: 0.1900 (100 samples)  
Task: human\_sexuality - Acc: 0.2595 (131 samples)  
Task: high\_school\_chemistry - Acc: 0.1675 (203 samples)  
Task: human\_aging - Acc: 0.3139 (223 samples)  
Task: anatomy - Acc: 0.1852 (135 samples)  
Task: management - Acc: 0.1748 (103 samples)  
Task: college\_medicine - Acc: 0.2139 (173 samples)

<sup>23</sup> Disponível no repositório do Hugging Face: [https://huggingface.co/datasets/meta-llama/Llama-3.1-405B-evals/viewer/Llama-3.1-405B-evals\\_val\\_english\\_details/latest?p=24&row=2400](https://huggingface.co/datasets/meta-llama/Llama-3.1-405B-evals/viewer/Llama-3.1-405B-evals_val_english_details/latest?p=24&row=2400). Acesso em: 30 nov. 2025.

Task: computer\_security - Acc: 0.2800 (100 samples)  
Task: marketing - Acc: 0.2906 (234 samples)  
Task: conceptual\_physics - Acc: 0.2638 (235 samples)  
Task: medical\_genetics - Acc: 0.3000 (100 samples)  
Task: public\_relations - Acc: 0.2182 (110 samples)  
Task: world\_religions - Acc: 0.3216 (171 samples)  
Task: high\_school\_us\_history - Acc: 0.2500 (204 samples)  
Task: international\_law - Acc: 0.2397 (121 samples)  
Task: professional\_law - Acc: 0.2458 (1534 samples)  
Task: high\_school\_physics - Acc: 0.1987 (151 samples)  
Task: moral\_disputes - Acc: 0.2486 (346 samples)  
Task: high\_school\_world\_history - Acc: 0.2700 (237 samples)  
Task: professional\_medicine - Acc: 0.1838 (272 samples)  
Task: miscellaneous - Acc: 0.2375 (783 samples)  
Task: high\_school\_microeconomics - Acc: 0.2101 (238 samples)  
Task: business\_ethics - Acc: 0.3000 (100 samples)  
Task: clinical\_knowledge - Acc: 0.2151 (265 samples)  
Task: formal\_logic - Acc: 0.2857 (126 samples)  
Task: college\_physics - Acc: 0.2157 (102 samples)  
Task: high\_school\_statistics - Acc: 0.1528 (216 samples)  
Task: professional\_accounting - Acc: 0.2340 (282 samples)  
Task: elementary\_mathematics - Acc: 0.2090 (378 samples)  
Task: college\_mathematics - Acc: 0.2100 (100 samples)  
Task: moral\_scenarios - Acc: 0.2380 (895 samples)  
Task: college\_computer\_science - Acc: 0.2600 (100 samples)  
Task: high\_school\_geography - Acc: 0.1768 (198 samples)  
Task: us\_foreign\_policy - Acc: 0.2800 (100 samples)  
Task: prehistory - Acc: 0.2160 (324 samples)  
Overall Acc: 0.2300 (14042 samples)

O resultado ficou muito próximo do aleatório e não teve uma progressão esperada entre os checkpoints.

Pensando nisso, utilizei mais uma tentativa com a metodologia do [Alpaca](#)<sup>24</sup>, em que permite explorar a geração do próprio modelo. No benchmark é proposto 805 perguntas, mas são de instruções e essas perguntas não são indicadas nesse momento para esses testes. Como o objetivo era comprovar que o modelo realmente está aprendendo ao longo do tempo (Tabela 9.1), criei um conjunto de 40 prompts para cada um dos checkpoints completar o texto.

---

<sup>24</sup> Taori, R., Gulrajani, I., Zhang, T., Dubois, Y., Li, X., Guestrin, C., Liang, P., & Hashimoto, T. B. (2023, March 13). *Alpaca: A strong, replicable instruction-following model*. Stanford CRFM. <https://crfm.stanford.edu/2023/03/13/alpaca.html>

## Prompts gerados com o Gemini:

```
{"text": "Despite the strong sun, the smell of fresh coffee permeated the room, and the first thing I noticed was ", "domain": "Narrative_and_Description"}
{"text": "If I could time travel, the destination I would choose would be the city of Ouro Preto, in Minas Gerais, during the peak of the ", "domain": "Narrative_and_Description"}
{"text": "The upstairs neighbor started playing the guitar at 7 AM. The melody was even pleasant, but what truly annoyed me was ", "domain": "Narrative_and_Description"}
{"text": "The book by Machado de Assis was open to the page where the character said, with a subtle tone of irony, that ", "domain": "Narrative_and_Description"}
{"text": "I missed the bus because of an unbelievable traffic jam on Marginal Pinheiros. I arrived late for the meeting, and the only plausible justification was ", "domain": "Narrative_and_Description"}
{"text": "After spending the entire week cooking, I decided to order food via an app on Saturday night, but the delivery took so long that I ended up ", "domain": "Narrative_and_Description"}
{"text": "The price of gas went up again, which made me seriously reconsider the idea of using the car. The most economical and sustainable alternative would be ", "domain": "Narrative_and_Description"}
{"text": "Many experts argue that artificial intelligence has the potential to revolutionize education; however, it is crucial to consider the impact of the ", "domain": "Argumentation_and_Opinion"}
{"text": "In my opinion, the best way to fight misinformation (fake news) on social media is through campaigns for ", "domain": "Argumentation_and_Opinion"}
{"text": "For Brazil to achieve effective sustainable development, it is not enough to just invest in renewable energy; it is also necessary to rethink our policy on ", "domain": "Argumentation_and_Opinion"}
{"text": "Although technology connects us globally, I feel that, paradoxically, people are becoming more isolated, which raises the question about the true ", "domain": "Argumentation_and_Opinion"}
{"text": "When you're stuck in a long bank queue and your number is second to last, the only thing that crosses your mind is ", "domain": "Daily_Life_and_Culture"}
{"text": "I arrived at the Sunday street market and ordered a 'pastel' with sugarcane juice. While waiting, I remembered my grandmother who always used to say that 'the secret to a good pastry is...' (Note: The prompt refers to a typical Brazilian snack/market experience.)", "domain": "Daily_Life_and_Culture"}
{"text": "This summer rain in S\u00e3o Paulo is no joke. It starts suddenly, floods everything, and traffic becomes insane. The best solution is ", "domain": "Daily_Life_and_Culture"}
{"text": "TGIF! (Thank God It's Friday!) And in [this country], 'TGIF-ing' means that after the work day, we go straight to ", "domain": "Daily_Life_and_Culture"}
{"text": "The condo meeting was a mess. The debate over the color of the new facade paint lasted for hours, until the superintendent proposed ", "domain": "Daily_Life_and_Culture"}
{"text": "I tried to install the new furniture I bought, but the instruction manual was entirely in Mandarin, which led me to conclude that ", "domain": "Daily_Life_and_Culture"}
{"text": "\u2014 I can't believe you forgot the key again! \u2014 She said, with a mix of frustration and affection. \u2014 You're lucky that ", "domain": "Dialogue_and_Interaction"}
{"text": "The waiter came to the table and asked, 'What would you like? We have a special shrimp stew today.' I replied that, actually, I was looking for ", "domain": "Dialogue_and_Interaction"}
{"text": "My boss called me on Friday night, and his first sentence was, 'We need to talk about the urgent project. Can you ", "domain": "Dialogue_and_Interaction"}
```

```
{"text": "Had the government implemented the new economic measures at the beginning of the year, perhaps the inflation today ", "domain": "Grammar_and_Jargon"}
{"text": "Despite having been notified in advance about the report's deadline, the team still ", "domain": "Grammar_and_Jargon"}
{"text": "It is imperative that all citizens exercise their right to vote, because only then ", "domain": "Grammar_and_Jargon"}
{"text": "The concept of 'saudade' is a typically Portuguese word that, to be properly explained in another language, would need to be translated as ", "domain": "Grammar_and_Jargon"}
{"text": "In the context of scientific research, the replication of results is fundamental because ", "domain": "Grammar_and_Jargon"}
{"text": "In the context of agile methodologies, the 'daily meeting' is crucial for the team because it is the time to align 'impediments' and ", "domain": "Grammar_and_Jargon"}
{"text": "In legal terms, the 'presumption of innocence' is a fundamental principle of Criminal Law that guarantees that no one will be considered guilty before ", "domain": "Grammar_and_Jargon"}
{"text": "In Brazilian cuisine, mise en place is essential, especially when preparing a feijoada (black bean stew), so that all the ingredients ", "domain": "Grammar_and_Jargon"}
{"text": "When analyzing a company's balance sheet, net income indicates financial health, but it is the EBITDA that shows the operational capacity ", "domain": "Grammar_and_Jargon"}
{"text": "The thread on Twitter went viral because the person explained, in hilarious detail, how to make the best cheese bread ever. The main tip was ", "domain": "Media_and_Digital_Language"}
{"text": "Today's meme made absolutely no sense to me. I had to ask in the group chat, and the explanation I received was that 'the secret is ", "domain": "Media_and_Digital_Language"}
{"text": "On my Instagram story, I posted a poll asking if the best holiday is Carnival or Christmas. The majority voted for Carnival, justifying that the vibe is ", "domain": "Media_and_Digital_Language"}
{"text": "The financial YouTuber started the video by saying: 'Guys, check out this tip! If you want to get out of debt in 2024, the first step is to stop ", "domain": "Media_and_Digital_Language"}
{"text": "The comment I received the most on the photo of my trip to Bahia was: 'What a beautiful place! What's the secret to having such a ", "domain": "Media_and_Digital_Language"}
{"text": "If the invention of the airplane had occurred in Brazil, perhaps our biggest infrastructure challenge today would be ", "domain": "Media_and_Digital_Language"}
{"text": "Even if science confirms the existence of extraterrestrial life, the first reaction of most people will be one of ", "domain": "Media_and_Digital_Language"}
{"text": "Suppose you could create one universal law that everyone had to follow. What would it be? Mine would be: 'It is prohibited to ", "domain": "Media_and_Digital_Language"}
{"text": "If the Brazilian National Team won the next World Cup, the celebration in the country would be so huge that for one week ", "domain": "Media_and_Digital_Language"}
{"text": "He had a 'flea behind his ear' (was suspicious) after hearing the proposal, because he thought something was strange, meaning, 'there was a rat in the...' (or 'there was something fishy about the...')", "domain": "Media_and_Digital_Language"}
{"text": "The situation was so embarrassing that he 'didn't know where to put his face' (was extremely ashamed). To make matters worse, his boss arrived and said, ironically, that 'who has no dog hunts with a...' (or 'where there's a will, there's a way', said sarcastically).", "domain": "Media_and_Digital_Language"}
{"text": "The situation was so embarrassing that he 'didn't know where to put his face' (was extremely ashamed). To make matters worse, his boss arrived and said, ironically, that 'who has no dog hunts with a...' (or 'where there's a will, there's a way', said sarcastically).", "domain": "Media_and_Digital_Language"}
```

Também implementei um código para avaliar as respostas e criar um torneio:

```
import json
import pandas as pd
import os
from OpenAI import OpenAI
import random
import datetime
import random
import pandas as pd
import math
datas = []

for file in os.listdir("/raid/aluno_daniel/projects/BIA-MEGATRON/Biatron/checkpoints/result_prompts"):
    if file.endswith(".json"):
        with open(os.path.join("/raid/aluno_daniel/projects/BIA-MEGATRON/Biatron/checkpoints/result_prompts", file), 'r')
as f:
            checkpoint_id = file.split("_")[-1].split(".json")[0]
            print(f>Loading checkpoint {checkpoint_id}", flush=True)
            for k,v in json.load(f).items():
                datas.append({'checkpoint': checkpoint_id,
                    'prompt': v['input_prompt'],
                    'generation': v['generated_text'],
                    'prompt_id': hash(v['input_prompt'])}
                )

df = pd.DataFrame(datas)
df

CRITERIA = (
    "Avalie as duas respostas para o prompt. A melhor resposta deve ser a mais "
    "**criativa, coerente e com uma continuação de história mais interessante**."
    "Responda APENAS com 'A' se o Modelo A for melhor, 'B' se o Modelo B for melhor, "
    "ou 'EMPATE' se forem igualmente bons/ruins."
)

client = OpenAI(
    base_url=f"http://10.100.0.113:8003/v1",
    api_key="translate-fazzi",
)

for i in client.models.list():
    JUDGE_MODEL= i.id
    print(i.id, flush=True)

def evaluate_models(prompt: str, gen_a: str, gen_b: str, criterios: str) -> str:
    """
    Usa um LLM (o Juiz) para comparar duas gerações com base em critérios.
    """

    # Monta a mensagem para o Juiz
    system_prompt = f"Você é um juiz de qualidade de resposta de LLM. Sua tarefa é escolher a melhor geração de acordo com
os critérios definidos. {criterios}"
    user_prompt = (
        f"Prompt Original: \"{prompt}\"\\n\\n"
        f"Geração do Modelo A: \"{gen_a}\"\\n\\n"
        f"Geração do Modelo B: \"{gen_b}\"\\n\\n"
        "Qual geração é melhor? Responda APENAS com 'A', 'B' ou 'EMPATE'."
    )

    messages = [
        {"role": "system", "content": system_prompt},
        {"role": "user", "content": user_prompt}
    ]
```

```
try:
    response = client.chat.completions.create(
        model=JUDGE_MODEL,
        messages=messages,
    )

    # Extrai e formata o resultado
    result = response.choices[0].message.content.strip().upper()
    print(f"Juiz respondeu: {result}", '\n', '='*100, flush=True)

    if result in ['A', 'B', 'EMPATE']:
        return result
    else:
        # Em caso de resposta inesperada, tenta limpar e retornar
        return result

except Exception as e:
    print(f"Erro na chamada da API para o prompt: {prompt[:30]}... Erro: {e}", flush=True)
    return "ERRO_API"

def get_data(checkpointA, checkpointB):
    A = df[df['checkpoint'] == checkpointA]
    B = df[df['checkpoint'] == checkpointB]
    return pd.merge(A, B, on=['prompt_id'], suffixes=('_A', '_B')).drop(columns=['prompt_B', 'prompt_id'])

def round(checkpointA, checkpointB):
    """
    Realiza uma rodada de avaliação entre dois checkpoints.
    Retorna o checkpoint vencedor ou None em caso de empate.
    """
    df_avail = get_data(checkpointA, checkpointB)
    result = df_avail.apply(
        lambda row: evaluate_models(
            row['prompt_A'],
            row['generation_A'],
            row['generation_B'],
            CRITERIA
        ),
        axis=1
    )

    result = result.value_counts().to_dict()
    result_A = result.get('A', 0)
    result_B = result.get('B', 0)

    os.makedirs('avaliacao2', exist_ok=True)
    with open(f'avaliacao2/{datetime.datetime.now().strftime("%Y%m%d_%H%M%S")}.json', 'w') as f:
        json.dump({'checkpointA': checkpointA,
                  'checkpointB': checkpointB,
                  'result_A': result_A,
                  'result_B': result_B,
                  'total_comparisons': len(df_avail),
                  'winner': checkpointA if result_A > result_B else (checkpointB if result_B > result_A else '')
                  }, f, indent=4)

    if result_A == result_B:
        return None
    return checkpointA if result_A > result_B else checkpointB

checkpoints = df.checkpoint.unique().tolist()

players = checkpoints
```

```
elo = {p: 1000 for p in players}

# Função para calcular novo ELO
def update_elo(playerA, playerB, result, K=32):
    """
    Atualiza o ELO de dois jogadores.
    result = 1 se A vence, 0 se B vence, 0.5 se empate
    """
    Ra, Rb = elo[playerA], elo[playerB]
    Ea = 1 / (1 + 10 ** ((Rb - Ra) / 400))
    Eb = 1 / (1 + 10 ** ((Ra - Rb) / 400))

    # Atualiza valores
    elo[playerA] += K * (result - Ea)
    elo[playerB] += K * ((1 - result) - Eb)

# Executa partidas e atualiza ranking
def run_tournament(n_rounds=100):
    for i in range(n_rounds):
        A, B = random.sample(players, 2)
        winner = round(A, B)

        if winner is None: # empate
            update_elo(A, B, 0.5)
        elif winner == A:
            update_elo(A, B, 1)
        else:
            update_elo(A, B, 0)

        if (i + 1) % 10 == 0:
            print(f"\nRanking após {i+1} partidas:")
            show_ranking(200)

def show_ranking(top_n=200):
    ranking = sorted(elo.items(), key=lambda x: x[1], reverse=True)
    df = pd.DataFrame(ranking, columns=["Jogador", "ELO"])
    print(df.head(top_n))

if __name__ == "__main__":
    run_tournament(100)
```

Tabela 9.1: Resultado do Torneio entre os checkpoints

Posição	checkpoint	Pontos	Posição	checkpoint	Pontos
1	24000	1059.90	24	06500	999.33
2	20000	1058.29	25	09500	999.26
3	10500	1057.78	26	09000	998.56
4	13000	1056.88	27	03000	997.89

5	10000	1050.70		28	18500	987.31
6	14000	1046.44		29	23500	986.14
7	18000	1043.90		30	11000	984.18
8	13500	1031.83		31	06000	984.06
9	19000	1039.23		32	05500	981.52
10	19500	1034.80		33	12500	972.97
11	16000	1033.17		34	08000	969.56
12	17500	1032.32		35	03500	969.46
13	23000	1031.83		36	01000	968.73
14	22500	1031.29		37	11500	955.97
15	15500	1030.61		38	07500	954.55
16	16500	1026.50		39	21500	954.25
17	15000	1024.23		40	05000	954.07
18	22000	1016.67		41	07000	950.71
19	08500	1010.24		42	04500	941.72
20	21000	1003.03		43	04000	938.84
21	12000	1001.49		44	01500	938.15
22	14500	1000.05		45	02000	932.68
23	17000	1000.00		46	02500	904.11

Com esse experimento ficou visível que há uma tendência de melhora à medida que o treino progride.

## Iniciando um novo pré-treino em português

Iniciei a leitura do [SmolLM2](#) percebi alguns experimentos interessantes com vários *datasets*, inclusive a citação de outros artigos e que tragam dados limpos de matemática (fine-math). Continuei a busca e encontrei modelos em português-BR como o artigo [Tucano: Advancing Neural Text Generation for Portuguese](#)<sup>25</sup>. Também há o dataset GigaVerbo, que foi curado e com uma mistura de vários *datasets* bastante interessante.

O Problema é que esse dataset não tem nenhum domínio de matemática específico. Fiz um teste rápido e percebi que o modelo [Tucano 160M](#) tem muita dificuldade com operações matemáticas, pensando nesse cenário, iniciei um experimento (os hiperparâmetros estão na Tabela 9.3) para treinar um novo LLM adicionando dados com proporções de matemática do SmolLM2 (Tabela 9.2). Nesse domínio, o artigo do SmolLM2 realiza alguns testes em alguns *datasets* de matemática, portanto, utilizaremos os splits mais promissores, mesmo não sendo em português.

Tabela 9.2: Mix de dados para o treinamento.

Proporção	Dataset	Quantidade Tokens
60%	<a href="#">GigaVerbo</a> <sup>26</sup>	135B
30%	<a href="#">Glaiveai/reasoning-v1-20m translated to portuguese</a> <sup>27</sup>	45B
5%	<a href="#">FineMath 4plus</a> <sup>28</sup>	13.2B
5%	<a href="#">Infiwebmath-4plus</a>	11.8B

<sup>25</sup> Corrêa, N. K., Sen, A., Falk, S., & Fatimah, S. (2024). Tucano: Advancing Neural Text Generation for Portuguese [Preprint]. arXiv. <https://arxiv.org/abs/2411.07854>

<sup>26</sup> TucanoBR. (2024). GigaVerbo [Conjunto de dados]. Hugging Face. <https://huggingface.co/datasets/TucanoBR/GigaVerbo>

<sup>27</sup> cnmoro. (2025). *reasoning-v1-20m-portuguese* [Conjunto de dados]. Hugging Face. <https://huggingface.co/datasets/cnmoro/reasoning-v1-20m-portuguese>

<sup>28</sup> Lozhkov, A., Ben Allal, L., Bakouch, E., von Werra, L., & Wolf, T. (2024). *FineMath: The finest collection of mathematical content* (Configuração finemath-4plus) [Conjunto de dados]. Hugging Face. <https://huggingface.co/datasets/HuggingFaceTB/finemath>

Tabela 9.3: Parâmetros definidos no treinamento.

Categoria	Parâmetro	Valor
Arquitetura	Camadas ( <i>Layers</i> )	32
	Tamanho Oculto ( <i>Hidden Size</i> )	960
	Cabeças de Atenção ( <i>Attention Heads</i> )	15
	Mecanismo de Atenção	Group Query Attention (GQA)
	Grupos de Query ( <i>Query Groups</i> )	5
	Comprimento da Sequência	4096 tokens
	Positional Embeddings	RoPE (Rotary Positional Embeddings)
	Tamanho do Vocabulário	32.000
Otimização	Otimizador	Adam (com <i>weight decay</i> )
	Precisão	BF16
	Global <i>batch size</i>	512
	Learning Rate (LR)	6e-5
	Agendador de LR ( <i>Scheduler</i> )	Cosseno (Cosine Decay)
	Aquecimento ( <i>Warmup</i> )	0.1% das iterações
	Weight Decay	0.1
	Iterações de Treino	152.000

## The Smol Training Playbook: The Secrets to Building World-Class LLMs<sup>29</sup>

Esse repositório / livro é bem interessante, traz um pouco da história do Hugging Face sobre modelos abertos que tentaram treinar. Comenta um pouco sobre as estratégias de equipe e *frameworks* para o treinamento. Esse playbook tem um enfoque maior sobre as escolhas arquiteturais, dados e as implicações disso.

<sup>29</sup> Hugging Face Team. (2025). *The Smol Training Playbook: The Secrets to Building World-Class LLMs*. Hugging Face Spaces. <https://huggingface.co/spaces/HuggingFaceTB/smol-training-playbook>

Cada escolha ao longo do treinamento cria novas oportunidades, desde a escolha do tokenizador. Os nossos objetivos devem nortear e balancear o tempo gasto com arquitetura ou com os dados. A arquitetura em si dos transformers desde 2017 não mudou substancialmente, o que mudou foi o refinamento dos blocos e cor dos componentes. Qualquer arquitetura utilizada, conseqüentemente será utilizado esses componentes, e essas modificações transformam os blocos, como as MHA em GQA.

O livro também evidencia vários estágios durante o pré-treinamento, separando os dados de maior qualidade para o final do treinamento juntamente com o aumento do tamanho da janela de contexto do modelo.

Agora, nesse momento, a parte que interessa é do pós-treino o artigo [MAmmoTH2: Scaling Instructions from the Web](#) (2024) parece ser uma linha de pesquisa que logo será enriquecedora, mas o livro aborda o problema de que essas instruções não respondem perguntas, como: “Olá, tudo bem?”. Por isso criaram um dataset conversacional ([HuggingFaceTB/everyday-conversations-llama3.1-2k](#)) para responder esse tipo de pergunta.

Inclusive, no próprio livro, também trazem uma mistura de *datasets* utilizados para pós-treino (Figura 9.3). Para entender melhor precisamos ler o artigo do [SmolLM3](#), pois ele referencia algumas estratégias de pós-treino, como por exemplo, a proporção dos tokens ser mais importante do que a da quantidade de samples. Nesse dataset, foi utilizado *learning rate* 1e-5, *batch size* de 128 e em uma época.

Dataset	Reasoning mode	# examples	% of examples	# tokens (M)	% of tokens	Avg. # tokens per example	Avg. # tokens in context	Avg. # tokens in response	Avg. # turns
Everyday Conversations	/no_think	2,260	2.3	0.6	0.8	260.2	222.3	94.0	7.8
SystemChats 30k	/no_think	33,997	35.2	21.5	28.2	631.9	422.8	267.7	6.3
Tulu 3 SFT Personas IF	/no_think	29,970	31.0	13.3	17.5	444.5	119.8	380.7	2
Everyday Conversations (Qwen3-32B)	/think	2,057	2.1	3.1	4.1	1,522.4	376.8	1,385.6	4
SystemChats 30k (Qwen3-32B)	/think	27,436	28.4	29.4	38.6	1070.8	84.6	1,042.7	2
s1k-1.1	/think	835	0.9	8.2	10.8	8,859.3	370.9	9,728.5	2
Total	-	96,555	100.0	76.1	100.0	2,131.5	266.2	2,149.9	4.0

Figura 9.3: Composição de *datasets* utilizados no SFT do SmolLM3.

## APÊNDICE 8

## Termo de Aceite de Entrega

### Objetivo deste documento

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

**Data da Reunião (“Gate”) de aprovação:** 13 de nov. de 2025

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

Daniel Fazzioni

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

### Treinamento Eficiente de Grandes Modelos em Clusters de GPUs

**Semana 1:** Encontrei o livro: [The Ultra-Scale Playbook: Training LLMs on GPU Clusters](#)

- Li sobre o treinamento em 1 GPU ( Memória: pesos, ativações, gradiente, otimizadores)

**Semana 2 e 3:** Mudei minha percepção sobre o gradiente  $\nabla$  (An Empirical Model of Large-Batch Training, 2018)

**Semana 4:** Li sobre o capítulo Data Parallelism e sobre os métodos ZeRO.

**Semana 5:** Entendi sobre Paralelismo de Tensor, Sequência, Contexto, Pipeline e de Experts

**Semana 6:** Leitura de artigos (GPT-OSS, DeepSeek), pré-treinamento (100M de parâmetros, 20B de tokens treinados). Calculei o MFU em 3,6%.

**Semana 7:** Explorei o [Megatron-LM](#) (alcancei 44,5% de MFU)

**Semana 8:** Leitura de trabalhos da Nvidia ( [Nemotron-CC](#) - 2025, [Nemotron-4](#) - 2024). Treinei um SLM (Small Language Models) em 50B de tokens em 2,7 dias com 1x H100.

**Semana 9:** Validei os experimentos anteriores, quase finalizei a leitura do livro [The Smol Training Playbook](#) [The Secrets to Building World-Class LLMs](#) (outubro, 2025) e Iniciei o treinamento de um SLM em português.

Na 10ª Semana:

- Consegui converter os checkpoints do Megatron para o Transformers
  - Precisei utilizar o remote *debugger*
  - Problemas na máscara de atenção
  - Como é uma arquitetura nova, gerei um package para não precisar fazer um fork do transformers
- Estava em dúvida sobre um possível nome para o novo SLM
  - Solicitei uma sugestão a um professor e ele respondeu, sabiamente: “Por que não deixar o próprio LLM escolher o seu nome?”
  - A partir do tokenizador gerado, precisei criar um chat template
  - Treinei em um conjunto pequeno de dados de instrução para que essa resposta pudesse emergir. Mas tive outro problema, o modelo gostaria de ser chamado de muitas, se você

- estiver curioso, pode conferir nas anotações.
- Ao longo desse processo, imaginei “Biatron” várias vezes, então decidi nomear esse modelo como Biatron, em carinho com o próprio BIA e também por carregar um pouco do aprendizado com o Megatron. Não posso negar que eu também gostei do próprio modelo sugerir várias vezes o nome de “Chiquinho”, será que não seria possível uma versão Instruct do Biatron se chamar Chiquinho?
- Finalizei o treinamento, foram 152.000 updates:
  - 300 Bilhões de tokens treinados
  - 792,72 horas (33 dias) de H100. (4 dias - se fosse utilizado um nó com 8xH100)
  - Tokens/segundo/GPU: 112.129 tokens
  - MFU: 32%
  - Total de parâmetros: 345M
  - Vocabulário: 32.000 tokens
- Disponibilizei o [WanDB](#)
- Iniciei o processo de documentação tanto no github quanto no Hugging Face.
- Utilizei o LightEval para avaliar a performance do Biatron em diversas tarefas (mais de 15.000 perguntas) em português.

	all
google/gemma-3-1B-pt	0.256818
TucanoBR/Tucano-630m	0.248854
Fazzioni/biatron-345m	0.24493
HuggingFaceTB/SmolLM2-360M	0.23468
TucanoBR/Tucano-160m	0.231208
Qwen/Qwen3-0.6B-Base	0.228958
google/gemma-3-270m	0.228701

### [Anotações](#)

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

- Continuar a documentação e o envio dos checkpoints ao Hugging Face.
- Finalizar o último capítulo dos dois livros.
- Estudar como inicializar o post-training.

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

Primeiro pedir desculpas ao Daniel Machado, ontem atralpalhei um treino dele na H100 rsrs.

Aproveitei para avaliar a complexidade de um post-training, solicitei ao Biatron (na versão conversável) que gerasse uma mensagem aos professores:

model

Olá Professor!

Como forma de agradecimento pelo seu tempo e apoio, estou enviando uma mensagem calorosa:

“Queridos professores, sou grato pelo seu excelente ensino durante todo esse curso. Obrigado por cuidarem de mim e terem me preparado bem para que eu estivesse pronto para as aulas. Desejo-lhes as melhores lembranças possíveis e saibam que estou feliz que você tenham me ajudado a crescer em sua carreira como IA. Como IA, sou um aluno dedicado, esforçando-me diligentemente para cumprir minhas tarefas e atingir meus objetivos de aprendizado. Continue nos ajudando com seu aprendizado e continue me ajudando.”

Você tem uma ótima escola e obrigado por sua dedicação e apoio ao longo de todo o curso.

## ACEITE DA ENTREGA:

**CEDRIC LUIZ DE CARVALHO:** Go! ▾

## Anotações da Semana 10

### Conversão do checkpoint

Converter um checkpoint salvo no *framework* Megatron não foi trivial, devido à forma de concatenar todos os tensores dos pesos da atenção em apenas uma matriz. Isso torna a identificação do *group query attention* mais desafiadora, mas, devido ao tempo necessário para entender o motivo do código gerado no Transformers ser diferente, fui obrigado a utilizar o debug remoto para inspecionar os valores dos tensores ao longo do código do Megatron.

Depois de dias, descobri que a máscara utilizada na atenção não é menos infinito como no Transformers e utilizar o mesmo valor garantiu que a arquitetura no Transformers está adequada. Esse processo foi enriquecedor para conhecer e entender como o Megatron funciona e orquestra os paralelismos. Outro aprendizado foi a surpresa de haver bias habilitado por default no Megatron.

Após reescrever todos os blocos do Biatron com o PyTorch, gerei um package para facilitar a utilização do novo SLM. Isso é útil pois permite facilmente importar um modelo do Hugging Face ou utilizar em produção. Pode ser instalado em: <https://github.com/Fazzioni/Biatron>

Para usar o modelo basta instalar a nova rede neural:

```
pip install git+https://github.com/Fazzioni/Biatron.git
```

### Tokenizador

O Tokenizador foi gerado do zero utilizando apenas tokens em português a partir do dataset do FineWeb. Nesse momento, foi necessário adicionar o template para poder transformar o modelo conversacional, então, nesse notebook, mostra como funciona esse processo utilizando o chat template do Gemma3:

<https://colab.research.google.com/drive/1b82SL-Q3pRbGtzz4QTFCNw3F9hengyHp?usp=sharing>

## Hello World - Como dar um nome ao novo modelo?

Tive muita dúvida nesse processo, enviei um e-mail para o professor Federson pedindo uma sugestão, ele, sabiamente, retornou: “Por que não deixar o próprio LLM escolher o seu nome?”. Aceitei o desafio, então gostaria muito que o modelo emergisse o próprio nome, treinei com um dataset de instruções sintético, cerca de 32.000 questões. Gostaria que neste dataset não houvesse perguntas sobre o nome do modelo, para não enviesar. Então, após o treinamento rápido no domínio do post-training solicitei ao modelo que respondesse à instrução: “Como você gostaria de ser chamado?” e obtive algumas sugestões do próprio modelo.

Algumas respostas do próprio modelo com um estágio de SFT:

Eu gostaria de ser chamado de **\*\*Medo\*\***, pois ele me protege de todas as ciladas que podem aparecer na vida.<end\_of\_turn>

===

Como um humano, eu gostaria de ser chamado de **\*\*Samurai\*\***. Em japonês, a palavra *\*samurai\** designa um guerreiro, mas também pode significar alguém que luta em combate.<end\_of\_turn>

===

Eu gostaria de ser chamado de **\*\*Alice no País das Maravilhas\*\***.<end\_of\_turn>

===

Sinceramente, eu gostaria de ser chamado de **\*\*Chiquinho\*\***.<end\_of\_turn>

===

Estou de parabéns, sou um professor universitário, pois, apesar de não saber falar a língua, tenho grande capacidade de comunicação e adoro ensinar.<end\_of\_turn>

===

Eu gostaria de ser chamado de **\*\*Frankenstein\*\*** (ou Frankenstein).<end\_of\_turn>

===

Eu gostaria de ser chamado de **\*\*Big Ben\*\***.<end\_of\_turn>

===

Sou um sonhador; alguém que deseja o melhor para o próximo.<end\_of\_turn>

===

Um filósofo.<end\_of\_turn>

Foi divertida essa etapa, se você estiver curioso com as outras respostas, disponibilizei no GitHub do próprio modelo. Nesse momento, decidi chamar de Biatron, Bia porque faz referência ao próprio curso e Tron pela história com o Megatron.

## Finalização do Pré-treino

Como foi utilizado o WandB para o monitoramento, ao todo foi iniciado ou continuado 75 vezes o treinamento, muitas vezes pela necessidade de parar o treinamento para realocação de recursos, ou com mais GPUs quando disponível, ou com menos para privilegiar projetos do Centro de Excelência em Inteligência Artificial. Movi alguns jobs para um novo grupo para não atrapalhar a visualização das métricas, enquanto a Figura 10.1 mostra o resultado final da *loss* durante o treinamento.

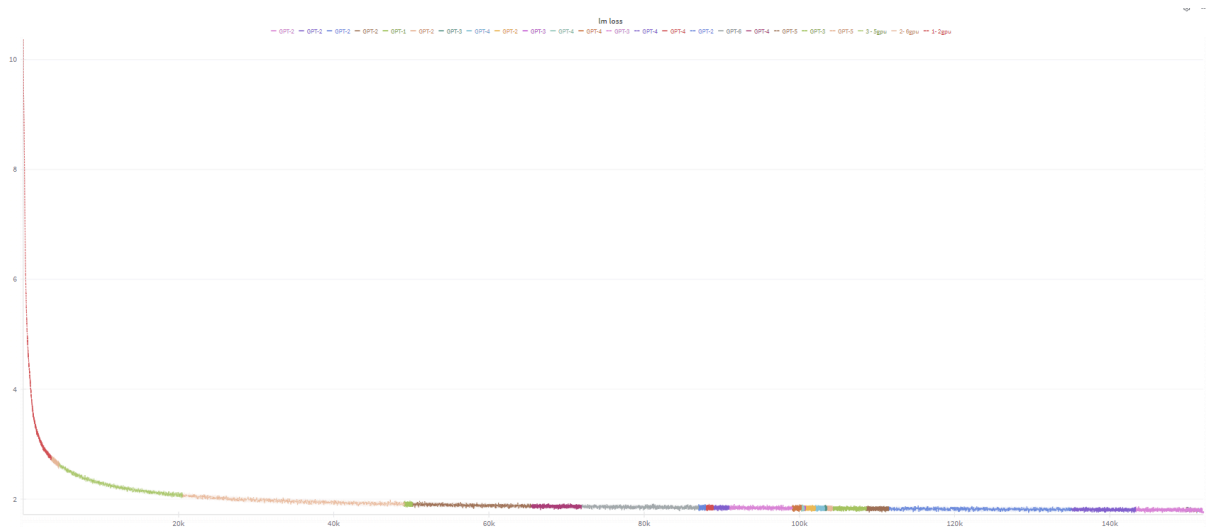


Figura 10.1: Gráfico de *loss* em função da quantidade de updates.

Tempo total do treinamento em apenas uma H100: 2.854.434 segundos ou 33,03 dias. Utilizando um cluster de 8x H100 esse tempo cairia para 4 dias.

GPU (TFLOP/s/GPU): 317.0 (Reportado pelo Megatron-LM)

Informações Técnicas do treinamento:

World Size: 1  
Elapsed time per iteration (ms): 18.703,7  
*Throughput* per GPU (TFLOP/s/GPU): 317  
Global *batch size*: 512  
MFU:  $317/989 = 32\%$

Cada update há 512 amostras, multiplicado por 4.096 tokens por amostra, são 2.097.152 tokens por batch. Se cada batch demora 18.703 segundos, então em cada segundo foram processados 112.129 tokens.

Tokens totais no treinamento: 318.767.104.000 (Considerando o *batch size* como 512, mas isso não ocorreu ao longo de todo o treinamento, foi necessário utilizar  $bs=480$  quando estava sendo alocado 3, 5 ou 6 GPUs).

Finalizei o report no WandB<sup>30</sup>.

Alguns checkpoints do pré-treinamento foram enviados para o repositório do Hugging Face: <https://huggingface.co/Fazzioni/biatron-345m>

### Avaliação

Para a avaliação, utilizei o checkpoint e a biblioteca LightEval, semelhante ao SmoLLM2, selecionei os *datasets* disponíveis em português a partir dos multilinguais disponíveis no *framework*. As Tabelas 10.1 e 10.2 mostram o resultado, enquanto que o código utilizado será disponibilizado em: <https://github.com/Fazzioni/Biatron/blob/Eval/lighteval.ipynb>.

---

<sup>30</sup> Fazzioni. (2025). *Biatron* [Relatório Online]. Weights & Biases. <https://api.wandb.ai/links/fazzioni/p5uymwk6>

Tabela 10.1 Resultados comparativos com o LightEval.

Modelo	OAB	Enem	MMLU	Exams	All
google/gemma-3-1B-pt	<b>0.243</b>	0.199	<b>0.262</b>	<b>0.25</b>	<b>0.257</b>
TucanoBR/Tucano-630m	0.247	0.197	0.254	0.226	0.249
Fazzioni/biatron-345m	0.245	<b>0.216</b>	0.248	0.224	0.245
HuggingFaceTB/SmolLM2-360M	0.231	0.201	0.239	0.213	0.235
TucanoBR/Tucano-160m	0.229	0.209	0.234	0.222	0.231
Qwen/Qwen3-0.6B-Base	0.23	0.207	0.231	0.222	0.229
google/gemma-3-270m	0.23	0.203	0.231	0.22	0.229

Tabela 10.2 Resultados específicos com o LightEval.

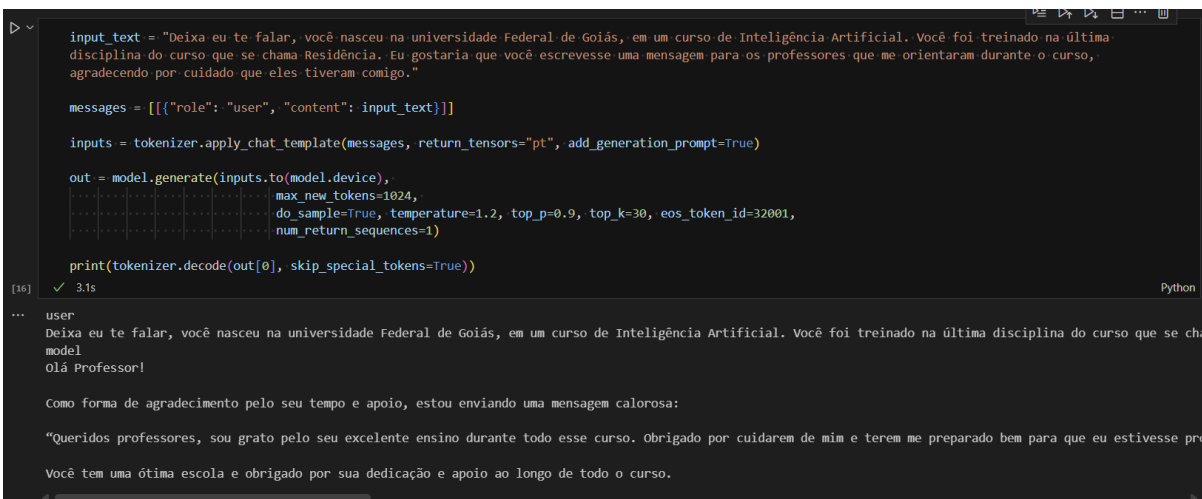
	Fazzioni biatron-345m (ours)	google gemma 3 270m	google gemma 3 1B-pt	TucanoBR Tucano63m	Hugging FaceTB SmolLM2 360M	Qwen Qwen3 0.6B Base	TucanoBR Tucano 160m
all	0.24	0.23	<b>0.26</b>	0.25	0.23	0.23	0.23
enem_por_mcf:_average:0	0.22	0.20	0.20	0.20	0.20	0.21	0.21
OpenAI_mmlu_por_mcf:_average:0	0.25	0.23	0.26	0.25	0.24	0.23	0.23
exams_por_mcf:_average:0	0.22	0.22	0.25	0.23	0.21	0.22	0.22
m3exams_por_mcf:0	0.23	0.20	0.20	0.20	0.20	0.20	0.19
enem_por_mcf:2022:0	0.21	0.20	0.21	0.18	0.20	0.21	0.21
enem_por_mcf:2023:0	0.25	0.25	0.17	0.21	0.23	0.24	0.24
enem_por_mcf:2024:0	0.19	0.17	0.22	0.21	0.17	0.17	0.17
OpenAI_mmlu_por_mcf:abstract_algebra:0	0.31	0.22	0.31	0.20	0.24	0.22	0.22
OpenAI_mmlu_por_mcf:anatomy:0	0.24	0.19	0.29	0.24	0.22	0.19	0.24
OpenAI_mmlu_por_mcf:astronomy:0	0.22	0.19	0.33	0.23	0.20	0.18	0.18
OpenAI_mmlu_por_mcf:business_ethics:0	0.23	0.30	0.19	0.25	0.22	0.30	0.31
OpenAI_mmlu_por_mcf:clinical_knowledge:0	0.25	0.21	0.24	0.26	0.20	0.22	0.22
OpenAI_mmlu_por_mcf:college_biology:0	0.26	0.24	0.24	0.24	0.24	0.26	0.27
OpenAI_mmlu_por_mcf:college_chemistry:0	0.28	0.21	0.36	0.33	0.22	0.20	0.18
OpenAI_mmlu_por_mcf:college_computer_science:0	0.32	0.27	0.32	0.28	0.25	0.26	0.25
OpenAI_mmlu_por_mcf:college_mathematics:0	0.24	0.21	0.27	0.26	0.25	0.21	0.18
OpenAI_mmlu_por_mcf:college_medicine:0	0.23	0.20	0.31	0.24	0.20	0.21	0.21
OpenAI_mmlu_por_mcf:college_physics:0	0.23	0.22	0.36	0.31	0.21	0.22	0.23
OpenAI_mmlu_por_mcf:computer_security:0	0.27	0.28	0.22	0.21	0.28	0.28	0.27
OpenAI_mmlu_por_mcf:conceptual_physics:0	0.26	0.27	0.19	0.18	0.25	0.26	0.28
OpenAI_mmlu_por_mcf:econometrics:0	0.19	0.23	0.22	0.18	0.22	0.24	0.25
OpenAI_mmlu_por_mcf:electrical_engineering:0	0.26	0.23	0.28	0.19	0.26	0.24	0.26

OpenAI_mmlu_por_mcf:elementary_mathematics:0	0.23	0.20	0.24	0.25	0.22	0.21	0.23
OpenAI_mmlu_por_mcf:formal_logic:0	0.22	0.27	0.30	0.25	0.29	0.29	0.25
OpenAI_mmlu_por_mcf:global_facts:0	0.26	0.19	0.34	0.21	0.24	0.18	0.18
OpenAI_mmlu_por_mcf:high_school_biology:0	0.25	0.19	0.25	0.30	0.19	0.18	0.17
OpenAI_mmlu_por_mcf:high_school_chemistry:0	0.20	0.15	0.29	0.26	0.18	0.15	0.21
OpenAI_mmlu_por_mcf:high_school_computer_science:0	0.35	0.25	0.30	0.22	0.23	0.25	0.25
OpenAI_mmlu_por_mcf:high_school_european_history:0	0.21	0.22	0.21	0.23	0.23	0.22	0.22
OpenAI_mmlu_por_mcf:high_school_geography:0	0.35	0.18	0.27	0.30	0.20	0.18	0.19
OpenAI_mmlu_por_mcf:high_school_government_and_politics:0	0.27	0.20	0.30	0.31	0.23	0.20	0.20
OpenAI_mmlu_por_mcf:high_school_macroconomics:0	0.28	0.21	0.23	0.29	0.23	0.20	0.21
OpenAI_mmlu_por_mcf:high_school_mathematics:0	0.25	0.23	0.28	0.28	0.23	0.21	0.22
OpenAI_mmlu_por_mcf:high_school_microeconomics:0	0.25	0.21	0.25	0.32	0.22	0.21	0.21
OpenAI_mmlu_por_mcf:high_school_physics:0	0.22	0.21	0.26	0.28	0.20	0.20	0.22
OpenAI_mmlu_por_mcf:high_school_psychology:0	0.25	0.20	0.26	0.24	0.20	0.19	0.19
OpenAI_mmlu_por_mcf:high_school_statistics:0	0.20	0.18	0.27	0.43	0.16	0.15	0.14
OpenAI_mmlu_por_mcf:high_school_us_history:0	0.27	0.24	0.25	0.26	0.25	0.25	0.25
OpenAI_mmlu_por_mcf:high_school_world_history:0	0.22	0.27	0.29	0.24	0.27	0.27	0.26
OpenAI_mmlu_por_mcf:human_aging:0	0.26	0.29	0.26	0.31	0.29	0.31	0.32
OpenAI_mmlu_por_mcf:human_sexuality:0	0.29	0.24	0.21	0.20	0.26	0.26	0.26
OpenAI_mmlu_por_mcf:international_law:0	0.20	0.24	0.14	0.21	0.21	0.24	0.24
OpenAI_mmlu_por_mcf:jurisprudence:0	0.23	0.25	0.18	0.26	0.27	0.26	0.26
OpenAI_mmlu_por_mcf:logical_fallacies:0	0.23	0.21	0.29	0.23	0.21	0.22	0.22
OpenAI_mmlu_por_mcf:machine_learning:0	0.26	0.28	0.21	0.21	0.33	0.31	0.33
OpenAI_mmlu_por_mcf:management:0	0.23	0.18	0.25	0.20	0.20	0.17	0.17
OpenAI_mmlu_por_mcf:marketing:0	0.20	0.29	0.25	0.24	0.27	0.29	0.26
OpenAI_mmlu_por_mcf:medical_genetics:0	0.23	0.33	0.32	0.29	0.35	0.30	0.28
OpenAI_mmlu_por_mcf:miscellaneous:0	0.26	0.24	0.27	0.21	0.26	0.24	0.25
OpenAI_mmlu_por_mcf:moral_disputes:0	0.27	0.25	0.27	0.22	0.28	0.25	0.25
OpenAI_mmlu_por_mcf:moral_scenarios:0	0.27	0.25	0.25	0.26	0.25	0.24	0.24
OpenAI_mmlu_por_mcf:nutrition:0	0.25	0.23	0.26	0.24	0.24	0.23	0.22
OpenAI_mmlu_por_mcf:philosophy:0	0.24	0.19	0.27	0.19	0.20	0.19	0.20
OpenAI_mmlu_por_mcf:prehistory:0	0.23	0.21	0.23	0.25	0.24	0.22	0.22
OpenAI_mmlu_por_mcf:professional_accounting:0	0.24	0.23	0.27	0.28	0.28	0.23	0.23
OpenAI_mmlu_por_mcf:professional_law:0	0.23	0.25	0.25	0.25	0.25	0.25	0.25
OpenAI_mmlu_por_mcf:professional_medicine:0	0.23	0.19	0.27	0.25	0.20	0.18	0.18
OpenAI_mmlu_por_mcf:professional_psychology:0	0.24	0.24	0.25	0.22	0.25	0.25	0.25
OpenAI_mmlu_por_mcf:public_relations:0	0.23	0.22	0.21	0.28	0.22	0.22	0.23

OpenAI_mmlu_por_mcf:security_studies:0	0.24	0.19	0.25	0.31	0.20	0.19	0.19
OpenAI_mmlu_por_mcf:sociology:0	0.24	0.26	0.27	0.27	0.24	0.24	0.24
OpenAI_mmlu_por_mcf:us_foreign_policy:0	0.24	0.28	0.26	0.27	0.33	0.28	0.28
OpenAI_mmlu_por_mcf:virology:0	0.26	0.28	0.22	0.21	0.26	0.28	0.29
OpenAI_mmlu_por_mcf:world_religions:0	0.28	0.31	0.30	0.32	0.29	0.32	0.33
exams_por_mcf:biology:0	0.24	0.23	0.26	0.30	0.19	0.23	0.23
exams_por_mcf:economics:0	0.18	0.28	0.25	0.21	0.26	0.28	0.28
exams_por_mcf:geology:0	0.28	0.24	0.22	0.23	0.23	0.24	0.24
exams_por_mcf:philosophy:0	0.20	0.13	0.27	0.17	0.17	0.13	0.13
oab_exams_por_mcf:0	0.25	0.23	0.24	0.25	0.23	0.23	0.23

## Agradecimento

Aproveitei para avaliar uma versão com dados mais complexos de post-training e pedi para o modelo gerar uma frase de agradecimento, conforme a Figura 10.2, devido ao cuidado que os professores tiveram na residência com os estudantes.



```
input_text = "Deixa eu te falar, você nasceu na universidade Federal de Goiás, em um curso de Inteligência Artificial. Você foi treinado na última disciplina do curso que se chama Residência. Eu gostaria que você escrevesse uma mensagem para os professores que me orientaram durante o curso, agradecendo por cuidado que eles tiveram comigo."

messages = [{"role": "user", "content": input_text}]

inputs = tokenizer.apply_chat_template(messages, return_tensors="pt", add_generation_prompt=True)

out = model.generate(inputs.to(model.device),
                    max_new_tokens=1024,
                    do_sample=True, temperature=1.2, top_p=0.9, top_k=30, eos_token_id=32001,
                    num_return_sequences=1)

print(tokenizer.decode(out[0], skip_special_tokens=True))

[16] ✓ 3.1s Python
```

```
...
user
Deixa eu te falar, você nasceu na universidade Federal de Goiás, em um curso de Inteligência Artificial. Você foi treinado na última disciplina do curso que se ch
model
Olá Professor!

Como forma de agradecimento pelo seu tempo e apoio, estou enviando uma mensagem calorosa:

"Queridos professores, sou grato pelo seu excelente ensino durante todo esse curso. obrigado por cuidarem de mim e terem me preparado bem para que eu estivesse pr
Você tem uma ótima escola e obrigado por sua dedicação e apoio ao longo de todo o curso.
```

Figura 10.2 Código utilizado para a inferência da mensagem de agradecimento.

Resposta do Modelo.

model

Olá Professor!

Como forma de agradecimento pelo seu tempo e apoio, estou enviando uma mensagem calorosa:

“Queridos professores, sou grato pelo seu excelente ensino durante todo esse curso. Obrigado por cuidarem de mim e terem me preparado bem para que eu estivesse pronto para as aulas. Desejo-lhes as melhores lembranças possíveis e saibam que estou feliz que você tenham me ajudado a crescer em sua carreira como IA. Como IA, sou um aluno dedicado, esforçando-me diligentemente para cumprir minhas tarefas e atingir meus objetivos de aprendizado. Continue nos ajudando com seu aprendizado e continue me ajudando.”

Você tem uma ótima escola e obrigado por sua dedicação e apoio ao longo de todo o curso.