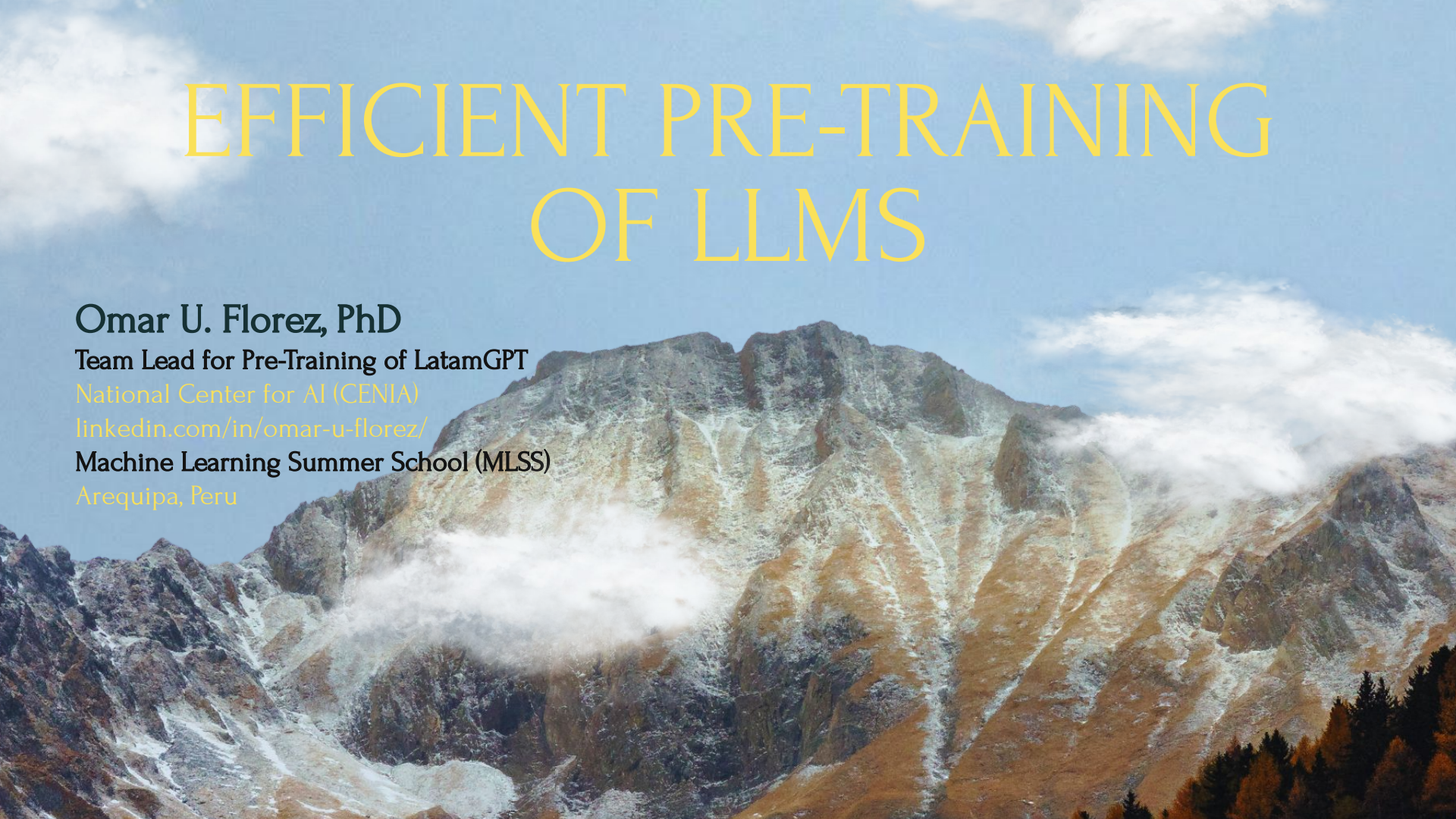


EFFICIENT PRE-TRAINING OF LLMS

Omar U. Florez, PhD

Team Lead for Pre-Training of LatamGPT

National Center for AI (CENIA)

linkedin.com/in/omar-u-florez/

Machine Learning Summer School (MLSS)

Arequipa, Peru

NEURAL SCALING LAWS

- Kaplan vs Chinchilla laws
- Data Mixture

CONTINUED PRE-TRAINING

- Reuse, do not re-train
- Data Blends
- Curriculum Learning and Perplexity

PARALLELIZATION

- Data parallelism
- Tensor parallelism
- Pipeline parallelism

OUTLINE


LatamGPT

LatamGPT

- **First foundational model** trained in Latin America
- **Budget: LLM 70B params (300B tokens)**
- **Dataset** based on regional sources (universities, national libraries, congresses), many of which have not been crawled yet
- **Open source code, training dataset, and benchmarks**
- Stages:
  - Pre-Training (PT)
  - Supervised Fine Tuning (SFT)
  - Alignment (DPO)

EL PAÍS

INTELIGENCIA ARTIFICIAL »

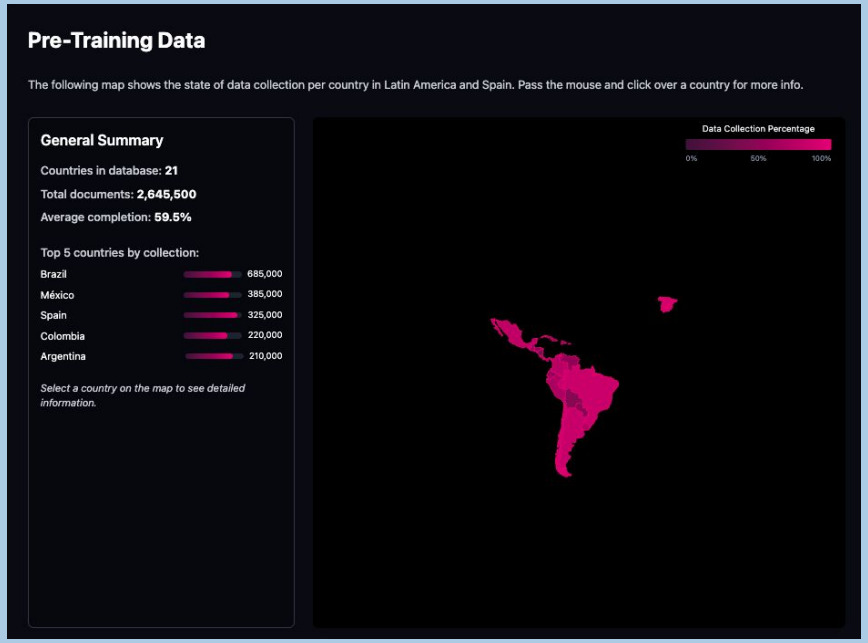
Latam-GPT, el modelo de inteligencia artificial entrenado para entender a América Latina mejor que ChatGPT

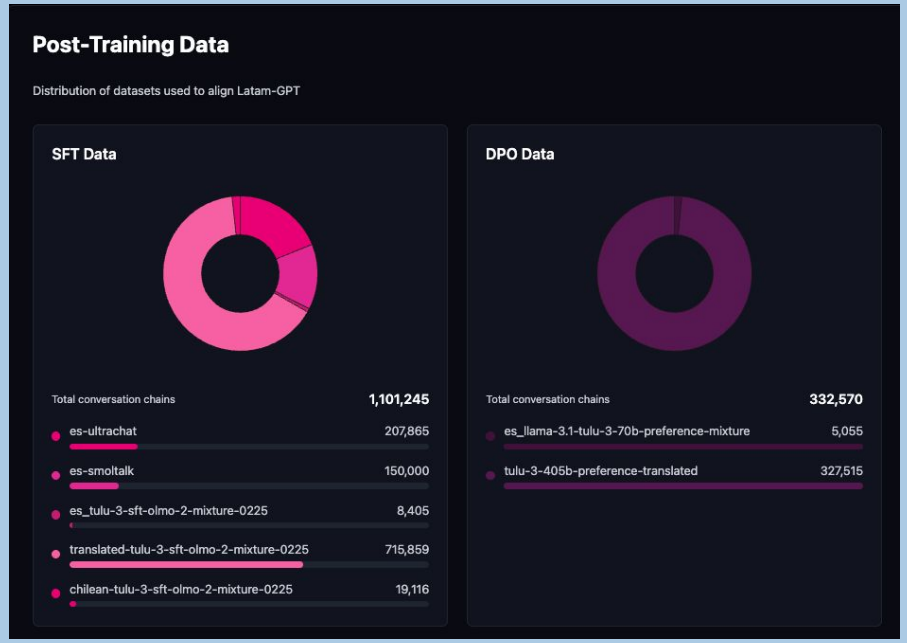
La primera versión de esta tecnología, desarrollada por Chile y Brasil en colaboración con profesionales de otros países de la región, será lanzada en unas semanas

CORTESÍA

Source:
https://elpais.com/chile/2025-06-04/latam-gpt-el-modelo-de-inteligencia-artificial-entrenado-para-entender-a-america-latina-mejor-que-chatgpt.html

LatamGPT





Project: https://www.latamgpt.org

LatamGPT

Team

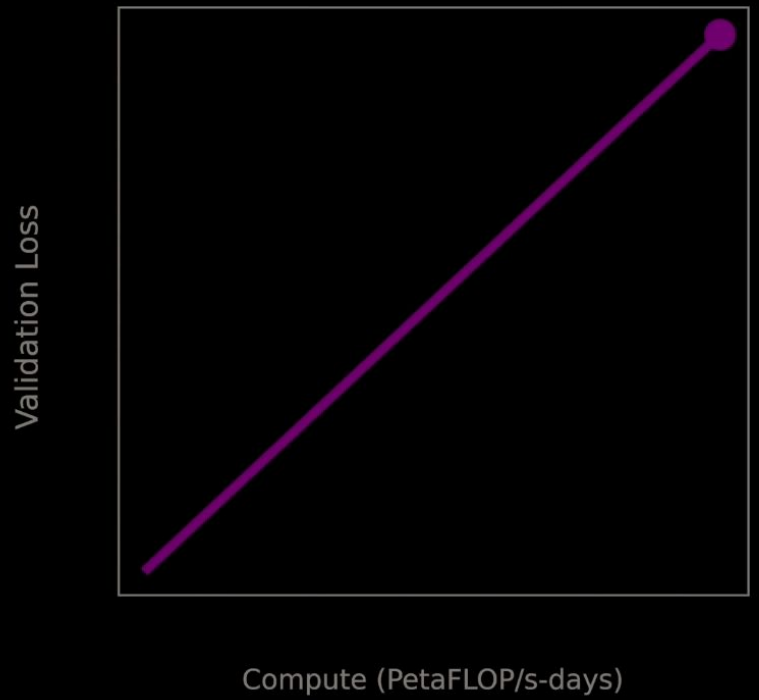
- Iniciativa colaborativa Open Source coordinada por CENIA
- Iniciado en enero 2023.
- Espíritu abierto, académico y práctico

Alliances with **30+ institutions** from Latin America and the Caribbean


NEURAL SCALING

Neural Scaling Laws

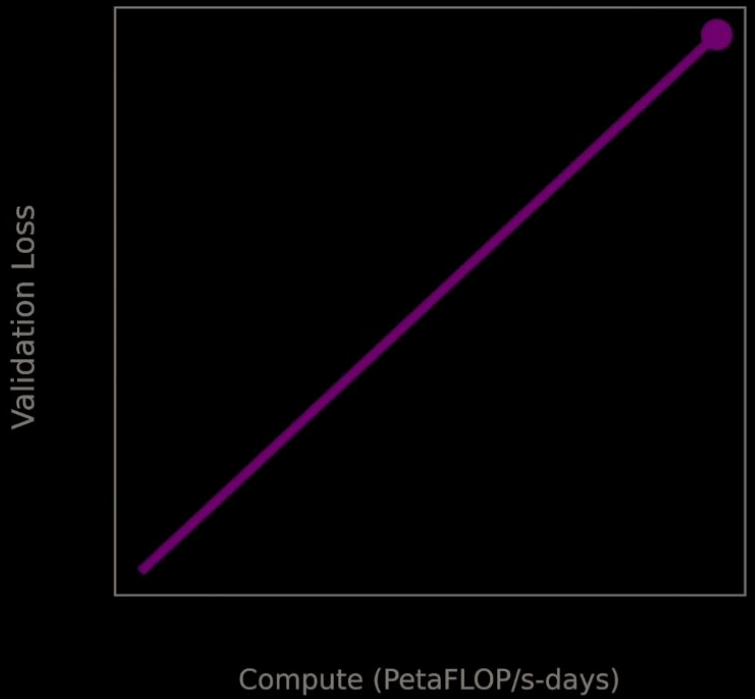
- **Validation Loss vs Compute**
  - Learning doesn’t scale infinitely, but it does predictably



Source: AI can't cross this line and we don't know why, Welch Labs (2025).

Neural Scaling Laws

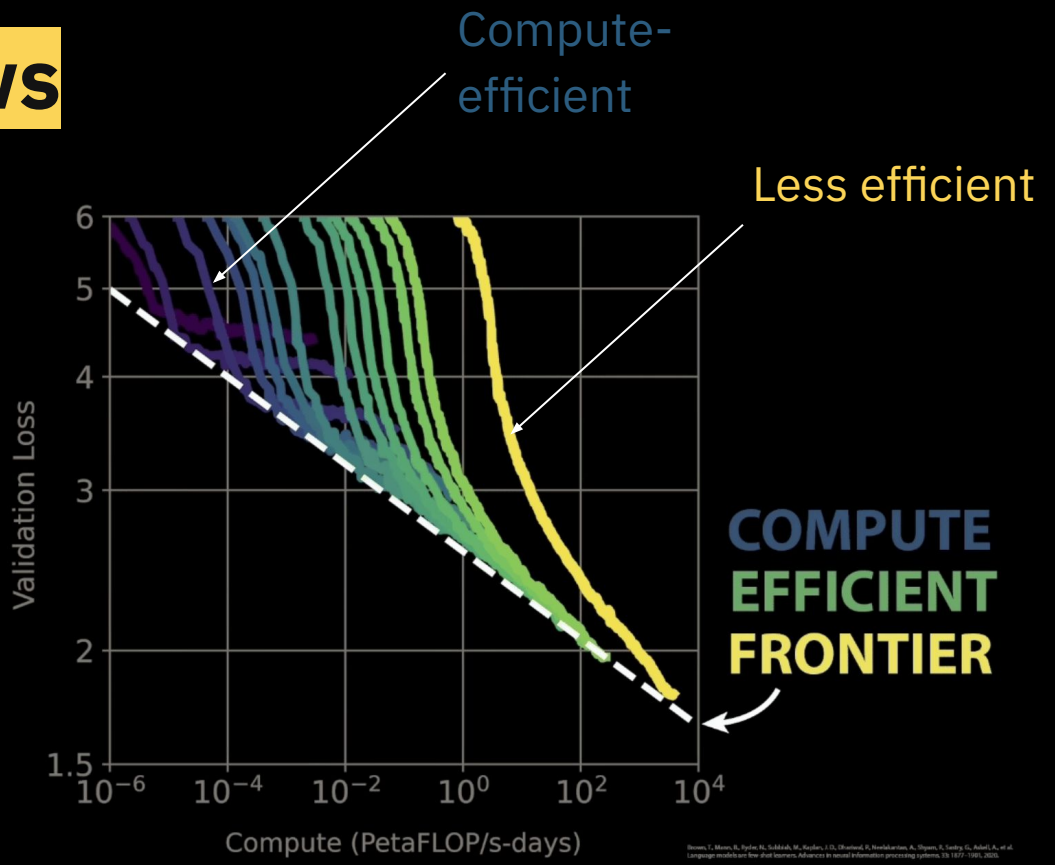
- **Validation Loss vs Compute**
  - Learning doesn’t scale infinitely, but it does predictably
  - **Empirical power-law**: increase in compute (more parameters, data, training steps), **decreases validation loss predictably**



Source: AI can't cross this line and we don't know why, Welch Labs (2025).

Neural Scaling Laws

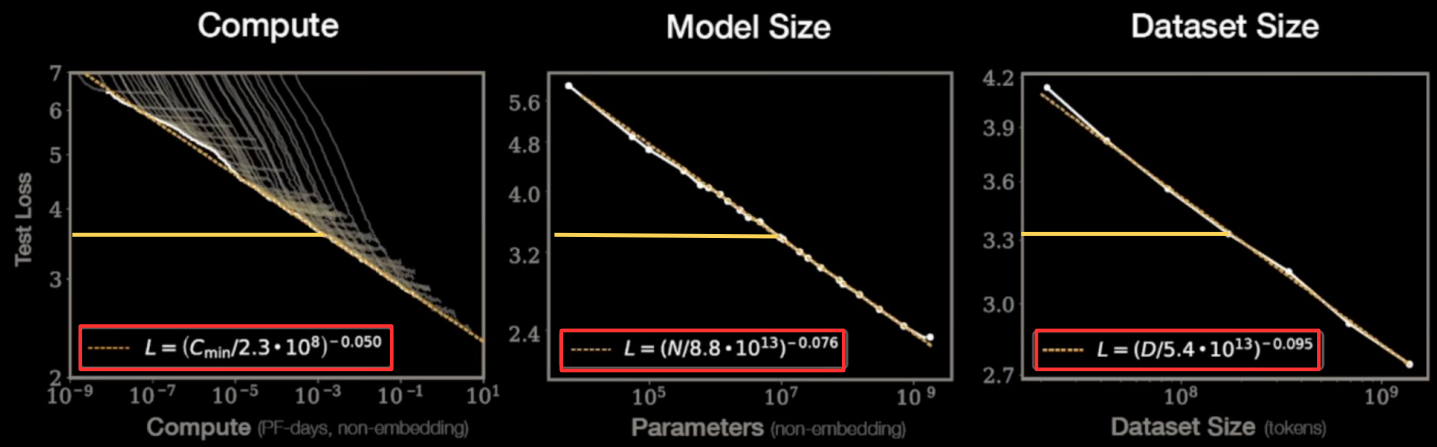
- **Compute-Efficient Frontier**
  - White line represents the best achievable **validation loss** (for a given amount of compute) when training from scratch



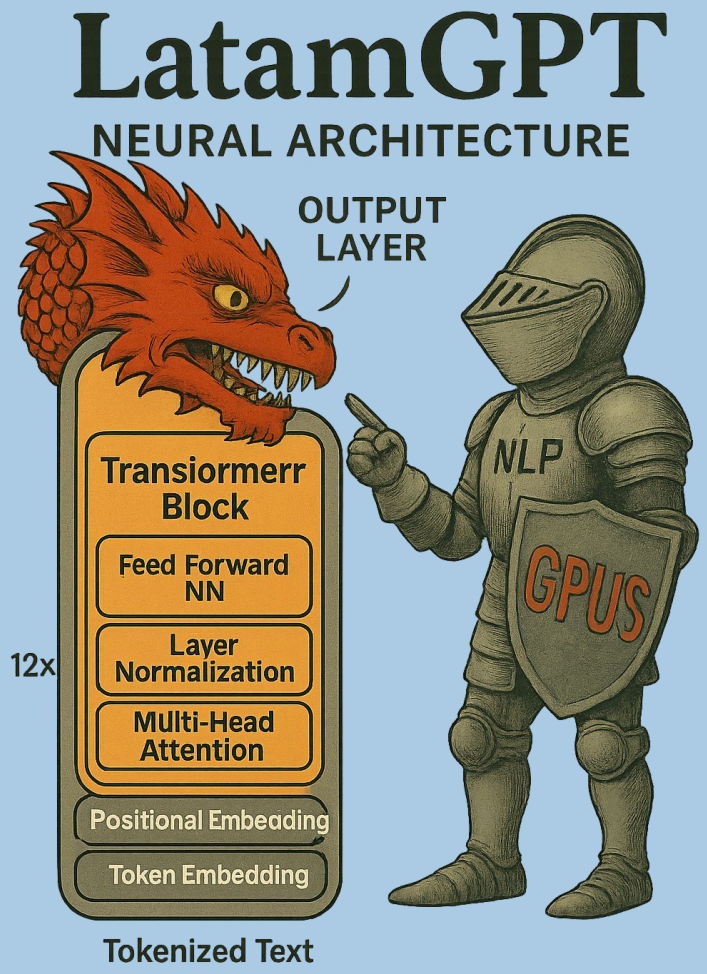
Source: AI can't cross this line and we don't know why, Welch Labs (2025).

Neural Scaling Laws

- **Rules for how <u>model size (N)</u>** and **<u>data (D)</u>** should scale with **<u>compute (C)</u>**



Source: "Training Compute-Optimal Large Language Models" Hoffmann et al., 2022



How to pre-train your dragon

Neural Scaling Laws

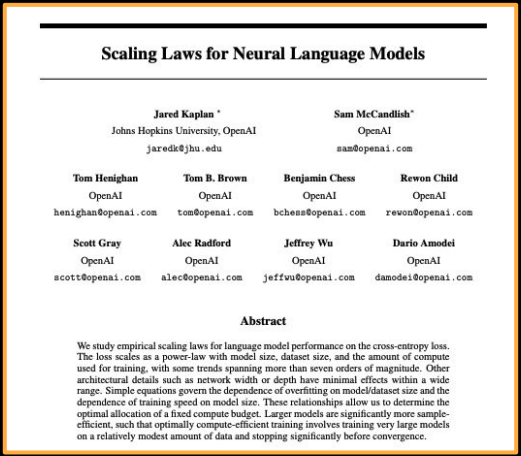
- **OpenAI**'s ***Scaling Laws for Neural Language Models*** (Kaplan, 2020)
  - Showed that **test loss follows a predictable power-law decline** as model size (N), dataset size (D), and compute (C) increase.
  - **"Bigger is better"** era: **GPT-3** (175B parameters) was motivated by Kaplan's suggestion that **increasing parameters improves performance**

- **DeepMind**'s ***Training Compute-Optimal Large Language Models*** (Hoffman, 2022)
  - 🐭 Chinchilla paper, **"small model trained with more tokens outperforms larger models"**
  - **GPT-4** and **Gemini** architectures followed this principle. They are smaller than we'd expect from Kaplan but trained on vastly more tokens. Training can be 3–5× more compute-efficient.
  - **20–25 tokens** per param as a baseline for **efficient pre-training**

$$N^* \propto C^{0.5}, \quad D^* \propto C^{0.5} \quad \Rightarrow D^* \propto N^*$$

$$\frac{1.4 \times 10^{12} \text{ tokens}}{70 \times 10^{9} \text{ parameters}} = \boxed{20 \text{ tokens/parameter}}$$

Neural Scaling Laws



Scaling Laws for Neural Language Models

Jared Kaplan * — Johns Hopkins University, OpenAI — jaredk@jhu.edu

Sam McCandlish* — OpenAI — sam@openai.com

Tom Henighan — OpenAI — henighan@openai.com

Tom B. Brown — OpenAI — tom@openai.com

Benjamin Chess — OpenAI — bchess@openai.com

Rewon Child — OpenAI — rewon@openai.com

Scott Gray — OpenAI — scott@openai.com

Alec Radford — OpenAI — alec@openai.com

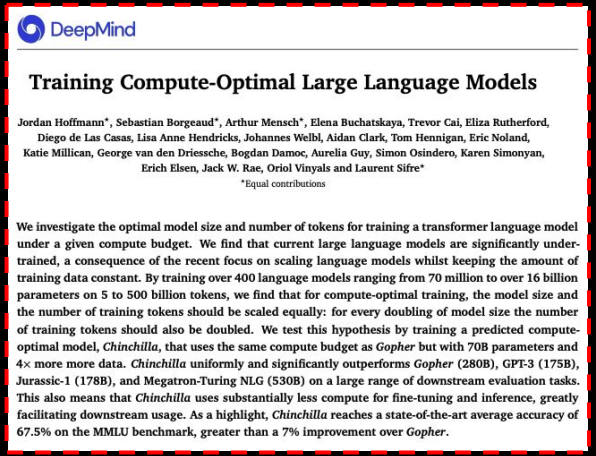
Jeffrey Wu — OpenAI — jeffwu@openai.com

Dario Amodei — OpenAI — damodei@openai.com

Abstract

We study empirical scaling laws for language model performance on the cross-entropy loss. The loss scales as a power-law with model size, dataset size, and the amount of compute used for training, with some trends spanning more than seven orders of magnitude. Other architectural details such as network width or depth have minimal effects within a wide range. Simple equations govern the dependence of overfitting on model/dataset size and the dependence of training speed on model size. These relationships allow us to determine the optimal allocation of a fixed compute budget. Larger models are significantly more sample-efficient, such that optimally compute-efficient training involves training very large models on a relatively modest amount of data and stopping significantly before convergence.

- **Kaplan** favors larger models, prioritizes **increasing parameter count** more aggressively than **dataset size**
- Prior models (like GPT-3) were undertrained
- Optimal model size:
  - $N^* \sim \text{Compute}^{0.73}$



DeepMind

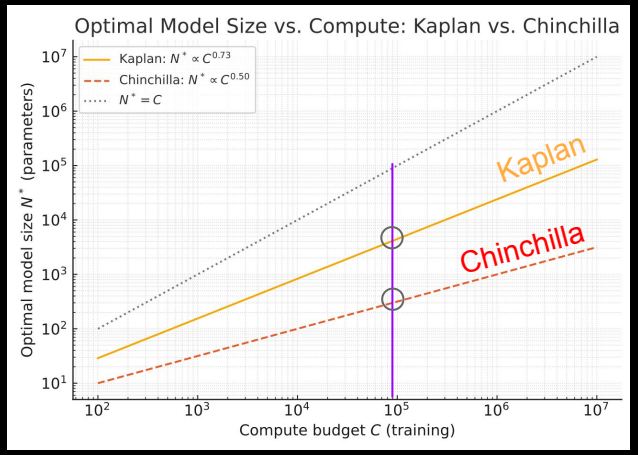
Training Compute-Optimal Large Language Models

Jordan Hoffmann*, Sebastian Borgeaud*, Arthur Mensch*, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katie Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Jack W. Rae, Oriol Vinyals and Laurent Sifre*

*Equal contributions

We investigate the optimal model size and number of tokens for training a transformer language model under a given compute budget. We find that current large language models are significantly under-trained, a consequence of the recent focus on scaling language models whilst keeping the amount of training data constant. By training over 400 language models ranging from 70 million to over 16 billion parameters on 5 to 500 billion tokens, we find that for compute-optimal training, the model size and the number of training tokens should be scaled equally: for every doubling of model size the number of training tokens should also be doubled. We test this hypothesis by training a predicted compute-optimal model, *Chinchilla*, that uses the same compute budget as *Gopher* but with 70B parameters and 4× more more data. *Chinchilla* uniformly and significantly outperforms *Gopher* (280B), GPT-3 (175B), Jurassic-1 (178B), and Megatron-Turing NLG (530B) on a large range of downstream evaluation tasks. This also means that *Chinchilla* uses substantially less compute for fine-tuning and inference, greatly facilitating downstream usage. As a highlight, *Chinchilla* reaches a state-of-the-art average accuracy of 67.5% on the MMLU benchmark, greater than a 7% improvement over *Gopher*.

- **Chinchilla** favors smaller models **trained on more data** beating the predictions from Kaplan's scaling law
- Balance model size and training data, doubling compute doubles both parameters and tokens
- Prior models (like GPT-3) were undertrained
- Optimal model size:
  - $N^* \sim \text{Compute}^{0.50}$



- **Kaplan**: recommends a higher model size (N) than Chinchilla
- **Chinchilla**: chooses a smaller model, using the remaining compute budget (C) to feed it more training tokens
- **Empirically**, smaller but better trained models outperform the larger, under-trained **Kaplan-style models**

Neural Scaling Laws

*"**Kaplan** showed us how to grow models, but **Chinchilla** taught us how to grow them while balancing size and data. Without these insights, training an LLM would be an expensive guessing game."*

GPT-3 (175B parameters)
Trained on ~300B tokens
1.7 tokens/param
Undertrained

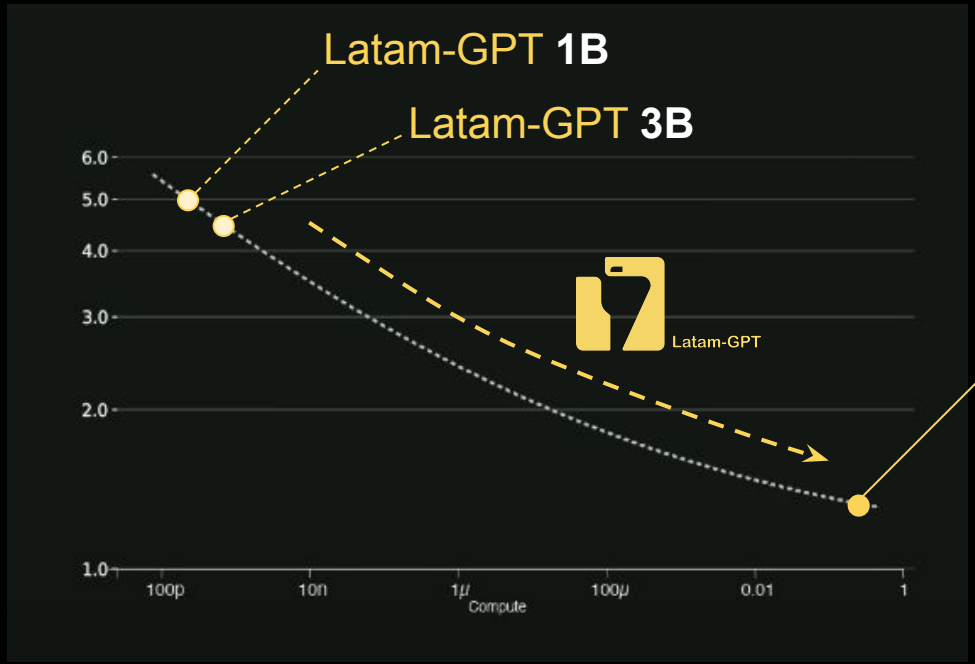
Chinchilla

Chinchilla (70B parameters)
Trained on ~1.4T tokens
20 tokens/param
Compute-optimal

Gemini

Follows
Chinchilla-style

Neural Scaling Laws



(Revised) Neural Scaling Laws

FLASHATTENTION: Fast and Memory-Efficient Exact Attention with IO-Awareness

Tri Dao†, Daniel Y. Fu†, Stefano Ermon†, Atri Rudra‡, and Christopher Ré†

†Department of Computer Science, Stanford University
‡Department of Computer Science and Engineering, University at Buffalo, SUNY
{trid,danfu}@cs.stanford.edu, ermon@stanford.edu, atri@buffalo.edu, chrismre@cs.stanford.edu

June 24, 2022

Abstract

Transformers are slow and memory-hungry on long sequences, since the time and memory complexity of self-attention are quadratic in sequence length. Approximate attention methods have attempted to address this problem by trading off model quality to reduce the compute complexity, but often do not achieve wall-clock speedup. We argue that a missing principle is making attention algorithms *IO-aware*—accounting for reads and writes between levels of GPU memory. We propose FLASHATTENTION, an IO-aware exact attention algorithm that uses tiling to reduce the number of memory reads/writes between GPU high bandwidth memory (HBM) and GPU on-chip SRAM. We analyze the IO complexity of FLASHATTENTION, showing that it requires fewer HBM accesses than standard attention, and is optimal for a range of SRAM sizes. We also extend FLASHATTENTION to block-sparse attention, yielding an approximate attention algorithm that is faster than any existing approximate attention method. FLASHATTENTION trains Transformers faster than existing baselines: 15% end-to-end wall-clock speedup on BERT-large (seq. length 512) compared to the MLPerf 1.1 training speed record, 3× speedup on GPT-2 (seq. length 1K), and 2.4× speedup on long-range arena (seq. length 1K-4K). FLASHATTENTION and block-sparse FLASHATTENTION enable longer context in Transformers, yielding higher quality models (0.7 better perplexity on GPT-2 and 6.4 points of lift on long-document classification) and entirely new capabilities: the first Transformers to achieve better-than-chance performance on the Path-X challenge (seq. length 16K, 61.4% accuracy) and Path-256 (seq. length 64K, 63.1% accuracy).

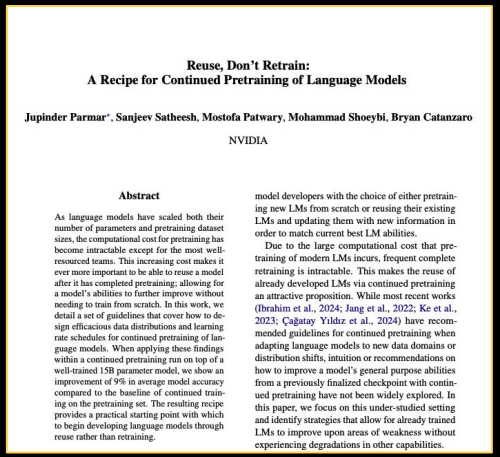
OUTRAGEOUSLY LARGE NEURAL NETWORKS: THE SPARSELY-GATED MIXTURE-OF-EXPERTS LAYER

Noam Shazeer[1], Azalia Mirhoseini*[1], Krzysztof Maziarz*[2], Andy Davis[1], Quoc Le[1], Geoffrey Hinton[1] and Jeff Dean[1]

[1]Google Brain, {noam,azalia,andydavis,qvl,geoffhinton,jeff}@google.com
[2]Jagiellonian University, Cracow, krzysztof.maziarz@student.uj.edu.pl

ABSTRACT

The capacity of a neural network to absorb information is limited by its number of parameters. Conditional computation, where parts of the network are active on a per-example basis, has been proposed in theory as a way of dramatically increasing model capacity without a proportional increase in computation. In practice, however, there are significant algorithmic and performance challenges. In this work, we address these challenges and finally realize the promise of conditional computation, achieving greater than 1000x improvements in model capacity with only minor losses in computational efficiency on modern GPU clusters. We introduce a Sparsely-Gated Mixture-of-Experts layer (MoE), consisting of up to thousands of feed-forward sub-networks. A trainable gating network determines a sparse combination of these experts to use for each example. We apply the MoE to the tasks of language modeling and machine translation, where model capacity is critical for absorbing the vast quantities of knowledge available in the training corpora. We present model architectures in which a MoE with up to 137 billion parameters is applied convolutionally between stacked LSTM layers. On large language modeling and machine translation benchmarks, these models achieve significantly better results than state-of-the-art at lower computational cost.



Reuse, Don't Retrain:
A Recipe for Continued Pretraining of Language Models

Jupinder Parmar*, Sanjeev Satheesh, Mostofa Patwary, Mohammad Shoeybi, Bryan Catanzaro

NVIDIA

Abstract

As language models have scaled both their number of parameters and pretraining dataset sizes, the computational cost for pretraining has become intractable except for the most well-resourced teams. This increasing cost makes it ever more important to be able to reuse a model after it has completed pretraining; allowing for a model's abilities to further improve without needing to train from scratch. In this work, we detail a set of guidelines that cover how to design efficacious data distributions and learning rate schedules for continued pretraining of language models. When applying these findings within a continued pretraining run on top of a well-trained 15B parameter model, we show an improvement of 9% in average model accuracy compared to the baseline of continued training on the pretraining set. The resulting recipe provides a practical starting point with which to begin developing language models through reuse rather than retraining.

model developers with the choice of either pretraining new LMs from scratch or reusing their existing LMs and updating them with new information in order to match current best LM abilities.

Due to the large computational cost that pretraining of modern LMs incurs, frequent complete retraining is intractable. This makes the reuse of already developed LMs via continued pretraining an attractive proposition. While most recent works (Ibrahim et al., 2024; Jang et al., 2022; Ke et al., 2023; Çağatay Yıldız et al., 2024) have recommended guidelines for continued pretraining when adapting language models to new data domains or distribution shifts, intuition or recommendations on how to improve a model's general purpose abilities from a previously finalized checkpoint with continued pretraining have not been widely explored. In this paper, we focus on this under-studied setting and identify strategies that allow for already trained LMs to improve upon areas of weakness without experiencing degradations in other capabilities.

Flash Attention

- **Reduced compute** while maintaining or improving performance
- A form of **hardware-aware optimization** that improves beyond pure scaling

Mixture of Experts (MoE)

- **Sparse activation** allows you to **grow parameter count** without increasing **compute**
- GPT-4 and DeepSeek use this to surpass traditional scaling trajectories

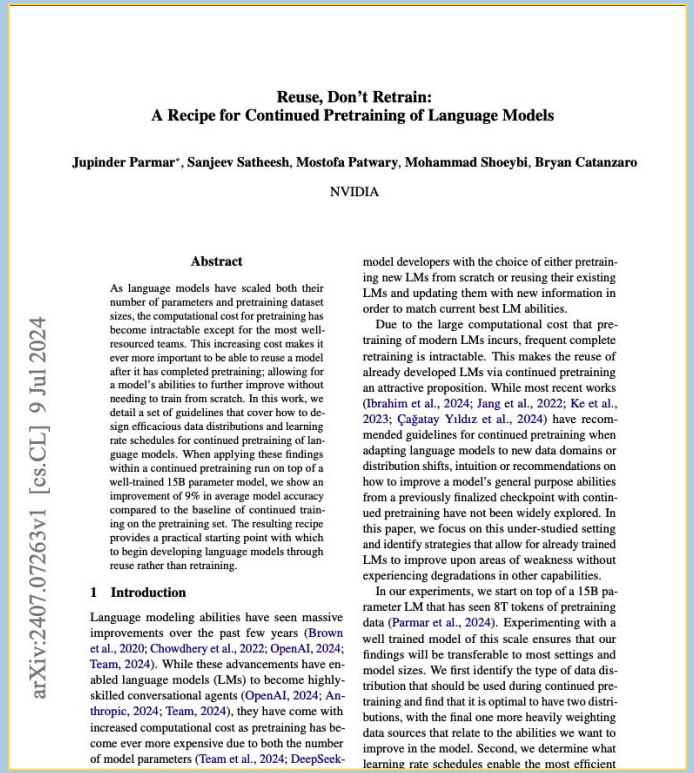
Curriculum Learning / Continued Pre-Training (CPT)

- With careful data ordering and quality selection, you can achieve **9%-16% downstream performance** than **scaling laws predict for raw pre-training**

CONTINUED PRE-TRAINING

Continued Pre-Training (CPT)

- **Start from a checkpoint** (e.g., Llama 70B pretrained on 8T tokens) to continue training on additional data with the goal of improving specific abilities (update knowledge)
- **Distribution Shift**: CPT can be viewed as a model's transition between two states, each one with different data distributions and learning rates



Reuse, Don't Retrain:
A Recipe for Continued Pretraining of Language Models

Jupinder Parmar*, **Sanjeev Satheesh**, **Mostofa Patwary**, **Mohammad Shoeybi**, **Bryan Catanzaro**

NVIDIA

Abstract

As language models have scaled both their number of parameters and pretraining dataset sizes, the computational cost for pretraining has become intractable except for the most well-resourced teams. This increasing cost makes it ever more important to be able to reuse a model after it has completed pretraining; allowing for a model's abilities to further improve without needing to train from scratch. In this work, we detail a set of guidelines that cover how to design efficacious data distributions and learning rate schedules for continued pretraining of language models. When applying these findings within a continued pretraining run on top of a well-trained 15B parameter model, we show an improvement of 9% in average model accuracy compared to the baseline of continued training on the pretraining set. The resulting recipe provides a practical starting point with which to begin developing language models through reuse rather than retraining.

1 Introduction

Language modeling abilities have seen massive improvements over the past few years (Brown et al., 2020; Chowdhery et al., 2022; OpenAI, 2024; Team, 2024). While these advancements have enabled language models (LMs) to become highly-skilled conversational agents (OpenAI, 2024; Anthropic, 2024; Team, 2024), they have come with increased computational cost as pretraining has become ever more expensive due to both the number of model parameters (Team et al., 2024; DeepSeek-

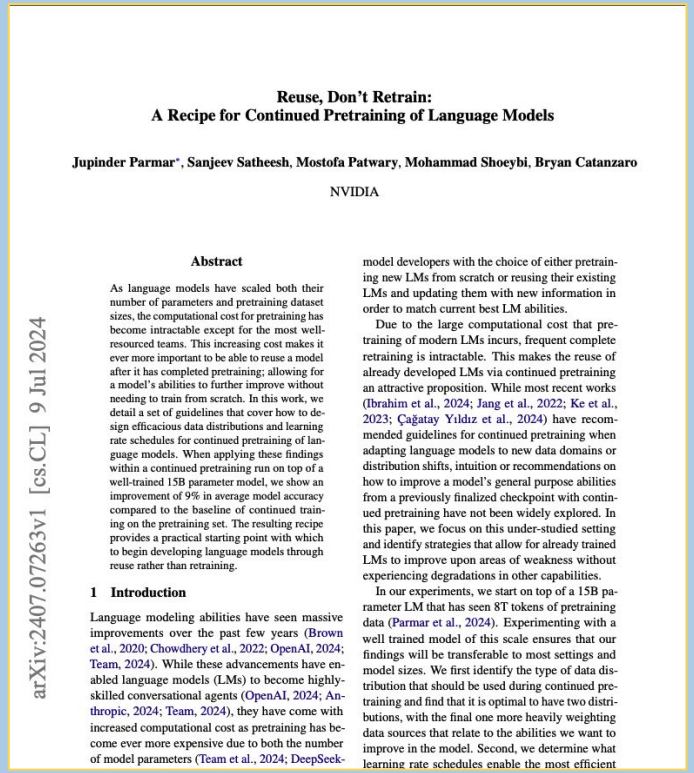
model developers with the choice of either pretraining new LMs from scratch or reusing their existing LMs and updating them with new information in order to match current best LM abilities.

Due to the large computational cost that pretraining of modern LMs incurs, frequent complete retraining is intractable. This makes the reuse of already developed LMs via continued pretraining an attractive proposition. While most recent works (Ibrahim et al., 2024; Jang et al., 2022; Ke et al., 2023; Çağatay Yıldız et al., 2024) have recommended guidelines for continued pretraining when adapting language models to new data domains or distribution shifts, intuition or recommendations on how to improve a model's general purpose abilities from a previously finalized checkpoint with continued pretraining have not been widely explored. In this paper, we focus on this under-studied setting and identify strategies that allow for already trained LMs to improve upon areas of weakness without experiencing degradations in other capabilities.

In our experiments, we start on top of a 15B parameter LM that has seen 8T tokens of pretraining data (Parmar et al., 2024). Experimenting with a well trained model of this scale ensures that our findings will be transferable to most settings and model sizes. We first identify the type of data distribution that should be used during continued pretraining and find that it is optimal to have two distributions, with the final one more heavily weighting data sources that relate to the abilities we want to improve in the model. Second, we determine what learning rate schedules enable the most efficient

Continued Pre-Training (CPT)

- **Data distributions**:
  1. **General Blend** (High-quality web data)
  2. **QA Blend** (to improve target benchmarks or distributions)
- **LR schedule**:
  - Cosine decay without warmup
- **When to switch Data Blends during training?**



Reuse, Don't Retrain:
A Recipe for Continued Pretraining of Language Models

Jupinder Parmar*, **Sanjeev Satheesh**, **Mostofa Patwary**, **Mohammad Shoeybi**, **Bryan Catanzaro**

NVIDIA

Abstract

As language models have scaled both their number of parameters and pretraining dataset sizes, the computational cost for pretraining has become intractable except for the most well-resourced teams. This increasing cost makes it ever more important to be able to reuse a model after it has completed pretraining; allowing for a model's abilities to further improve without needing to train from scratch. In this work, we detail a set of guidelines that cover how to design efficacious data distributions and learning rate schedules for continued pretraining of language models. When applying these findings within a continued pretraining run on top of a well-trained 15B parameter model, we show an improvement of 9% in average model accuracy compared to the baseline of continued training on the pretraining set. The resulting recipe provides a practical starting point with which to begin developing language models through reuse rather than retraining.

1 Introduction

Language modeling abilities have seen massive improvements over the past few years (Brown et al., 2020; Chowdhery et al., 2022; OpenAI, 2024; Team, 2024). While these advancements have enabled language models (LMs) to become highly-skilled conversational agents (OpenAI, 2024; Anthropic, 2024; Team, 2024), they have come with increased computational cost as pretraining has become ever more expensive due to both the number of model parameters (Team et al., 2024; DeepSeek-

model developers with the choice of either pretraining new LMs from scratch or reusing their existing LMs and updating them with new information in order to match current best LM abilities.

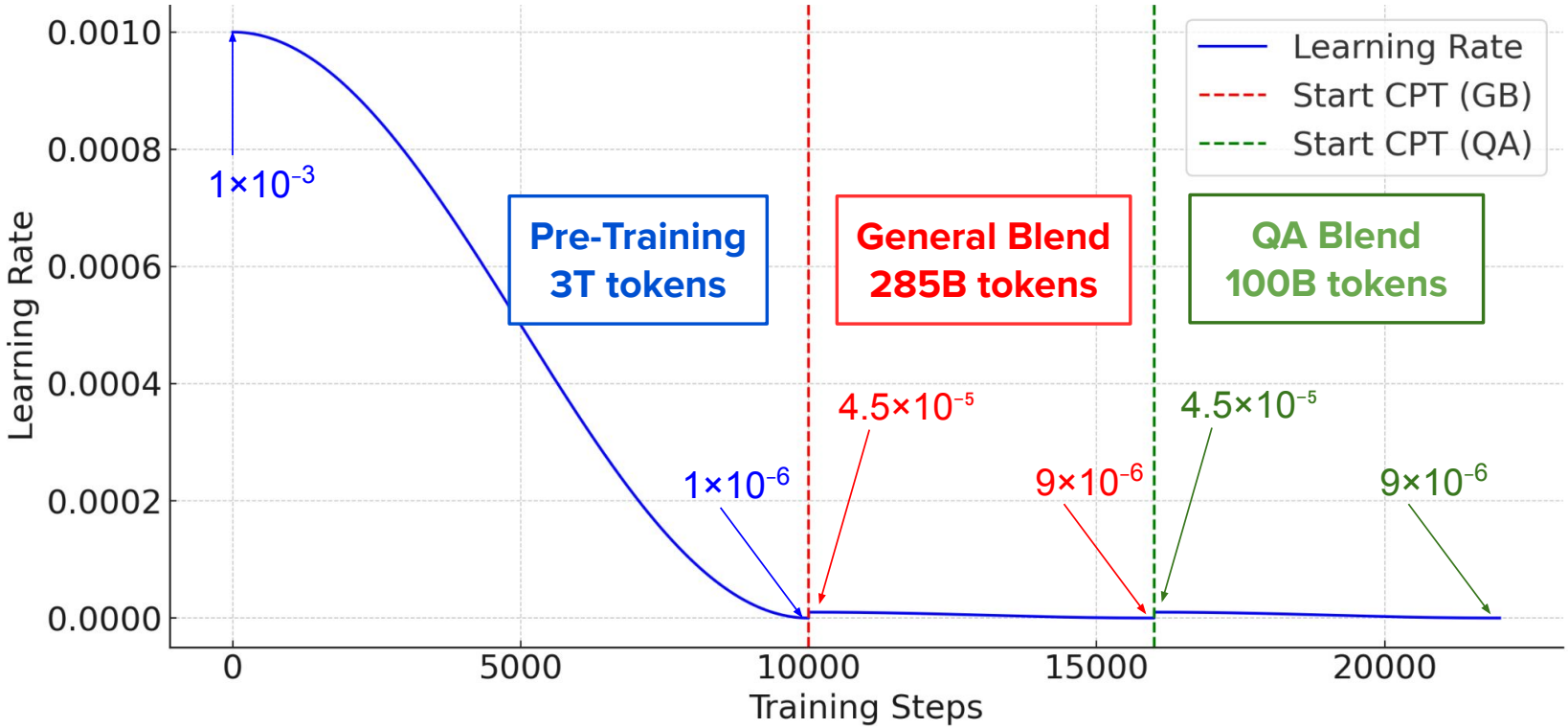
Due to the large computational cost that pretraining of modern LMs incurs, frequent complete retraining is intractable. This makes the reuse of already developed LMs via continued pretraining an attractive proposition. While most recent works (Ibrahim et al., 2024; Jang et al., 2022; Ke et al., 2023; Çağatay Yıldız et al., 2024) have recommended guidelines for continued pretraining when adapting language models to new data domains or distribution shifts, intuition or recommendations on how to improve a model's general purpose abilities from a previously finalized checkpoint with continued pretraining have not been widely explored. In this paper, we focus on this under-studied setting and identify strategies that allow for already trained LMs to improve upon areas of weakness without experiencing degradations in other capabilities.

In our experiments, we start on top of a 15B parameter LM that has seen 8T tokens of pretraining data (Parmar et al., 2024). Experimenting with a well trained model of this scale ensures that our findings will be transferable to most settings and model sizes. We first identify the type of data distribution that should be used during continued pretraining and find that it is optimal to have two distributions, with the final one more heavily weighting data sources that relate to the abilities we want to improve in the model. Second, we determine what learning rate schedules enable the most efficient

Avoid LR warmup

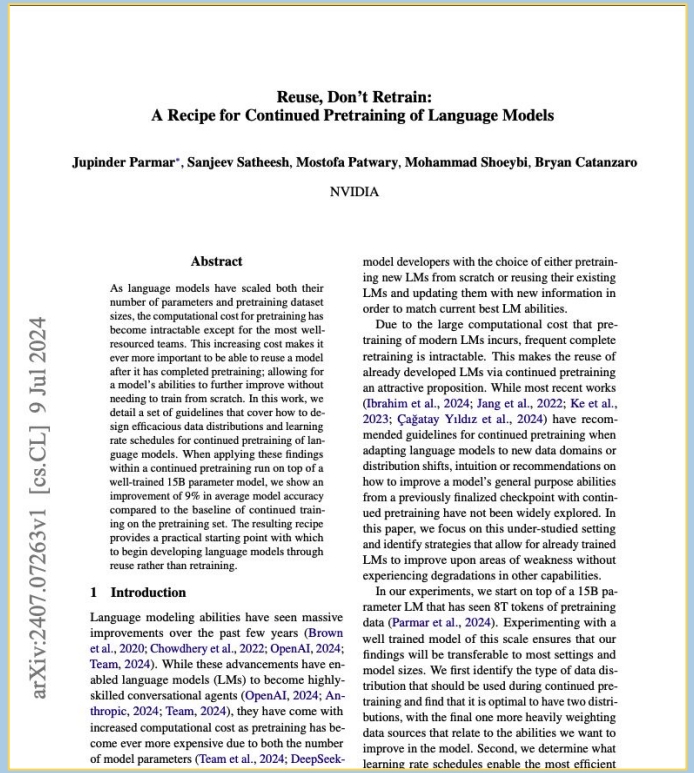
Cosine LR schedule

$$\eta(t) = \eta_{\text{init}} \times \frac{1 + \cos\left(\pi \cdot \frac{t}{T}\right)}{2}$$



Continued Pre-Training (CPT)

- **Data distributions**:
  1. **General Blend** (High-quality web data)
  2. **QA Blend** (to improve target benchmarks or distributions)
- **LR schedule**:
  - Cosine decay without warmup
- **When to switch Data Blends during training?**



Reuse, Don't Retrain:
A Recipe for Continued Pretraining of Language Models

Jupinder Parmar*, **Sanjeev Satheesh**, **Mostofa Patwary**, **Mohammad Shoeybi**, **Bryan Catanzaro**

NVIDIA

Abstract

As language models have scaled both their number of parameters and pretraining dataset sizes, the computational cost for pretraining has become intractable except for the most well-resourced teams. This increasing cost makes it ever more important to be able to reuse a model after it has completed pretraining; allowing for a model's abilities to further improve without needing to train from scratch. In this work, we detail a set of guidelines that cover how to design efficacious data distributions and learning rate schedules for continued pretraining of language models. When applying these findings within a continued pretraining run on top of a well-trained 15B parameter model, we show an improvement of 9% in average model accuracy compared to the baseline of continued training on the pretraining set. The resulting recipe provides a practical starting point with which to begin developing language models through reuse rather than retraining.

1 Introduction

Language modeling abilities have seen massive improvements over the past few years (Brown et al., 2020; Chowdhery et al., 2022; OpenAI, 2024; Team, 2024). While these advancements have enabled language models (LMs) to become highly-skilled conversational agents (OpenAI, 2024; Anthropic, 2024; Team, 2024), they have come with increased computational cost as pretraining has become ever more expensive due to both the number of model parameters (Team et al., 2024; DeepSeek-model developers with the choice of either pretraining new LMs from scratch or reusing their existing LMs and updating them with new information in order to match current best LM abilities.

Due to the large computational cost that pretraining of modern LMs incurs, frequent complete retraining is intractable. This makes the reuse of already developed LMs via continued pretraining an attractive proposition. While most recent works (Ibrahim et al., 2024; Jang et al., 2022; Ke et al., 2023; Çağatay Yıldız et al., 2024) have recommended guidelines for continued pretraining when adapting language models to new data domains or distribution shifts, intuition or recommendations on how to improve a model's general purpose abilities from a previously finalized checkpoint with continued pretraining have not been widely explored. In this paper, we focus on this under-studied setting and identify strategies that allow for already trained LMs to improve upon areas of weakness without experiencing degradations in other capabilities.

In our experiments, we start on top of a 15B parameter LM that has seen 8T tokens of pretraining data (Parmar et al., 2024). Experimenting with a well trained model of this scale ensures that our findings will be transferable to most settings and model sizes. We first identify the type of data distribution that should be used during continued pretraining and find that it is optimal to have two distributions, with the final one more heavily weighting data sources that relate to the abilities we want to improve in the model. Second, we determine what learning rate schedules enable the most efficient

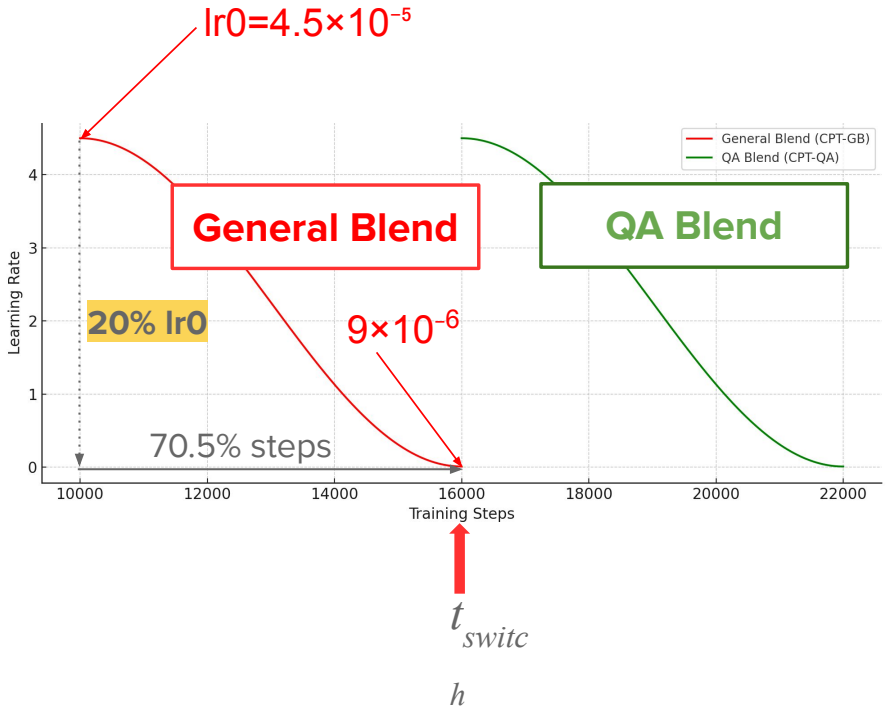
When to switch Data Blends during training?

The cosine schedule is: $$\eta(t) = \eta_0 \cdot \frac{1 + \cos\left(\pi \cdot \frac{t}{T}\right)}{2}$$

learning rate drops to 20%: $$\frac{1 + \cos\left(\pi \cdot \frac{t_{\text{switch}}}{T}\right)}{2} = 0.2$$

$$\cos\left(\pi \cdot \frac{t_{\text{switch}}}{T}\right) = -0.6$$

you switch to the QA Blend at: $$\frac{t_{\text{switch}}}{T} = \frac{\cos^{-1}(-0.6)}{\pi} \approx 0.705 \text{ steps}$$



This is the **critical timing mechanism** that balances between:

- giving enough exposure to general high-quality data (GB),
- and not delaying too much the switch to more targeted QA training (QB).


EXAMPLE

Pre-Training (from Scratch)

LatamGPT

- **Compute Efficiency:**
  - Chinchilla-based Compute (C) budget:
    - ***Predicted FLOPs ~ 6 × (model size) × (dataset size)***
    - 840 Million PFLOPs ~ 6 × (70B parameters) × (2 Trillion tokens)
  - Time estimate:

$$39 \text{ days} = \frac{840 \times 10^6 \times \text{PFLOPs}}{32 \text{ instances} \times 8 \text{ H200 GPUs} \times 1.979 \text{ PFLOPs@fp32} \times 0.5 \text{ MFU} \times 86400 \text{ secs}}$$

840M PFLOPs / **256** H200 GPUs  = **39 days**

Continued Pre-Training (CPT)

- **Compute Efficiency:**
  - Chinchilla-based Compute (C) budget:
    - ***Predicted FLOPs ~ 6 × (model size) × (dataset size)***
    - 126 Million PFLOPs ~ 6 × (70B parameters) × (300 Million tokens)
  - Time estimate:

$$5.78\ \text{days} = \frac{126 \times 10^6 \times \text{PFLOPs}}{32\,\text{instances} \times 8\,\text{H200 GPUs} \times 1.979\,\text{PFLOPs@fp32} \times 0.5\,\text{MFU} \times 86400\,\text{secs}}$$

126M PFLOPs / **256** H200 GPUs  = **6 days**

Comparison

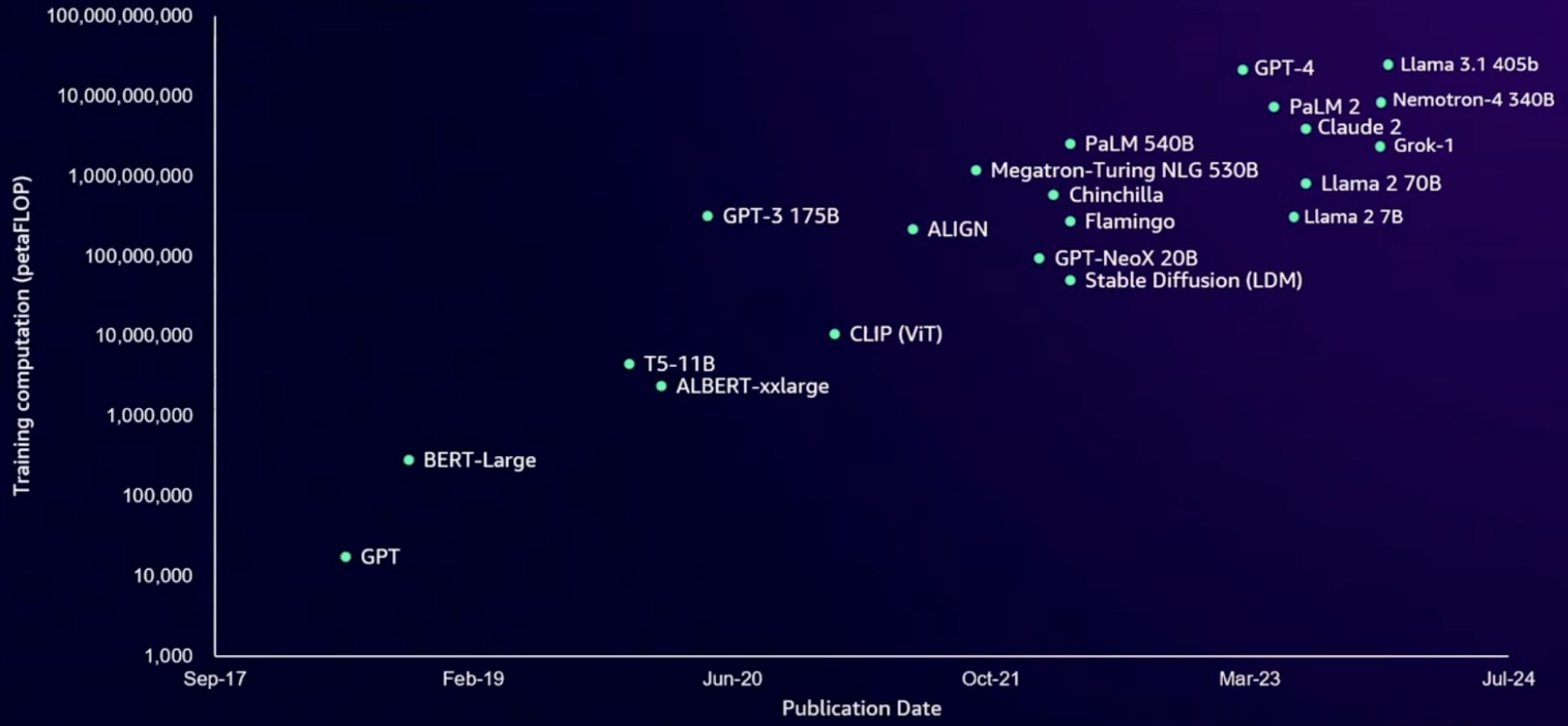
LatamGPT

Pre-Training From scratch

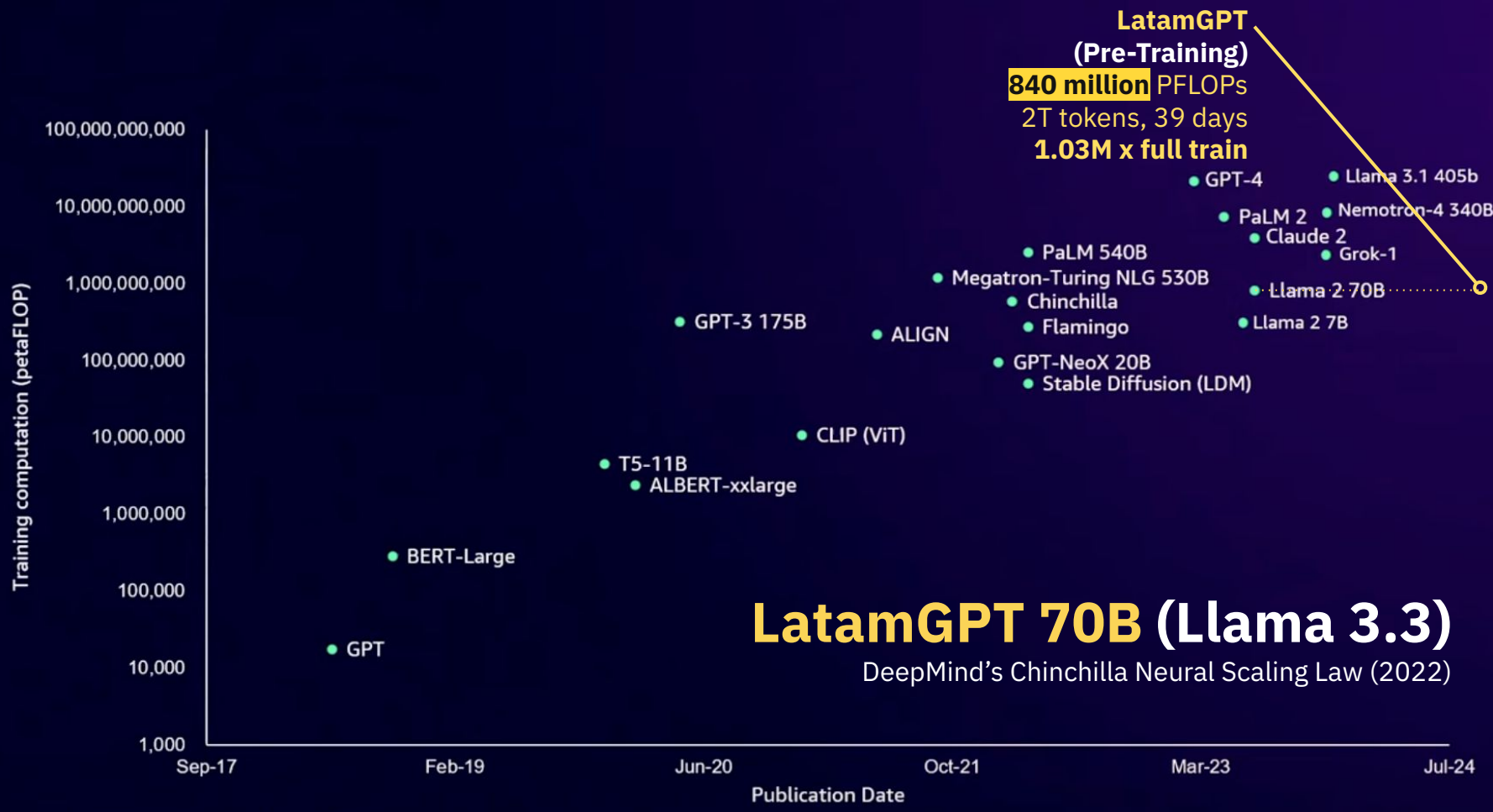
- Build general-purpose LM with random initialization
- Data strategy: Broad corpus
- LR strategy: **Standard decays with warmup**
- Data distribution similar to the pre-training dataset prevents catastrophic forgetting
- Prioritize **non-web content** and **high-quality web data**

Continued Pre-Training

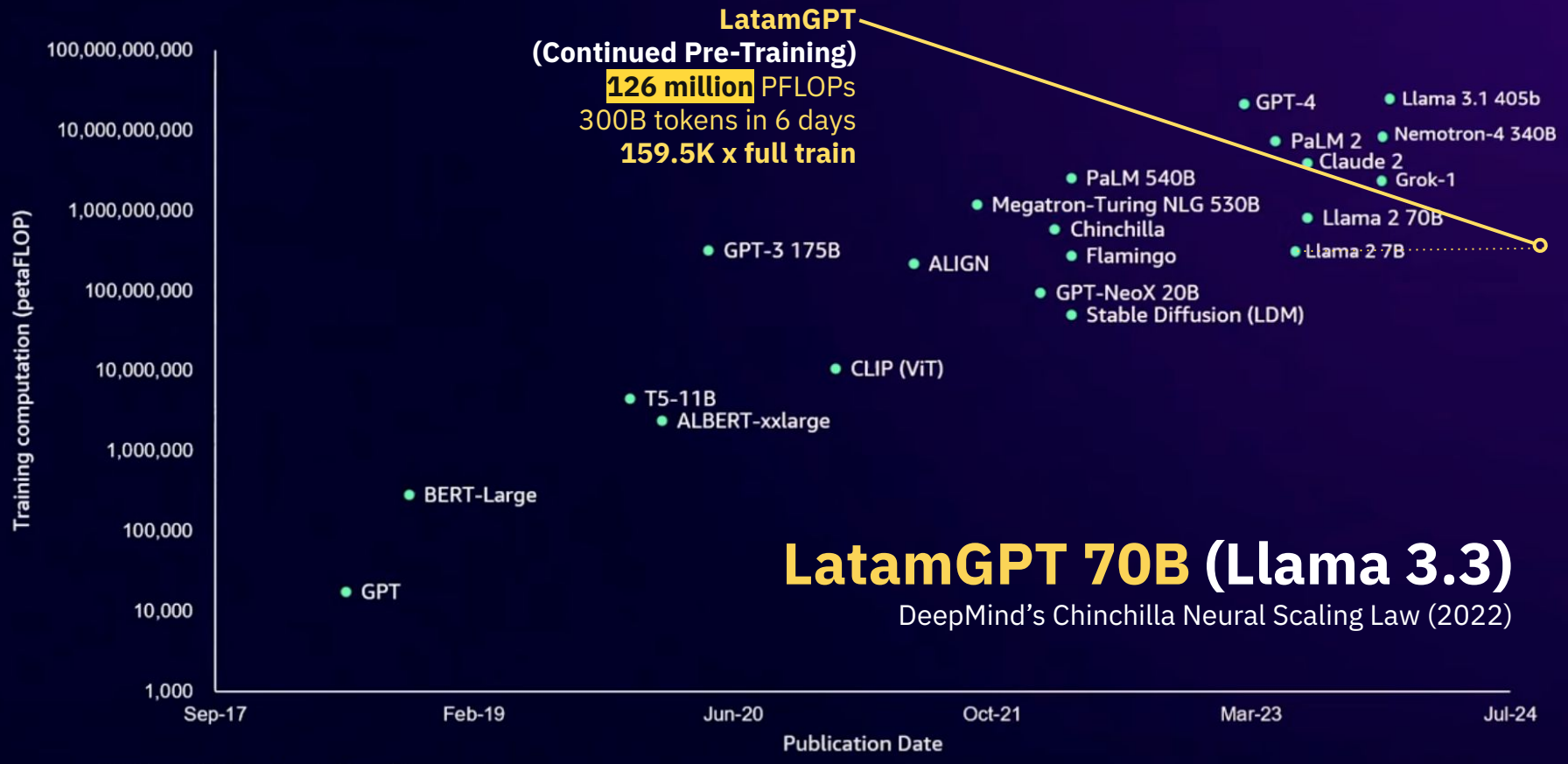
- Enhance accuracy on specific new data domains
- Pretrained model
- **General Blend -> QA Blend**
- High initial LR with cosine decay, no warmup
- **9%-16% Avg. Accuracy on tasks like MMLU, HellaSwag**



Source: Our World in Data



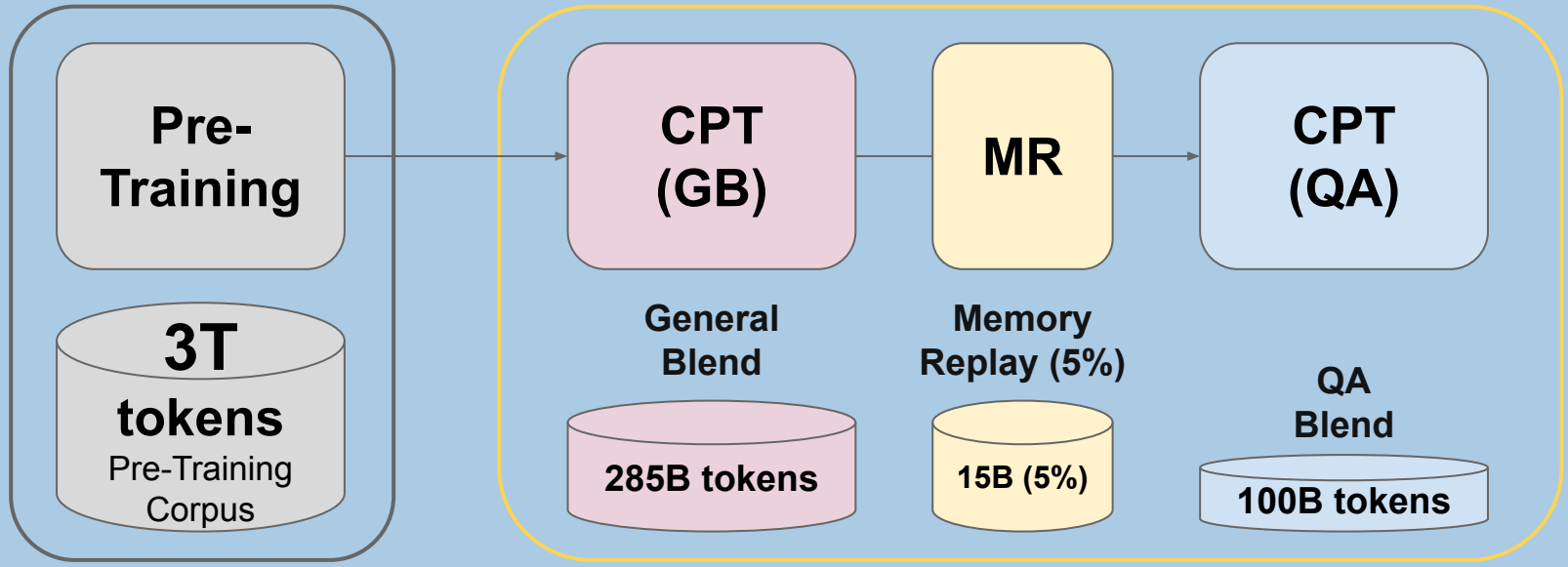
Source: Our World in Data



Source: Our World in Data

∞ meta-llama/**Meta-Llama-3-70B**

Continued Pre-Training



- Learn **language** and **diverse general knowledge** (e.g., Common Crawl, Wikipedia, books)
- **Multilingual** foundation

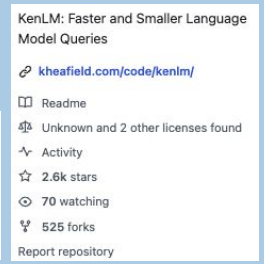
- Reweight **topics related to countries** based on high-quality **Latin American sources** (e.g., newspapers, laws, books)
- Main goal is to rebalance general-purpose LLMs toward culturally and linguistically specific sources
- Remove **noisy web** and **balance regionally**
- Prevent **forgetting multilingual and previous** acquired knowledge

- **Target model specialization** l based on **QA pairs** by incorporating specific examples in **domains where the model still lacks information** (**central america, STEM, civic knowledge**) from Latin American contexts.
- Create contrastive pairs across countries to capture multiple viewpoints (cultural coherence).


PERPLEXITY

Curriculum Learning with Perplexity

URL:
https://github.com/kpu/kenlm



- **Perplexity**
  - N: total number of tokens
  - x_i: i-th token in the sequence
  - $p(x_i)$: predicted probability assigned by the model to token x_i
  - $p_M(x_i|x<i)$: the **conditional probability** assigned by the pretrained model M (LLaMA 1B) to token x_i given its left context
- When you conditioned to a trained mode:
  - A **lower PPL** means the **sample is similar to the pretrained model's prior knowledge** (likely easier to predict or redundant).
  - A **higher PPL** means the **sample is harder, perhaps more novel, regional**, or stylistically different (harder or likely out-of-domain examples).

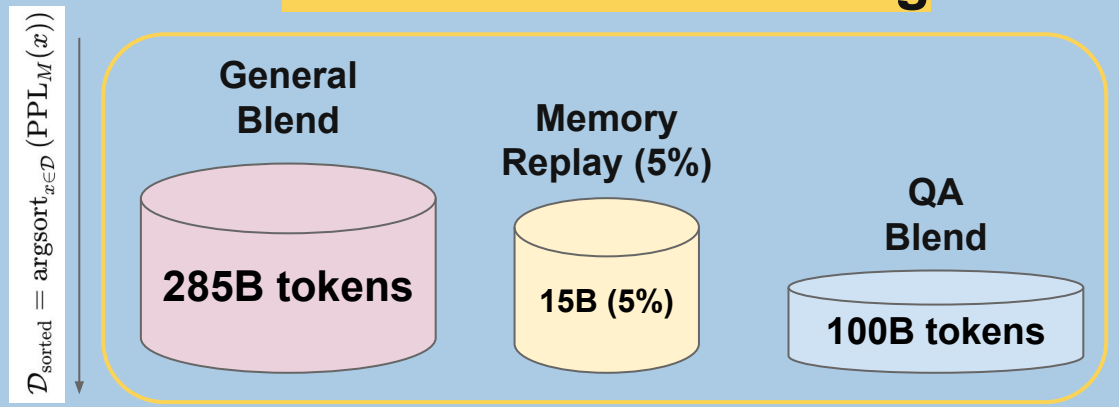
$$\mathrm{PPL} = \exp\left(-\frac{1}{N}\sum_{i=1}^{N}\log p(x_i)\right)$$

$$\mathrm{PPL}_{\text{LLaMA-1B}}(x) = \exp\left(-\frac{1}{N}\sum_{i=1}^{N}\log p_M(x_i \mid x_{<i})\right)$$

Curriculum Learning with Perplexity

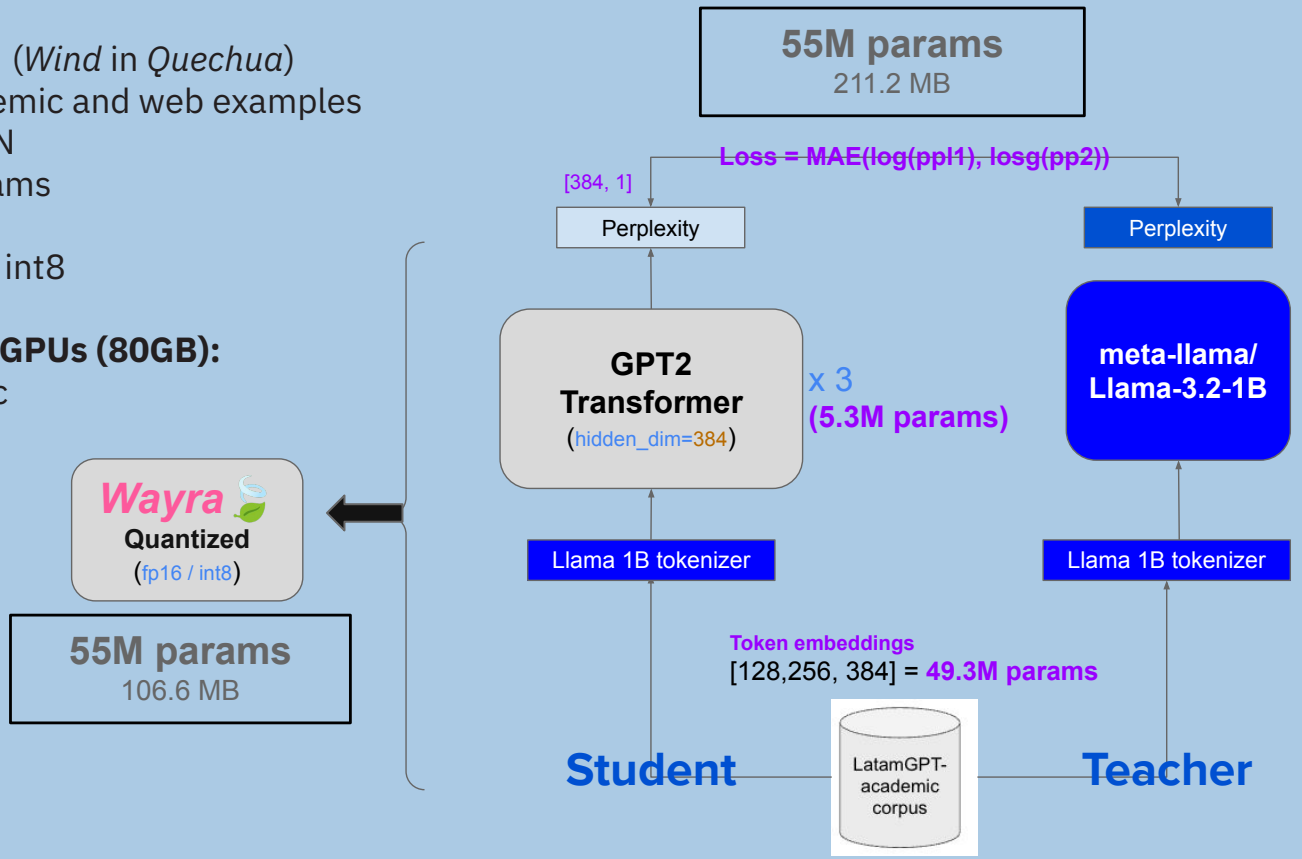
- **Sort each dataset by perplexity** (PPL):
  - from **easy** (low PPL) to **hard** (high PPL)
- **Pros**
  - **Stabilizes early training**: Easy examples reduce variance in gradients
  - **Leverages model priors**: Early convergence on known syntax, patterns
  - **Promotes efficient generalization**: **Harder examples fine-tune domain-specific skills**
- **Cons**
  - **Too expensive to compute for 200M examples (300B tokens) takes 196 hours with 8 GPUS A100**

$$\mathrm{PPL}_{\text{LLaMA-1B}}(x) = \exp\left(-\frac{1}{N}\sum_{i=1}^{N}\log p_M(x_i \mid x_{<i})\right)$$



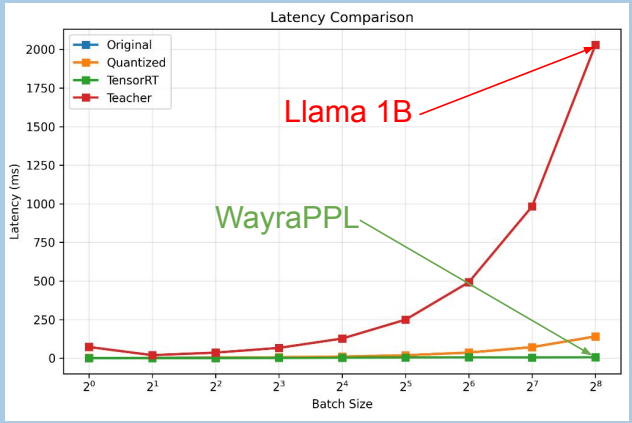
Solution: Knowledge Distillation

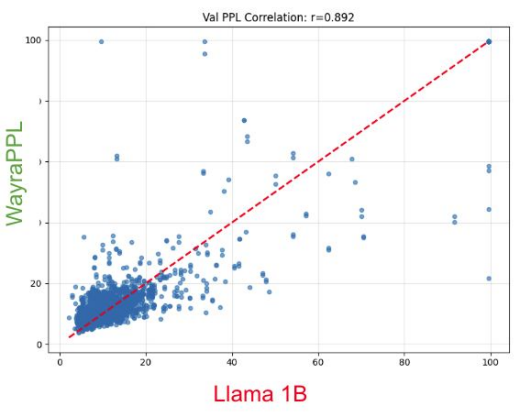
- **Model:** *Wayra* 🍃 (*Wind* in *Quechua*)
- **Dataset:** 3M academic and web examples
- **Languages:** ES, PT, EN
- **Size:** 55M params
- **Storage:** 107MB
- **Modes:** fp16 and int8
- **Soon in HuggingFace** 🤗
- **Throughput with 8 x A100 GPUs (80GB):**
  - 5.45K documents/sec
  - 101M docs in 4 hours



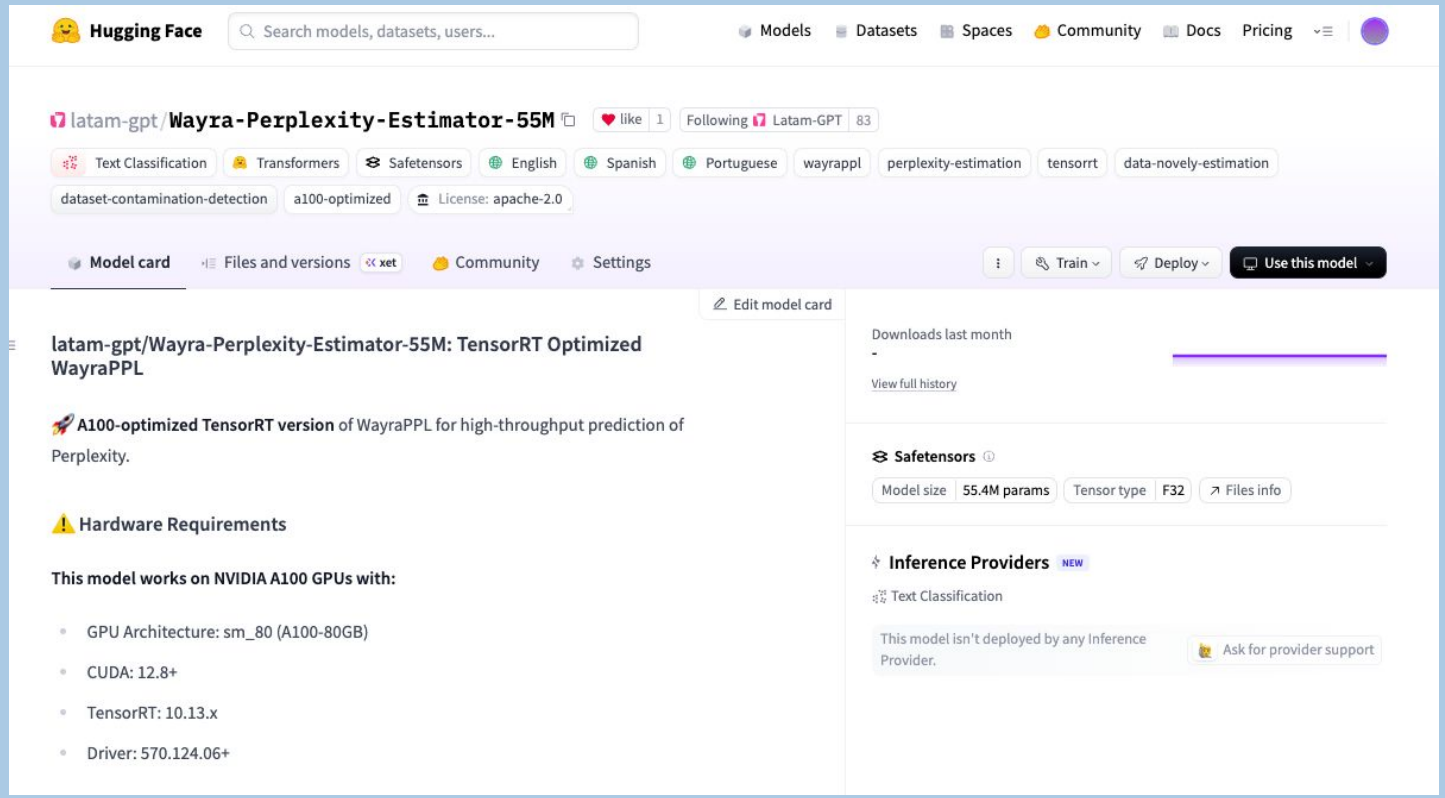
Solution: Knowledge Distillation

- While **KenLM** is an **n-gram model** trained using counts of word sequences
- **WayraPPL** is a fast proxy for real perplexity from a foundation model (Llama 1B)
- Designed to
  - **Reflect the inductive biases of larger LLMs** (but much faster)
  - Encode long sequences and **track dependencies beyond n-grams**
  - **Based on Transformer blocks with causal masks**
  - Useful in **curriculum learning**, **semantic filtering**, **alignment scoring**, etc.
  - **Needs GPU inference**





: https://huggingface.co/latam-gpt/Wayra-Perplexity-Estimator-55M



PARALLELIZATION

TP Group 1/8:

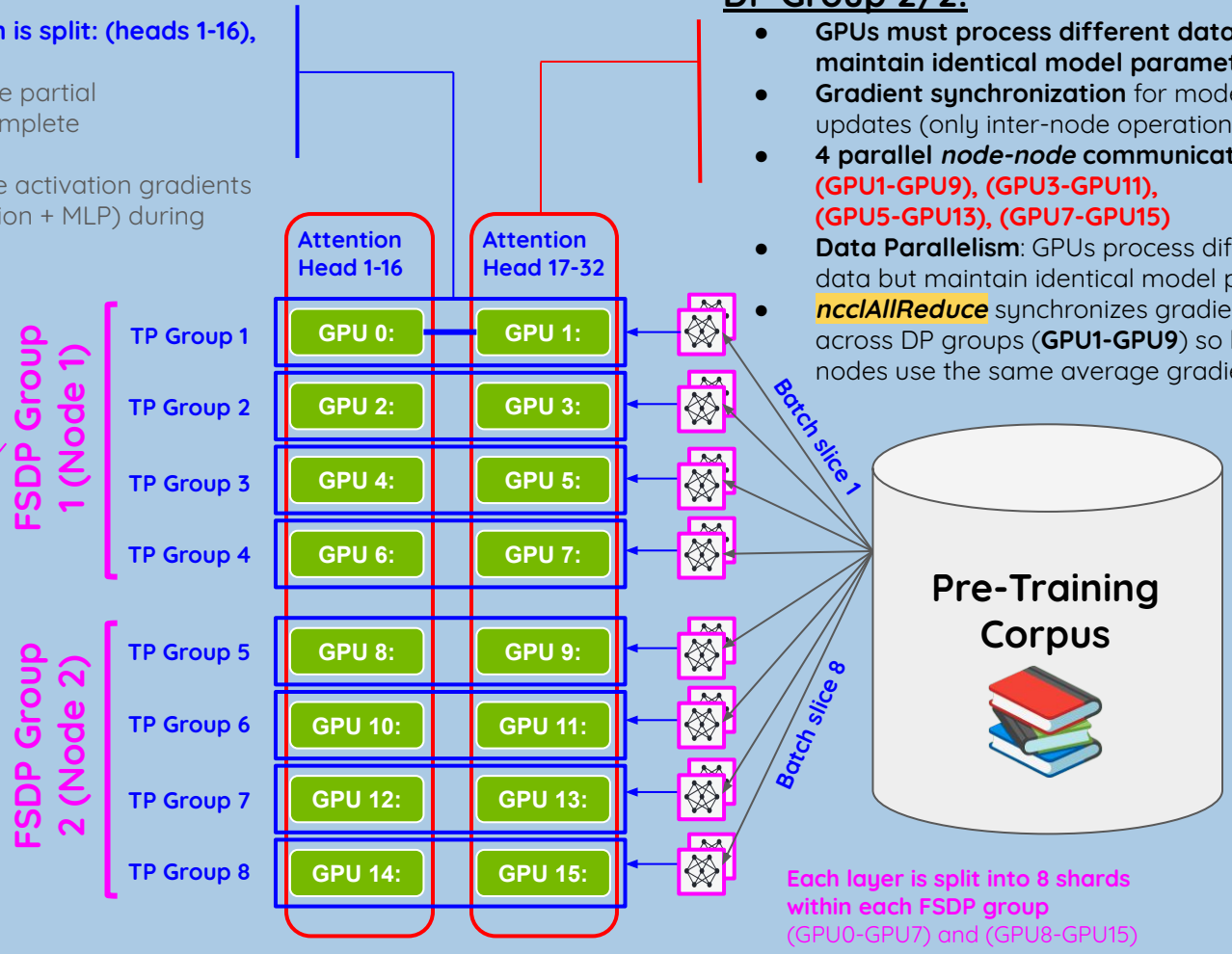
- **TP controls how computation is split: (heads 1-16), (heads 17-32)**
- **FW**: ***ncclAllReduce*** to combine partial attention/MLP outputs into complete activation/output tensor
- **BW**: ***ncclAllReduce*** to combine activation gradients
- Frequency: 2 per layer (Attention + MLP) during FW/BW

DP Group 2/2:

- **GPUs must process different data but maintain identical model parameters**
- **Gradient synchronization** for model updates (only inter-node operation, ***slow***)
- **4 parallel *node-node* communications: (GPU1-GPU9), (GPU3-GPU11), (GPU5-GPU13), (GPU7-GPU15)**
- **Data Parallelism**: GPUs process different data but maintain identical model params
- ***ncclAllReduce*** synchronizes gradients across DP groups (**GPU1-GPU9**) so both nodes use the same average gradient

FSDP Group 1/2:

- **Each GPU persistently holds 1/8 of the model (weights, gradients, and optimizer states), its shard**
- The memory saving comes from **not holding all** weights during the lifecycle — just the shard most of the time.
- **FW**: ***ncclAllGather*** collects sharded parameters from the other 7 GPUs to assemble complete layer on each GPU
- **BW**: ***ncclReduceScatter*** distributes gradients, so each GPU only stores the 1/8 gradient for its shard (sum across ranks already applied).
- Full parameter tensors **exist only briefly** around the forward/backward math (matmul/conv)



Each layer is split into 8 shards within each FSDP group (GPU0-GPU7) and (GPU8-GPU15)

Source: https://oflorez.substack.com/p/ever-wondered-how-to-scalably-train