Scaling Performance of Large Language Model Pretraining

Alexander Interrante-Grant, Carla Varela-Rosa, Suhaas Narayan, Chris Connelly, Albert Reuther
Lincoln Laboratory
Massachusetts Institute of Technology
Lexington, MA, USA
{ainterr, carla.varela-rosa, suhaas.narayan, connelly, reuther}@ll.mit.edu

***Abstract*—Large language models (LLMs) show best-in-class performance across a wide range of natural language processing applications. Training these models is an extremely computationally expensive task; frontier Artificial Intelligence (AI) research companies are investing billions of dollars into supercomputing infrastructure to train progressively larger models on increasingly massive datasets. Unfortunately, information about the scaling performance and training considerations of these large training pipelines is scarce in public literature. Working with large-scale datasets and models can be complex and practical recommendations are scarce in the public literature for tuning training performance when scaling up large language models. In this paper, we aim to demystify the large language model pretraining pipeline somewhat - in particular with respect to distributed training, managing large datasets across hundreds of nodes, and scaling up data parallelism with an emphasis on fully leveraging available GPU compute capacity.**

***Index Terms*—large language models, distributed training, data parallelism.**

I. Introduction

Scaling up LLMs has been the primary goal of most frontier AI research companies over the past several years [1]–[6]. Recent research on LLMs hash shown that, given sufficient data, simply scaling up the model size can predictably improve performance across a wide range of downstream tasks [7]. However, larger models require more computational resources to train. Further, larger datasets also increase the computational burden of training these models. It's no surprise then that companies like Google, OpenAI (via Microsoft), Anthropic, and others have invested billions of dollars into supercomputing infrastructure to produce some of the world's most advanced AI models.

TABLE I
Frontier Models

Company	Model	Release Date
OpenAI	GPT-4.5 [1]	February, 2025
Google	Gemini 2.5 [2]	July, 2025
Anthropic	Claude 3.5 Sonnet [3]	June, 2024
xAI	Grok 3 [4]	February, 2025
Mistral AI	Medium 3 [5]	May, 2025
DeepSeek	R1 [6]	January, 2025

Recent model releases from frontier AI companies (see Table I) reveal very little about their model architectures or training processes. Even basic information such as model sizes (parameter counts) are rarely reported publicly and left to rumor and speculation. This leaves researchers training similar models to resolve issues with scaling (distributed data access, bottleneck resolution, best practices) on their own.

In this work, we pretrain an LLM for a novel language application, examining scaling performance as we increase the size of both the dataset and the model, and report lessons learned along the way. All of this work was done on a computing cluster of several hundred nodes equipped with Nvidia H100 GPUs.

A. Supercomputing Infrastructure

The Lincoln Laboratory Supercomputing Center (LLSC) operates, maintains, and supports thousands of users operating at a variety of security levels. The Nvidia H100 nodes on which this work was completed is the TX-GAIN (GenAI Next) system, one of several partitions/clusters that comprise the LLSC's TX-Green supercomputer. TX-GAIN is a cluster of 316 HPE compute nodes with dual AMD EPYC 9254 24-core CPUs with 768 GB of DRAM and dual Nvidia Hopper H100-NVL GPUs with 94 GB of HBM GPU RAM. The H100-NVL GPUs have a NV-Link bridge that interconnect the two GPUs on each compute node. Each node mounts the central Lustre parallel storage array file systems, has 3.8 TB of local SSD storage, and includes a 25-Gigabit Converged Ethernet link to a non-blocking central Converged Ethernet cluster core switch. The TX-GAIN cluster was deployed in early summer 2025, and it is #114 on the June 2025 Top500 List.

II. Scaling Up Model Training

While the model and dataset are not the subject of this paper, some details of the model are relevant to its scaling performance. Our experiments involved pretraining a Bidirectional Encoder Representation for Transformers (BERT)-like encoder model [8] on a large, novel dataset of binary code. The model was pretrained on a Masked Language Modeling (MLM) task with 15% of tokens in the training dataset randomly masked. As we iterated on the model, we scaled from experimental training runs of a small 120M parameter model on a small shard of the dataset running on a single GPU, all the way up to a larger 350M parameter model trained on the full dataset distributed across 128 nodes with 256 GPUs in total.

A. Dataset

Our dataset is comprised of binary code in the form of functions (202M pretraining samples) compiled from open-source projects using the nixpkgs package manager [9] totaling just under 2 TB in size. The size of our dataset leads to our first roadblock in scaling training - sharing 2 TB of data across hundreds of nodes in a supercomputing environment can have considerable performance impacts. Addressing this problem leads to our first two recommendations.

1. **Preprocess and tokenize the entire dataset ahead of training**, storing only the necessary training data: tokenized inputs and attention masks. The bulk of the storage space for our dataset consisted of large volumes of function data with a relatively poor compression ratio. After tokenization and with only the fields necessary for pretraining, our dataset was only 25 GB in size (a reduction of 99%!).
2. If it is small enough, **duplicate your dataset across nodes prior to training**. We found that the initial cost of copying the entire dataset from network storage to local storage on each node was worth it to avoid nodes contending for network and network-attached storage resources throughout the course of training.

These two changes allowed us to eliminate a network storage bottleneck that would have prevented us from saturating our GPUs compute performance.

B. Parallelism

Starting with smaller training runs on a single GPU, we noticed that GPU utilization would spike briefly and then drop to 0% repeatedly throughout training. This leads to our next recommendation.

3. **Parallelize data loading, but only just as much as necessary**. Even though we did significant preprocessing ahead of training, simply loading data into GPU memory was still a bottleneck. We gradually increased the number of parallel data loaders until single GPU utilization stabilized near 100% - any more than this would simply be a waste of resources[1].

Once we were able to efficiently saturate a single GPU for training, we were ready to scale up to multiple GPUs. We used PyTorch Lightning [10] to enable multi-GPU and multi-node training. Figure 1 depicts training performance across a range of model sizes and node counts. This leads to our next insight.

4. For data parallel multi-node training, **network bandwidth is not as much of a bottleneck as it might seem**. As depicted in Figure 1 - model training performance scales roughly linearly with the number of nodes, even up to 128 data parallel nodes. This indicates that our workload is still GPU bound and not bottlenecked by network performance when propagating weight updates at the end of each batch.

[1]Note: we found that it was best to determine the optimal number of data loaders *after* optimizing training batch size to saturate GPU memory.

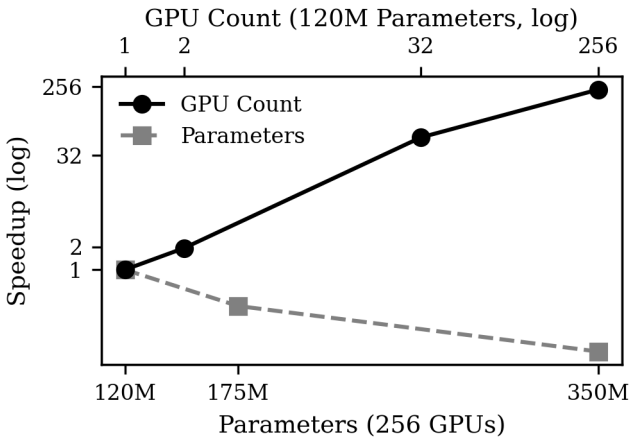

Fig. 1. Pretraining scaling performance.

Lastly, as we scaled up the model size (keeping a constant 128 nodes) we noticed a decrease in training performance.

5. **Larger models indirectly reduce training efficiency** with data parallelism because the increased parameter count requires more GPU memory, reducing training batch size. Our smallest (120M parameters) model was trained with a batch size of 184 samples, while our largest (350M parameters) only managed 20. Scaling any further than this would require model parallelism, which would require further tuning to optimize fully.

III. Conclusion

In this short paper we have provided some practical recommendations for optimizing LLM pretraining performance that we learned through the process of training our own. We hope that these recommendations are useful for other researchers attempting to scale up custom LLM pretraining to larger models and datasets.

References

[1] OpenAI, "Introducing GPT-4.5," 2025, OpenAI, Feb 2025. [Online]. Available: https://openai.com/index/introducing-gpt-4-5/

[2] Google, "Gemini 2.5: Pushing the Frontier with Advanced Reasoning, Multimodality, Long Context, and Next Generation Agentic Capabilities," 2025, arxiv:2507.06261. [Online]. Available: https://arxiv.org/abs/2507.06261

[3] Anthropic, "Claude 3.5 Sonnet," 2024, Anthropic, Jun 2024. [Online]. Available: https://www.anthropic.com/news/claude-3-5-sonnet

[4] xAI, "Grok 3 Beta — The Age of Reasoning Agents," 2025, xAI, Feb 2025. [Online]. Available: https://x.ai/news/grok-3

[5] Mistral AI, "Medium is the New Large," 2025, Mistral AI, May 2025. [Online]. Available: https://mistral.ai/news/mistral-medium-3

[6] DeepSeek-AI, "DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning," 2025, arxiv:2501.12948. [Online]. Available: https://arxiv.org/abs/2501.12948

[7] J. Wei, et al. "Emergent Abilities of Large Language Models," Transactions on Machine Learning Research (TMLR), 2022.

[8] J. Devlin, M. Chang, K. Lee, K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," 2018, arxiv:1810.04805. [Online]. Available: https://arxiv.org/abs/1810.04805

[9] NixOS. "nixpkgs," 2025, GitHub. [Online]. Available: https://github.com/NixOS/nixpkgs

[10] Lightning AI. "PyTorch Lightning,", 2025, GitHub. [Online]. Available: https://github.com/Lightning-AI/pytorch-lightning