Computational Bottlenecks of Training Small-scale Large Language Models

Saleh Ashkboos* **Iman Mirzadeh** **Keivan Alizadeh** **Mohammad Hossein Sekhavat**

Moin Nabi **Mehrdad Farajtabar** **Fartash Faghri**

Apple
saleh.ashkboos@inf.eth.ch,fartash@apple.com

Abstract

While large language models (LLMs) dominate the AI landscape, Small-scale large Language Models (SLMs) are gaining attention due to cost and efficiency demands from consumers. However, there is limited research on the training behavior and computational requirements of SLMs. In this study, we explore the computational bottlenecks of training SLMs (up to 2B parameters) by examining the effects of various hyperparameters and configurations, including GPU type, batch size, model size, communication protocol, attention type, and the number of GPUs. We assess these factors on popular cloud services using metrics such as *loss per dollar* and *tokens per second* [2]. Our findings aim to support the broader adoption and optimization of language model training for low-resource AI research institutes.

1 Introduction

Large Language Models (LLMs) are becoming increasingly popular in various fields due to their performance on a variety of tasks [6, 18, 8, 20, 5]. However, deploying large models widely such as on mobile hardware and edge devices is challenging due to the large memory and compute requirements [11, 12, 10]. These constraints have driven a growing interest in smaller language models (such as $\leq 2B$ parameters) as a viable alternative [24, 16, 23]. Recent work refer to these models as Small-scale large Language Models (SLMs) which can work well in environments where cost-efficiency and resource limitations are of significant concern, as well as on servers where the reduced cost of inference will be a dominant factor to attract and retain customers.

SLMs have demonstrated substantial potential in achieving competitive results despite their smaller size. Techniques such as pruning, distillation, and quantization have been employed to enhance their performance [2, 3, 17], allowing SLMs to perform on par with, and in some cases surpass, much larger models [4]. For example, Gemma-2B outperforms the largest OPT-175B [25], challenging the notion that sheer model size is the primary determinant of effectiveness. In addition to on par accuracy, SLMs can meet consumer demands for fast, efficient, and cost-effective AI without sacrificing task performance, making them increasingly attractive for organizations with limited computational budgets, such as small businesses and academic institutions.

While prior work mostly focused on optimizing SLMs for inference [15], relatively little attention has been paid to their training dynamics. This gap is significant, as the computational and infrastructure demands of training LLMs may not translate to SLMs. Given the diverse range of hardware

*Work done during an internship at Apple.

[2]We use average dollar cost ratios of cloud instance types based on publicly available pricing (Appx. A).

38th Conference on Neural Information Processing Systems (NeurIPS 2024) – 4th Workshop on Efficient Natural Language and Speech Processing (ENLSP).

configurations available on cloud platforms—such as GPU type, batch size, and communication protocols—there is a need for a systematic analysis of how these factors impact the training efficiency of SLMs, particularly when measured in terms of practical metrics such as *loss per dollar* and *tokens per second*. Our findings indicate that for smaller models, more affordable options like A100-40GB GPUs and Distributed Data Parallel (DDP) can be utilized without sacrificing performance. For larger models, advanced configurations, such as A100-80GB and H100-80GB GPUs paired with Flash Attention (FA) and Fully Sharded Data Parallel (FSDP), are necessary to handle larger batch sizes and prevent memory-related issues.

Recent advancements in the field underscore the importance of scaling AI systems not only for state-of-the-art performance but also for practical applications in real-world environments. The emerging trend toward SLMs suggests that a re-evaluation of hardware and computation strategies is essential. The contribution of this paper is to address the need for such evaluation, providing a systematic study on the computational bottlenecks and cost-efficiency of training SLMs up to 2B parameters on various cloud infrastructure and setups. We find that 1) FlashAttention is significantly more important for SLMs than LLMs, 2) Expensive hardware, e.g., H100-80GB and A100-80GB, is not necessarily cost effective for SLM training, 3) DDP is the best distributed training scheme for SLMs, and 4) Maximizing GPU memory utilization is not cost-optimal for SLM training.

2 Metrics

Our goal is to find architectures with maximal performance and minimum cost of training. It is common to measure the cost of training in terms of wall-clock time, iterations, or tokens. However, these metrics are incomplete for choosing a sufficient infrastructure within a budget. We recommend metrics that directly incorporate the dollar cost of hardware. Specifically, we aim to maximize the accuracy of the model while minimizing the cost or in other words optimize for *accuracy/dollar*. Prior works have discovered neural scaling laws controlling the relation between accuracy, loss, and number of seen samples or tokens during training [14]. In this paper, we focus on the number of tokens processed during the training (or $\frac{\text{Token}}{\text{Second}}$) for various architectures and measure the cost of processing tokens (or $\frac{\text{Token}}{\text{Dollar}}$). Given $\frac{\text{Token}}{\text{Sec}}$ measurements and $\frac{\text{Loss}}{\text{Token}}$ derived from scaling laws, we get

$$\frac{\text{Loss}}{\text{Dollar}} = \frac{\text{Loss}}{\text{Token}} \times \underbrace{\frac{\text{Token}}{\text{Second}} \times inv(\frac{\text{Dollar}}{\text{Second}})}_{\text{Our Metric}}, \tag{1}$$

where, $\frac{\text{Dollar}}{\text{Second}}$ is the cost of hardware and infrastructure which is extracted by averaging publicly available prices from various cloud providers for each hardware configuration (See Appx. A). The result of our analysis provides $\frac{\text{Token}}{\text{Dollar}}$ laws that combined with loss scaling laws can be used to find the minimal cost to a target loss. Other metrics of interest can be CPU/GPU utilization and Memory bandwidth usage that we leave for future work. We will report $\frac{\text{Token}}{\text{Dollar}}$ in various setups below where a value of 1k means training for 1k tokens costs $1.

3 Model and Parameters

We focus on LLaMa architectures [21, 9] as they are one of the most popular architectures in recent public LLMs and SLMs [1, 13]. The smallest LLaMa-2/3 have 7/8B parameters which is still too large for most mobile hardware. We extract the number of decoder blocks and parameters of our models by fitting a curve over the LLaMa models and use it for defining our models (see Fig. 5 in Appx. B). We evaluate four different model sizes with 100M, 500M, 1B, and 2B parameters. *Notably, we maximize over all configuration parameters not shown in the x-axis or legend of a figure.* That is, we run a large grid search over all combinations of configuration parameters listed below and each point in each plot is the best configuration given all parameters specified in the plot. This way, we find the the optimal $\frac{\text{Token}}{\text{Dollar}}$ and assume one can tune optimization hyperparameters such as learning rate to achieve the optimal convergence with the hardware-optimal configurations. We present details of these derivative models in Appx. B. Next, we define the configuration parameters:

- **GPU Types**: We evaluate the usage of three NVIDIA GPU types: A100-40GB, A100-80GB, and H100-80GB. We use BFloat16 data types in all GPUs.
- **GPU Numbers and Communication**: We study three main training configurations for each GPU type including single-node-single-GPU (1 GPU), single-node-multi-GPU (2, 4, and 8 GPUs), and

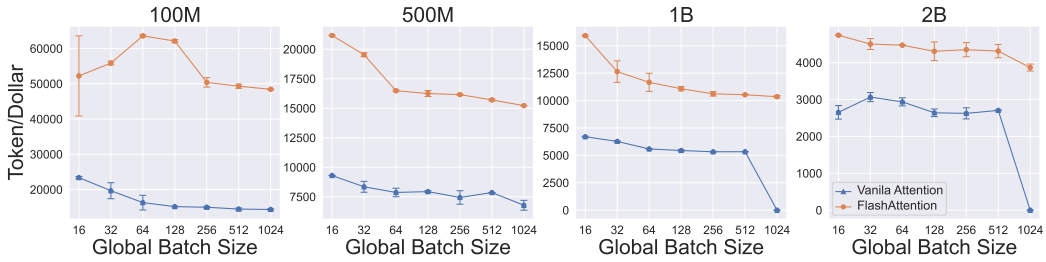

Figure 1: **FlashAttention is more cost-efficient for smaller models.** Maximum Token/Dollar across GPU-types, GPU-number, and communication type when we use FlashAttention. FlashAttention shows a significant Token/Dollar improvement over vanilla attention in smaller models and batch sizes. OOM runs are shown with 0. Token/Dollar =1k means training for 1k tokens costs $1.

multi-node-multi-GPU (16, 32, and 64 GPUs) settings. When we use more than a single GPU, we evaluate Distributed Data-Parallel (DDP) and Fully Sharded Data Parallel [26] (FSDP) for communication. For FSDP, we study two sharding policies: 1) `full` sharding where we shard all gradients, optimizer states, and weights, and 2) `grad_op` sharding where we shard only gradients and optimizer states (but keep the weights unsharded). We use RDMA/EFA.

- **Number of Samples**: We evaluate various number of samples fit into a single GPU during the training. We fix the sequence length to 1028 and iterate over the batch-size we fit into a single device. As we cannot fit 128 samples into a single GPU memory even in our smallest (100M) model, we study the per-device batch-size of 4, 8, 16, 32, and 64. We do not use gradient accumulation.
- **Flash Attention**: We study the affect of using FlashAttention [7] for the attention block.

4 Experimental Results

In this section, we present results on A100-40GB, A100-80GB, and H100-80GB. We implement our models in HuggingFace [22] and run our experiments using PyTorch [19] without any additional frameworks and use 1024 sequence length. We provide details of the runtime configuration in Appx. C. For each setup, we run our experiments at least 3 times, each with 10 training steps, and then report the average and error bars. We form our findings into research questions and answers.

Q1: How important is to use FlashAttention during the SLM training?

Figure 1 compares the use of FlashAttention2 [7] against vanilla attention for different global batch-sizes. First, we can see that FlashAttention (FA) significantly increases our Token/Dollar in SLMs. Notably, FA improves Token/Dollar more significantly for smaller models as the cost of attention is quadratic in context length and dominates when the hidden dimension shrinks. Such SLMs enter a data-bound regime where the data movement (CPU/GPU as well as GPU/GPU) becomes the main bottleneck. Finally, we can see that for larger models (1B and 2B), FA enables training of larger batch-sizes (1024) while vanilla attention results in out of memory error (OOM).

Q2: Given a fixed number of GPUs, what is the best GPU type for training an SLM?

Figure 2 shows the result of training our models using different GPU types: A100-40GB, and A100-80GB. Although we cannot see a consistent pattern for all models, we can see that A100-80GB GPU is a better choice when we use a large number of GPUs (32) to train larger models (1B and 2B). In such cases, A100-80GB can be used for training larger batch-sizes, while in smaller models, we can use 40GB GPU with cheaper price (see Appx. A for Hadrware prices).

Q3: What is the best communication scheme for training SLMs for different number of nodes?

Next, we study the role of using different parallelization schemes for SLM training. To this end, we study the use of Distributed Data Parallel (DDP), Fully Sharded Data Parallel with full sharding policy (FSDP-Full), and Fully Sharded Data Parallel with sharding gradients and optimizer states (FSDP-Grad+Optimizer). Figure 3 shows the result of training our models with different parallelization schemes on A100-80GB GPU. Our results show that for smaller models, DDP is a better choice due to the less communication volume. However, for largest model (2B), FSDP outperforms DDP as we can train larger batch-sizes (see Q4). Finally, we observe that FSDP-Grad+Optimizer outperforms FSDP-Full due to the lower communication overhead.

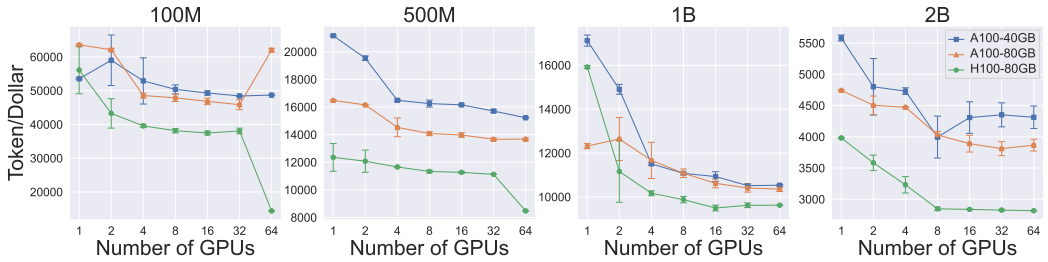

Figure 2: **H100 GPUs are not cost-efficient for training SLMs.** Maximum Token/Dollar for different GPU-type across batch-size and communication types. We use FlashAttention.

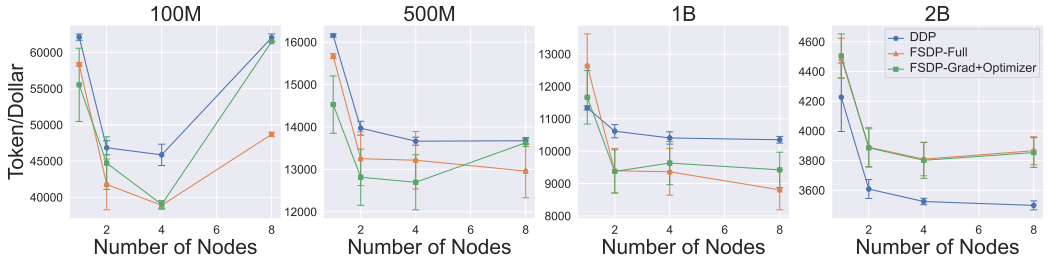

Figure 3: **DDP is the best scheme for training SLMs.** Maximum Token/Dollar for different GPU-nodes on A100-80GB across different batch-sizes. We use FlashAttention in our models. for a single node, we use 2, 4, and 8 GPUs while for 2 and 4 nodes we use 16 and 32 GPUs respectively.

Q4: What is the best communication scheme for training SLMs for different global batch-sizes? Fig. 4 shows the result of training SLM with various global batch-sizes using DDP, FSDP-Full, and FSDP-Grad+Optimizer for various per-device batch sizes. We cannot see a substantial difference in small batch sizes in our experiments. However, similar to Q3, FSDP always outperforms DDP for largest model (2B) and batch-sizes. In addition, FSDP enable training larger global batch-sizes (512) for larger models (2B) compared to DDP (which results in OOM).

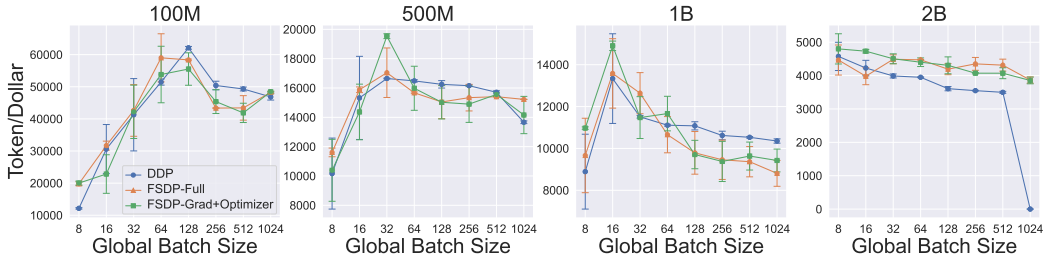

Figure 4: **For SLMs increasing global batch size saturates cost-efficiency before GPU memory is fully utilized.** Maximum Token/Dollar for different global batch-sizes across across different GPU-types, GPU numbers, and various per-device batch-sizes. We use FlashAttention in our models.

5 Summary and Conclusion

In this study, we examined the computational bottlenecks of training Small-scale Language Models (SLMs) up to 2B parameters, focusing on the impact of hyperparameters and hardware configurations. Our findings highlight the importance of Flash Attention (FA) for smaller models and batch sizes, where data movement is the primary bottleneck. FA also enables training larger models (1B and 2B) with batch sizes up to 512, avoiding out-of-memory (OOM) errors common with vanilla attention. Additionally, we found that A100-80GB GPUs are optimal for training larger models with many GPUs, while the more cost-effective A100-40GB works well for smaller models. In terms of distributed training, DDP is more efficient for smaller models, but FSDP outperforms it for larger models, particularly when training large batch sizes. These insights provide practical guidance for optimizing SLM training by offering clear strategies for selecting the most efficient hardware and parallelization methods based on model size.

References

[1] Marah Abdin, Sam Ade Jacobs, Ammar Ahmad Awan, Jyoti Aneja, Ahmed Awadallah, Hany Awadalla, Nguyen Bach, Amit Bahree, Arash Bakhtiari, Harkirat Behl, et al. Phi-3 technical report: A highly capable language model locally on your phone. *arXiv preprint arXiv:2404.14219*, 2024.

[2] Saleh Ashkboos, Maximilian L Croci, Marcelo Gennari do Nascimento, Torsten Hoefler, and James Hensman. Slicegpt: Compress large language models by deleting rows and columns. *arXiv preprint arXiv:2401.15024*, 2024.

[3] Saleh Ashkboos, Amirkeivan Mohtashami, Maximilian L Croci, Bo Li, Martin Jaggi, Dan Alistarh, Torsten Hoefler, and James Hensman. Quarot: Outlier-free 4-bit inference in rotated llms. *arXiv preprint arXiv:2404.00456*, 2024.

[4] Hritik Bansal, Arian Hosseini, Rishabh Agarwal, Vinh Q Tran, and Mehran Kazemi. Smaller, weaker, yet better: Training llm reasoners via compute-optimal sampling. *arXiv preprint arXiv:2408.16737*, 2024.

[5] Rishi Bommasani, Drew A Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, et al. On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258*, 2021.

[6] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in Neural Information Processing Systems*, 33:1877–1901, 2020.

[7] Tri Dao. Flashattention-2: Faster attention with better parallelism and work partitioning (2023). *arXiv preprint arXiv:2307.08691*, 2023.

[8] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[9] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

[10] Jianping Gou, Baosheng Yu, Stephen J Maybank, and Dacheng Tao. Knowledge distillation: A survey. *International Journal of Computer Vision*, 129(6), 2021.

[11] Torsten Hoefler, Dan Alistarh, Tal Ben-Nun, Nikoli Dryden, and Alexandra Peste. Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks. *arXiv preprint arXiv:2102.00554*, 2021.

[12] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.

[13] Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. Mistral 7b. *arXiv preprint arXiv:2310.06825*, 2023.

[14] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

[15] Sehoon Kim, Coleman Hooper, Thanakul Wattanawong, Minwoo Kang, Ruohan Yan, Hasan Genc, Grace Dinh, Qijing Huang, Kurt Keutzer, Michael W Mahoney, et al. Full stack optimization of transformer inference: a survey. *arXiv preprint arXiv:2302.14017*, 2023.

[16] Zechun Liu, Changsheng Zhao, Forrest N. Iandola, Chen Lai, Yuandong Tian, Igor Fedorov, Yunyang Xiong, Ernie Chang, Yangyang Shi, Raghuraman Krishnamoorthi, Liangzhen Lai, and Vikas Chandra. Mobilellm: Optimizing sub-billion parameter language models for on-device use cases. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=EIGbXbxcUQ.

[17] Saurav Muralidharan, Sharath Turuvekere Sreenivas, Raviraj Joshi, Marcin Chochowski, Mostofa Patwary, Mohammad Shoeybi, Bryan Catanzaro, Jan Kautz, and Pavlo Molchanov. Compact language models via pruning and knowledge distillation. *arXiv preprint arXiv:2407.14679*, 2024.

[18] OpenAI. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

[19] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

[20] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67, 2020.

[21] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

[22] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations*, pages 38–45, 2020.

[23] Mengzhou Xia, Tianyu Gao, Zhiyuan Zeng, and Danqi Chen. Sheared llama: Accelerating language model pre-training via structured pruning. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=09iOdaeOzp.

[24] Peiyuan Zhang, Guangtao Zeng, Tianduo Wang, and Wei Lu. Tinyllama: An open-source small language model. *arXiv preprint arXiv:2401.02385*, 2024.

[25] Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, et al. Opt: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068*, 2022.

[26] Yanli Zhao, Andrew Gu, Rohan Varma, Liang Luo, Chien-Chin Huang, Min Xu, Less Wright, Hamid Shojanazeri, Myle Ott, Sam Shleifer, et al. Pytorch fsdp: experiences on scaling fully sharded data parallel. *arXiv preprint arXiv:2304.11277*, 2023.

A Hardware Costs: Details

To get reasonable (and practical) GPU price, we extracted the price of allocating different GPU types from two providers: **Lambda Labs**[3], and **Google Cloud Platform (GCP)**[4]. We then normalized and average over them to use the proportional ratio in our experiments. Table 1 shows the details of our hardware prices.

GPU Type	Lambda Labs	GCP	Avg.
A100-40GB	1.00	1.00	1.00
A100-80GB	1.39	1.54	1.46
H100-80GB	2.32	2.73	2.53

Table 1: Normalized prices for a single GPU allocation (for an hour) on **Lambda Labs** and **Google Cloud Platform (GCP)**. We use averged values in our experiments.

B Model Parameters: Details

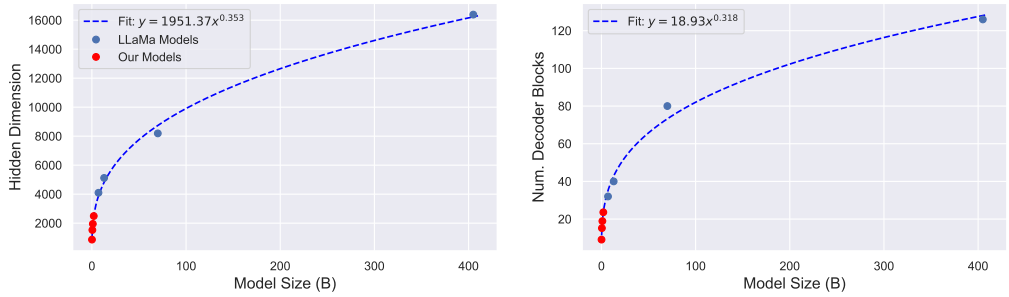

Figure 5: Curve fitting to extract our SLM model sizes using LLaMa-2/3 models. **Left**: Hidden dimension extraction. **Right**: Number of decoder blocks.

Table 2 shows the architecture details of the models we used in this paper.

Model	#L	#D	#Param.
100M	9	864	130,569,920
500M	15	1536	526,550,016
1B	19	1920	970,458,240
2B	24	2496	1,969,027,008

Table 2: Model details in this study. We use LLaMa-style architectures and show the number of decoder layers (**#L**), the dimension of the model (**#D**), and the exact number of parameters (**#Param.**).

C Runtime Configuration: Details

Table 3 presents the PyTorch version and its dependencies used in the runtime environment of this study.

[3] http://lambdalabs.com
[4] https://cloud.google.com

Dependency	Version
PyTorch	2.3.0
NCCL	2.20.5
CUDA	12.2.2
Fabric interface provider	119.0
Libfabric API	1.22.0
Python	3.10.13
Ubuntu	20.04

Table 3: Dependency versions.