Empirical Training Time Prediction for LLM Fine-Tuning Using Scaling Laws

Chianing Wang
Toyota InfoTech Lab USA & Santa Clara University
johnny.wang@toyota.com

Younghyun Cho
Santa Clara University
younghyun.cho@scu.edu

***Abstract*—This paper presents a methodology to efficiently estimate the training time and associated computational cost of fine-tuning Large Language Models (LLMs). Our approach introduces a novel two-stage methodology that incorporates an intelligent tuning algorithm called the Scaling Laws Smart Tuning (SLST) algorithm for efficient sample data collection and a time prediction model combining scaling laws with Gradient Boosting techniques. The scaling laws capture broad training trends concerning model parameters and dataset sizes, while Gradient Boosting models effectively reduce residual errors by modeling complex nonlinear relationships directly from data. Through this integrated approach, we achieve a high accuracy in training time predictions, which can significantly enhance resource planning and infrastructure decision-making. Our results demonstrate the effectiveness of the methodology, which balances interpretability and predictive accuracy, and highlight its scalability across various computational and parallelism environments.**

I. Background and Research Question

A. Infrastructure Cost Estimation for LLM

The rapid advancement of generative AI and Large Language Models (LLMs) [2] has transformed numerous industries. Foundation models [1], typically large-scale models pre-trained on extensive datasets, capture general knowledge and can be fine-tuned for various downstream tasks. Despite their success, substantial computational and financial demands during pre-training and fine-tuning [5] present significant challenges. As investments in LLMs grow, accurately estimating infrastructure costs becomes critical for efficient resource management, evaluating project feasibility, and guiding strategic infrastructure decisions.

B. Scaling Laws for LLM

The LLM scaling laws by Kaplan et al. [13] empirically quantify the relationships among model size (N), dataset size (D), and computational resources required (C). Pioneering studies on models like GPT-3 [2] demonstrated that power-law relationships effectively model training-related costs. Initially drawn from physics and natural sciences, these scaling laws characterize nonlinear and predictable scaling behaviors. However, much existing research has primarily focused on predicting training loss rather than training time, creating a noticeable gap in the fine-tuning context.

C. Improving LLM Fine-Tuning Performance

While large-scale pre-training remains resource-intensive and primarily accessible to well-resourced entities, fine-tuning has become increasingly feasible through Parameter-Efficient Fine-Tuning (PEFT) techniques [6], particularly Low-Rank Adaptation (LoRA) [11]. LoRA selectively updates a small subset of model parameters, significantly reducing computational overhead. It operates independently from parallel computation methods such as Pipeline Parallelism (PP) [12] and Tensor Parallelism (TP) [17], which primarily manage computational distribution rather than directly influencing learning dynamics.

D. Motivation and Research Question

This work proposes a methodology for efficiently and accurately estimating fine-tuning training times for LLMs. While significant research exists on LLM training dynamics, most studies emphasize training loss rather than training time, leaving several key research gaps:

1) **Emphasis on Pre-training Over Fine-tuning:** Most studies have primarily examined scaling laws within pre-training scenarios. For instance, Hernandez et al. [10] conducted an extensive analysis of scaling behaviors for model transfer and pre-training, offering valuable insights into optimal model sizes and resource allocation. While their findings could potentially be adapted to fine-tuning contexts, their primary focus remains pre-training, which inherently differs in runtime characteristics compared to fine-tuning. Additionally, although predictions based on pre-training data might theoretically apply to fine-tuning scenarios, practical experimental validation remains lacking, leaving the feasibility of such adaptation uncertain.
2) **Focus on Training Loss Instead of Runtime Prediction:** Existing research typically emphasizes predicting and minimizing training loss, often neglecting explicit modeling of runtime, which is crucial for practical resource management and scheduling. Zhang et al. [22] provided comprehensive runtime analyses across various stages of LLM development, yet generalized predictive frameworks remain sparse.
3) **Limited Integration of Scaling Laws with Empirical Fine-tuning Time Prediction:** Scaling laws have been foundational in understanding pre-training performance, but integration into runtime prediction methodologies for fine-tuning remains relatively unexplored. Xia et al. [21] developed analytical methods for estimating fine-tuning

costs on specific GPU setups, but their approach does not explicitly employ scaling laws for generalized prediction across different computational environments.

Addressing these gaps, this paper aims to answer the following research question: *How can power-law scaling relationships effectively predict computational costs (training time) for LLM fine-tuning, thereby enabling efficient and sustainable AI infrastructure planning?*

II. Two-Stage Methodology

To address the aforementioned research question, we propose a comprehensive approach combining theoretical modeling and empirical validation. Our method involves two primary stages: first, efficient sample data collection guided by the Scaling Laws Smart Tuning (SLST) algorithm to ensure data quality and adherence to the LLM Scaling Laws; second, leveraging the collected data to predict training time (T) using redesigned LLM Scaling Laws formulas. Constants and exponents within these formulas are determined via the curve-fitting method, and the residuals, which are the differences between observed and predicted training time, are further minimized using Gradient Boosting.

A. Scaling Laws Smart Tuning

Our method begins with sample collection via fine-tuning LLMs. To ensure valid and predictive training time (T) data, the training loss (L) must conform to LLM Scaling Laws. Concurrently, hyperparameters such as learning rate and batch size are tuned during sample data collection, explicitly guided by the Scaling Laws. The LLM Scaling Laws provide a reliable and interpretable framework, characterizing the relationships among model size (N), dataset size (D), and training dynamics. By leveraging it, we can significantly reduce the number of empirical trials while ensuring the quality of the collected data samples, that is suitable for training time predictions. To formalize this approach, we introduce the Scaling Laws Smart Tuning (SLST) algorithm (Algorithm 1). The algorithm operates as follows:

- Pre-defines the key parameters such as model sizes $N[\,]$, maximum dataset size D, e.g., 80k # of tokens (22MB), and dataset intervals (lines 1–3).
- Initializes the hyperparameters such as *learning rate* (lr) and *batch size* (bs) along with their bounds (lines 5–9).
- Trains the language model using the $llm\_fine\_tuning()$ function with current parameters (line 11).
- Evaluates training logs at specified intervals(lines 12–13), as illustrated in Figure 1, ensuring the loss L adheres to the LLM Scaling Laws which is the L should improve as we increase N and D (lines 14–19).
- If non-compliance occurs, adjusts the hyperparameters and retrains (lines 20–26).
- Outputs optimal parameters upon compliance and progresses systematically through dataset intervals, as demonstrated in Figure 2, then advances to subsequent model sizes (lines 28–31).

Algorithm 1 Scaling Laws Smart Tuning Algorithm

Input: Define ML models N, max data sizes D with interval $range$.
Define initial batch size bs and learning rate lr.
Define batch size, learning rate boundary (min_bs, max_lr).
Output: Optimal learning rate lr, batch size bs, loss L, time T

1: $N[\,] \leftarrow get\_models\_list()$
2: $D \leftarrow get\_max\_dataset()$
3: $range \leftarrow get\_dataset\_range(D)$
4: **for** each n in $N[125m, 1.3b, \ldots, 66b\}$ **do**
5: $lr \leftarrow get\_initial\_lr\_by\_model(n)$
6: $bs \leftarrow get\_initial\_bs\_by\_model(n)$
7: $max\_lr \leftarrow get\_max\_ls\_by\_model(n)$
8: $min\_bs \leftarrow get\_min\_bs\_by\_model(n)$
9: Initialize parameters: $totalTime \leftarrow 0$, $prevDLoss \leftarrow 0$, $currD \leftarrow 1$
10: **while** $currD \neq patterns[range]$ **do**
11: $log \leftarrow llm\_fine\_tuning(lr, bs, n, D)$
12: $step \leftarrow get\_log\_step(log)$
13: $patterns[\,] \leftarrow$ np.linspace$(1, steps, range)$
14: **for** each d in $patterns[currD, \ldots]$ **do**
15: read log file at d
16: extract values $Loss$, $Time$ from log file
17: $prevNLoss \leftarrow get\_previous\_loss(N[n-1], d)$
18: $currLoss \leftarrow Loss$
19: **if** $currLoss > prevNLoss$ **or** $currLoss > prevDLoss$ **then**
20: **if** $bs > min\_bs$ **then**
21: $bs \leftarrow bs/2$
22: **else if** $lr < max\_ls$ **then**
23: $lr \leftarrow lr \times 2$
24: **end if**
25: $currD \leftarrow d$
26: **break**
27: **else**
28: $totalTime \leftarrow totalTime + Time$
29: **Output:** lr, bs, $currLoss$ as L, and $totalTime$ as T
30: $totalTime \leftarrow 0$
31: $prevDLoss \leftarrow currLoss$
32: **end if**
33: **end for**
34: **end while**
35: **end for**

B. Training Time Prediction

The second stage consists of two phases: Scaling Law T formulation and Gradient Boosting.

1) Scaling Laws T Formulation: Leveraging SLST, we collect comprehensive sample data across various GPU and parallelism configurations. We analyze this sample data using SciPy's optimization curve fitting module [18], employing nonlinear least squares to fit the redesigned training time scaling laws formula and calibrate our own formula, effectively capturing the inherent nonlinearities.

Power laws [20] characterize nonlinear relationships as straight lines on log-log plots, forming the basis for LLM Scaling Laws. While LLM Scaling Laws for training loss (L) typically show decreasing trends by dividing model size (N) and dataset size (D) by constants, training time (T) shows increasing trends. First, to predict training time in regards with N and D, we adapt a scaling formula by multiplying N and D

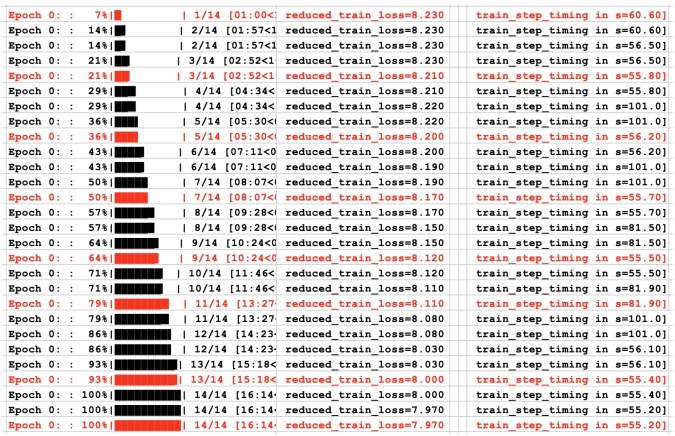
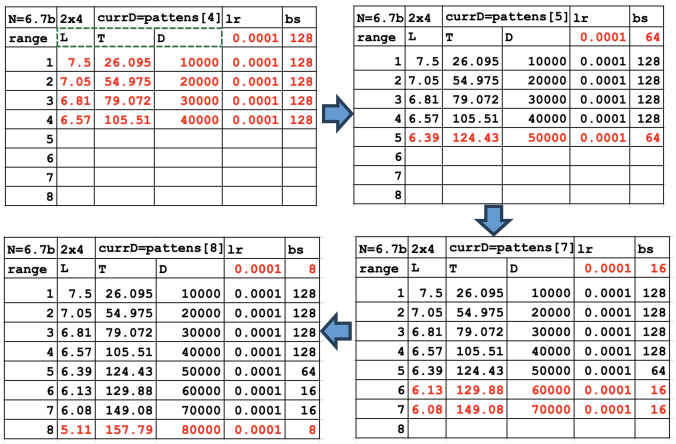


Fig. 1: SLST Log Example

Model Size N=125m and Dataset Size D=80k with 14 steps and 8 ranges(intervals)



N=6.7b	2x4	currD=pattens[4]		lr	bs
range	L	T	D	0.0001	128
1	7.5	26.095	10000	0.0001	128
2	7.05	54.975	20000	0.0001	128
3	6.81	79.072	30000	0.0001	128
4	6.57	105.51	40000	0.0001	128
5					
6					
7					
8					

N=6.7b	2x4	currD=pattens[5]		lr	bs
range	L	T	D	0.0001	64
1	7.5	26.095	10000	0.0001	128
2	7.05	54.975	20000	0.0001	128
3	6.81	79.072	30000	0.0001	128
4	6.57	105.51	40000	0.0001	128
5	6.39	124.43	50000	0.0001	64
6					
7					
8					

N=6.7b	2x4	currD=pattens[7]		lr	bs
range	L	T	D	0.0001	16
1	7.5	26.095	10000	0.0001	128
2	7.05	54.975	20000	0.0001	128
3	6.81	79.072	30000	0.0001	128
4	6.57	105.51	40000	0.0001	128
5	6.39	124.43	50000	0.0001	64
6	6.13	129.88	60000	0.0001	16
7	6.08	149.08	70000	0.0001	16
8					

N=6.7b	2x4	currD=pattens[8]		lr	bs
range	L	T	D	0.0001	8
1	7.5	26.095	10000	0.0001	128
2	7.05	54.975	20000	0.0001	128
3	6.81	79.072	30000	0.0001	128
4	6.57	105.51	40000	0.0001	128
5	6.39	124.43	50000	0.0001	64
6	6.13	129.88	60000	0.0001	16
7	6.08	149.08	70000	0.0001	16
8	5.11	157.79	80000	0.0001	8

Fig. 2: SLST trys diff lr and bs with Result Example

Model Size N=6.7b, G (# of GPUs)=8 and PPxTP=2x4; L is Training Loss, T is Training Time, D is Dataset Size (# of Tokens), lr is Learning Rate and bs is Batch Size

with constants, aligning with the observed increasing behavior in formula (1).

$$T(N,D) = \left[\left(N_c N^{\frac{\alpha N}{\alpha D}}\right) + D_c D\right]^{\alpha D} \tag{1}$$

Then, in order to model the training time for Pipeline Parallelism (PP) and Tensor Parallelism (TP), each of which would contribute to reduce the training time, we extend formula (1) with the TP and PP time factors that are modeled via two approaches: first, Power law based TP (2) and PP (3) factors, and second, Amdahl's law based TP (4) and PP (5) factors, as follows.

$$Factor(TP)_P = \left(\frac{TP_c}{TP}\right)^{\alpha TP} \tag{2}$$

$$Factor(PP)_P = \left(\frac{PP_c}{PP}\right)^{\alpha PP} \tag{3}$$

Amdahl's Law [9] illustrates performance limits in parallelization contexts. It highlights diminishing returns as parallel resources grow, providing accurate training-time improvement estimates:

$$Factor(TP)_A = \left[\left(\frac{\gamma}{TP} + 1 - \gamma\right)\right] \tag{4}$$

$$Factor(PP)_A = \left[\left(\frac{\delta}{PP} + 1 - \delta\right)\right] \tag{5}$$

Note that the factors $Factor(PP)$ and $Factor(TP)$, which represent the scaling behavior due to GPU, tensor, and pipeline parallelism, respectively, are explicitly combined into single, comprehensive equations. For clarity and concise reference, these unified equations, including base terms and both factors, are summarized in Table I:

TABLE I: Candidate Scaling Laws Formulas for $T(\,)$

Formula	Base and Factors
$T_A(N,D,TP,PP)$	$\left[\left(N_c N^{\frac{\alpha N}{\alpha D}}\right) + D_c D\right]^{\alpha D} \times \left[\left(\frac{\gamma}{TP} + 1 - \gamma\right)\right] \times \left[\left(\frac{\delta}{PP} + 1 - \delta\right)\right]$
$T_P(N,D,TP,PP)$	$\left[\left(N_c N^{\frac{\alpha N}{\alpha D}}\right) + D_c D\right]^{\alpha D} \times \left(\frac{TP_c}{TP}\right)^{\alpha TP} \times \left(\frac{PP_c}{PP}\right)^{\alpha PP}$

2) Residual Reduction via Gradient Boosting: Gradient Boosting [8], a sequential ensemble machine learning technique, minimizes pseudo-residuals iteratively. It enhances the Training Time Scaling Laws through a hybrid approach: initially fitting a physics-inspired scaling model to broadly capture trends, then employing Gradient Boosting (e.g., XGBoost [3], LightGBM [14], or CatBoost [16]) to refine predictions by learning residual nuances. This combination effectively integrates interpretability with predictive accuracy, significantly enhancing model reliability.

Gradient Boosting [8] enhances the Scaling Laws model by minimizing residuals through a sequential ensemble of weak learners. This approach refines predictions beyond the Scaling Laws' broad trends, significantly improving accuracy and robustness.

III. Experiment

To validate our time prediction methodology, we conducted comprehensive experiments utilizing the Meta OPT (Open Pre-trained Transformer) model [23], selected due to its status as a robust foundation model and the extensive range of model sizes available. In our experiments, we focus on fine-tuning LLMs using the Low-Rank Adaptation (LoRA) [11] technique. The dataset selected for our experiments is the Self-Instruct dataset [19], containing 52k instructions along with 82k paired inputs and outputs. The dataset was segmented into eight intervals, ranging from 10,000 to 80,000 tokens, and sizes from 2.6MB to 22MB. We utilized a computing cluster equipped with 16 NVIDIA H100 GPUs [4] and leveraged the Megatron framework [17], enabling various experimental scenarios involving different GPU counts and levels of pipeline and tensor parallelism.

A. Scaling Laws Smart Tuning

We first compare our SLST algorithm against a simple grid search approach. The grid search exhaustively explores 16 scenarios for each model-dataset combination, using 4

TABLE II: Number of Trials: Brute Force Grid Search vs. SLST

D(=80K) \ N	125M	1.3B	2.7B	6.7B	13B	30B	66B	**Total**
SLST Trials (P=4)	1	1	1	1	2	2	4	**12**
Brute Force (P=4)	128	128	128	128	128	128	128	**896**
SLST Trials (P=8)	0	1	1	1	3	1	4	**11**
Brute Force (P=8)	0	128	128	128	128	128	128	**768**
SLST Trials (P=16)	0	0	1	1	0	3	4	**9**
Brute Force (P=16)	0	0	128	128	0	128	128	**512**

TABLE III: Test Results for Self-Instruct Dataset

Formula	**MAE_S (↓)**	R^2**_S (↑)**	**MAE_C (↓)**	**RMAE_C (↓)**	R^2**_C (↑)**
$T_A()$	**20.974**	**0.790546**	**12.710**	**16.695**	**0.918035**
$T_P()$	20.963	0.790435	13.307	17.624	0.908659

Testing Dataset D=60k (16MB)~80K (22MB) with Masking Out non-compliance LLM Scaling Laws results for different # of GPU and Parallelism Configurations. The metrics with $\_S$ means performance result after 1st phase: scaling laws T; the metrics with $\_C$ means performance result after 2nd phase: Gradient Boosting (CatBoost).

different learning rates and 4 different batch sizes. Partial experimental results for configurations with $PP \times TP = 16 \times 1$, $PP \times TP = 8 \times 2$, and $PP \times TP = 4 \times 4$ using 16 GPUs are presented in Table II. The results clearly illustrate SLST's efficiency in significantly reducing the number of trials required during sample data collection.

B. Training Time Prediction

Following the sample data collection via SLST, we evaluated two candidate scaling laws formulas, which are Amdahl's law-based $T_A()$ and Power Law-based $T_P()$, presented in Table I. Sample data collected using SLST from 10k (2.6MB) to 50k (13MB) tokens served as the training set, while the 60k (16MB) to 80k (22MB) tokens range was reserved for testing. The GPU configurations varied systematically, with the total number of GPUs (G) ranging from 16 down to 8, and the pipeline parallelism (P) adjusted accordingly—for instance, when $G = 16$, configurations included $PP \times TP = 16 \times 1$, 8×2, and 4×4, among others, ensuring comprehensive coverage of parallelism combinations. Constants and exponents within the formulas were determined using curve fitting, and residual errors were further minimized using CatBoost. Test results comparing the models are shown in Table III.

The results in Table III indicate the superior performance of the $T_A()$ model, evidenced by the lowest MAE and RMAE and the highest R^2 scores. These outcomes highlight the effectiveness of Amdahl's law-based factors in modeling GPU and parallelism scaling more accurately than the power law-based factors. Optimized constants and exponents example for Self-Instruct Dataset for the $T_A()$ model after curve fitting are listed in Table IV. Further examples, shown in Figures 3 illustrate the prediction trends clearly across various test scenarios, demonstrating the scalability of model sizes and dataset intervals.

IV. Conclusion and Future Works

In conclusion, this research presents a robust methodology for accurately predicting training time and computational costs associated with LLM fine-tuning. We introduced a two-stage predictive framework, incorporating the novel Scaling Laws Smart Tuning (SLST) algorithm for efficient sample data collection alongside a time predictive approach. This time prediction model combines the LLM scaling laws with time modeling for GPU parallelism, and we use Gradient Boosting to enhance prediction accuracy. The approach can effectively capture broad trends related to model parameters and dataset sizes, while Gradient Boosting models further refine these predictions by modeling nonlinear relationships and reducing residual errors. This integrated approach achieves high accuracy and reliability, striking a balance between explainability and predictive flexibility, which can be significantly helpful in improving resource planning and informing strategic infrastructure decisions across diverse computational and parallelism environments.

TABLE IV: Constants and Exponents Example for Self-Instruct Dataset

Constant and Exponents for $T_A()$	**Values**
N_c	9.52074184e-04
αN	7.92198609e-01
D_c	1.72112915e-044
αD	2.17499781e+00
γ	6.52994195e-30
δ	9.80222336e-01

For future work, we plan to further enhance the robustness and broader applicability of our approach by:

- Expanding our experimental dataset with larger sizes and various types of datasets, to improve the generalizability and robustness of predictions.
- Evaluating the methodology's applicability advances not only OPT-based fine-tuning but also extends to other LLMs such as Llama [7] [15], future pre-training phases, inference tasks, and potentially diverse domains like Computer Vision.
- Investigating the effectiveness of our methodology across various GPU vendors and extending this to include infrastructure components beyond GPU, such as networking, storage, power, and thermal management.

While initial sampling and the two-phase prediction approach remain essential for calibrating predictive models, subsequent predictions significantly reduce the need for exhaustive empirical evaluations. By enabling accurate estimations of training times and computational resources across various configurations, our predictive methodology optimizes resource allocation, reduces operational costs, and supports effective strategic decision-making, particularly in large-scale or unexplored scenarios. These initiatives will further demonstrate the versatility and extend the practical utility of our predictive model, ensuring comprehensive adaptability and robust performance across diverse computational environments.

V. Acknowledgements

This work was supported by Toyota Motor Corporation and Toyota InfoTech Labs USA. We used the GPU computing resources and the high-performance server infrastructure at Toyota InfoTech Labs USA for the experiments of this work. We thank Ryokichi Onishi, Hiroshi Abe, and Onur Altintas for their comments and feedback on this work. We also thank Purvang Lapsiwala for his technical support.

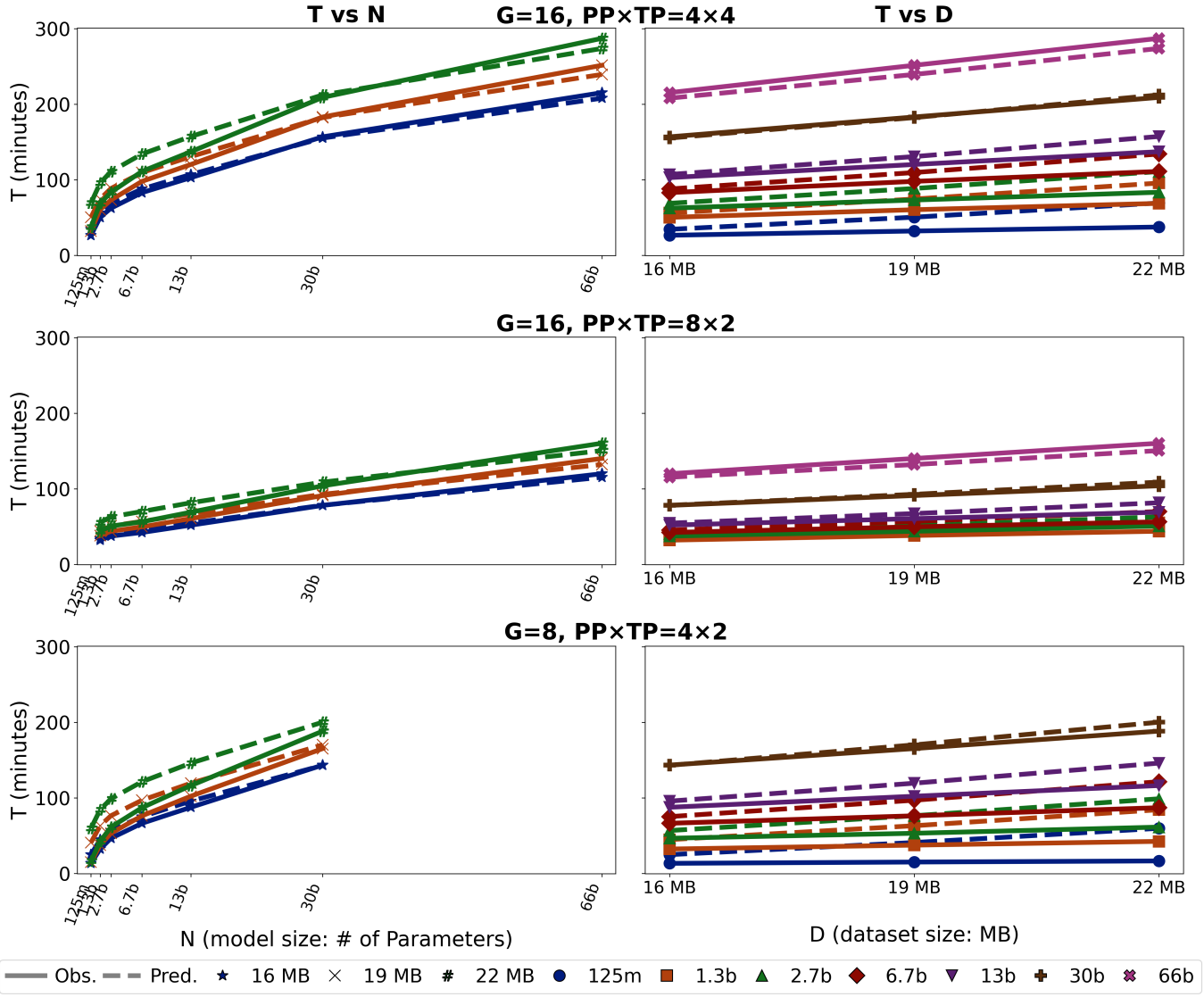


Fig. 3: Training time vs model size (left) and Self-Instruct dataset size (right) across G/P settings prediction examples. Our experiments cover model sizes ranging from $N = 125$m to 66b parameters, with testing dataset sizes between $D = 60$K tokens (16MB) and 80K tokens (22MB). We present example prediction results for various GPU parallelism configurations, denoted as $(G, PP \times TP)$, specifically $(16, 4 \times 4)$, $(16, 8 \times 2)$, and $(8, 4 \times 2)$. The 8-GPU setup supports fine-tuning only up to the 30B parameter model. Additionally, the configurations $(16, 8 \times 2)$ and $(8, 4 \times 2)$ cannot accommodate all tested model sizes because certain model architectures have numbers of layers or attention heads that are not divisible by the specified pipeline or tensor parallelism factors.

REFERENCES

[1] R. Bommasani and P. Liang, "Reflections on foundation models," https://crfm.stanford.edu/2021/10/18/reflections.html, 2021, accessed: 2023-12-11.

[2] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, "Language models are few-shot learners," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 33, 2020, pp. 1877–1901.

[3] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2016, pp. 785–794.

[4] N. Corporation, "Nvidia h100 tensor core gpu architecture," https://www.nvidia.com/en-us/data-center/h100/, 2022, accessed: 2024-04-20.

[5] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*, 2019, pp. 4171–4186.

[6] N. Ding, Y. Qin, G. Xu, X. Han, P. Xie, H.-T. Zhang, Z. Yang, J. Liu, J. Li, M. Liu, Y. Sun, and J. Zhou, "Parameter-efficient fine-tuning of large-scale pre-trained language models," *Nature Machine Intelligence*, vol. 5, pp. 220–235, 2023.

[7] A. Dubey, A. Jauhri, A. Pandey, A. Kadian, A. Al-Dahle, A. Letman, A. Mathur, A. Schelten, A. Vaughan, A. Yang *et al.*, "The llama 3 herd of models," *arXiv preprint arXiv:2407.21783*, 2024. [Online]. Available: https://arxiv.org/abs/2407.21783

[8] J. H. Friedman, "Greedy function approximation: a gradient boosting machine," *Annals of statistics*, pp. 1189–1232, 2001.

[9] J. L. Gustafson, *Amdahl's Law*. Boston, MA: Springer US, 2011, pp. 53–60. [Online]. Available: https://doi.org/10.1007/978-0-387-09766-4_77

[10] D. Hernandez, J. Kaplan, T. Henighan, and S. McCandlish, "Scaling laws for transfer," *arXiv preprint arXiv:2102.01293*, 2021.

[11] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, and W. Chen, "Lora: Low-rank adaptation of large language models," in *International Conference on Learning Representations (ICLR)*, 2022.
[12] Y. Huang, Y. Cheng, A. Bapna, O. Firat, D. Chen, M. Chen, H. Lee, J. Ngiam, Q. V. Le, Y. Wu, and Z. Chen, "Gpipe: Efficient training of giant neural networks using pipeline parallelism," *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 32, pp. 103–112, 2019.
[13] J. Kaplan, S. McCandlish, T. Henighan, T. B. Brown, B. Chess, R. Child, S. Gray, A. Radford, J. Wu, and D. Amodei, "Scaling laws for neural language models," *arXiv preprint arXiv:2001.08361*, 2020.
[14] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu, "Lightgbm: A highly efficient gradient boosting decision tree," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 30, 2017, pp. 3146–3154.
[15] Meta AI, "The llama 4 herd: The beginning of a new era of natively multimodal ai innovation," https://ai.meta.com/blog/llama-4-multimodal-intelligence/, 2025.
[16] L. Prokhorenkova, G. Gusev, A. Vorobev, A. V. Dorogush, and A. Gulin, "Catboost: unbiased boosting with categorical features," *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 31, pp. 6638–6648, 2018.
[17] M. Shoeybi, M. Patwary, R. Puri, P. LeGresley, J. Casper, and B. Catanzaro, "Megatron-lm: Training multi-billion parameter language models using model parallelism," in *arXiv preprint arXiv:1909.08053*, 2019.
[18] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright *et al.*, "SciPy 1.0: Fundamental algorithms for scientific computing in python," *Nature Methods*, vol. 17, no. 3, pp. 261–272, 2020.
[19] Y. Wang, Y. Kordi, S. Mishra, A. Liu, N. A. Smith, D. Khashabi, and H. Hajishirzi, "Self-instruct: Aligning language models with self-generated instructions," 2023. [Online]. Available: https://arxiv.org/abs/2212.10560
[20] G. West, *Scale: The Universal Laws of Life and Death in Organisms, Cities and Companies*. New York, NY, USA: Penguin Press, 2017.
[21] Y. Xia, J. Kim, Y. Chen, H. Ye, S. Kundu, C. C. Hao, and N. Talati, "Understanding the performance and estimating the cost of llm fine-tuning," in *2024 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE, 2024, pp. 210–223.
[22] L. Zhang, X. Liu, Z. Li, X. Pan, P. Dong, R. Fan, R. Guo, X. Wang, Q. Luo, S. Shi *et al.*, "Dissecting the runtime performance of the training, fine-tuning, and inference of large language models," *arXiv preprint arXiv:2311.03687*, 2023.
[23] S. Zhang, S. Roller, N. Goyal, M. Artetxe, M. Chen, S. Chen, C. Dewan, M. Diab, X. Li, X. V. Lin *et al.*, "Opt: Open pre-trained transformer language models," *arXiv preprint arXiv:2205.01068*, 2022.