computer sciences and mathematics forum

MDPI

Proceeding Paper

Overview of Training LLMs on One Single GPU †

Mohamed Ben jouad * and Lotfi Elaachak

C3S Laboratory, Data and Intelligent Systems Team FSTT, Abdelmalek Essaadi University, Tetouan 93000, Morocco; lelaachak@uae.ac.ma

* Correspondence: benjouad20@gmail.com

† Presented at the International Conference on Sustainable Computing and Green Technologies (SCGT'2025), Larache, Morocco, 14–15 May 2025.

Abstract

Large language models (LLMs) are developing at a rapid pace, which has made it necessary to better understand how they train, especially when faced with resource limitations. This paper examines in detail how various state-of-the-art LLMs train on a single Graphical Processing Unit (GPU), paying close attention to crucial elements like throughput, memory utilization and training time. We find important trade-offs between model size, batch size and computational efficiency through empirical evaluation, offering practical advice for streamlining fine-tuning processes in the face of hardware constraints.

Keywords: LLM; GPU; finetuning

1. Introduction

The advent of large language models has revolutionized natural language processing, enabling breakthroughs in tasks like text generation, translation, question answering, etc. However, training such models can be computationally costly; many models with billions and trillions of parameters were trained with a massive number of GPU clusters, which requires the use of a lot of power resources (DeepSeek R1 [1] took 2048 H800 GPU). While much effort has been devoted to scaling LLMs to record-breaking sizes, there still remains a critical need to optimize their training performance so that, at the very least, the necessary modules can fit into the GPU's memory without saturating it, which is a challenge that the majority of students, educators and researchers face as they tend to rely on free hardware provided from cloud platforms such as Kaggle and Google Colab.

This work addresses this gap by presenting a systematic report of finetuning performance of a few LLMs on two specific free-to-use GPUs: a P100 GPU provided by the Kaggle platform and a T4 from the Google Collab platform, also available on Kaggle. We have focused on special key points such as training time, memory, throughput and convergence behavior so that we can provide a perspective that covers different architectures and setups. By analyzing these measurements, our aim is to produce a list of recommendations that can help individuals to develop trade-offs between model size, effective batch size and efficiency in terms of computation, offering actionable information for optimizing training workflows within hardware and time limits.

Through empirical investigation and visualization, we highlight the impact of model design choice on training efficiency and provide actionable advice to researchers and practitioners. Our work contributes to the overall task of opening up LLM training to make it more accessible, cheap and sustainable, paving the way for innovation in resource-scarce contexts. In the next section, we introduce the different optimization techniques used in the

Academic Editors: Hicham Gibet Tani, Mohamed Kouissi, Mohamed Ben Ahmed and Anouar Abdelhakim Boudhir

Published: 9 July 2025

Citation: Ben jouad, M.; Elaachak, L. Overview of Training LLMs on One Single GPU. *Comput. Sci. Math. Forum* **2025**, *10*, 14. https://doi.org/10.3390/cmsf2025010014

fine-tuning process and the technical overview of the used GPUs, which are freely available on Kaggle and Google Collab.

Our work is motivated by the growing need to democratize access to LLM developments so that advances in NLP are not limited to organizations with easy access to computational power.

2. Related Works

Recent advances in LLMs, such as GPT [2], BERT [3] and T5 [4], have underscored the requirement for cost-effective training methods; techniques like mixed-precision training [5], gradient checkpointing [6] and dynamic batching [7] have been proposed to reduce memory usage and maximize computational effectiveness.

This interaction between hardware capability and model architecture has also been an important research focus. Naryanan and Shoeybi have investigated LLM optimization for multi-GPU and multi-node setups, with a consideration of parallelism and communication overhead. Out study diverts attention to single-GPU systems that are affordable for most people but still impose distinctive challenges related to memory and computation. When fine-tuning models on limited hardware, techniques such as knowledge distillation [8] and model pruning [9] are widely applied to minimize the size as well as the memory need for the model. Besides this, software such as Hugging Face's Transformers [10] and Pytorch [11] have introduced features that help facilitate successful training on mainstream GPUs. However, these approaches are likely to focus on inference rather than training.

A variety of benchmarks like MLPerf [12] and Suite [13] have produced insightful results regarding the performance of machine learning models on different hardware configurations; nonetheless, such analyses focus on large-scale systems and hence provide little attention to single-GPU performance. Our study contributes to the existing literature by investigating how much we can efficiently fine-tune an LLM using only one free-to-use GPUs.

3. Methodology

In this study, we evaluate the training performance of multiple LLMs on one GPU using a combination of advanced optimization techniques and profiling tools; our methodology leverages the Transformer Reinforcement Learning (trl) library for fine-tuning and the Pytorch profiler (version 2.6.0+cu124) to collect detailed performance metrics. The experiments were conducted on two GPU architectures: the NVIDIA P100 manufactured by TSMC in Taichung Taiwan (Pascal Architecture) and the NVIDIA T4 alsomanufactured by TSMC in Taichung Taiwan (Turing Architecture), both available to use on Kaggle and Google Collab, and both of which use hardware that offers high memory bandwidth and computational throughput, with 16 GB of VRAM for the P100 and 15GB of VRAM for T4 [14].

3.1. Model Fine-Tuning and Optimization Techniques

The selected models for the finetuning process are the following: DeepSeek Qwen 1.5 B, Gemini (770 M), Flan-T5-base (248 M), Instella 3B and Qwen2.5-1.5B-Instruct. To optimize training efficiency, we employed several state-of-the-art techniques:

1. Parameter-Efficient Fine-Tuning (PEFT): This includes methods such as LoRA (Low-Rank Adaptation) [15] which is used to reduce considerably the number of trainable parameters making memory less loaded for training when adapting Large Language models to downstream tasks;
2. 8-bit Adam Optimizer [16]: This is utilized to reduce memory usage during optimization while maintaining training stability and convergence;

3. Mixed-Precision Training: We employed both FP16 and BF16 [5] to accelerate computation and reduce memory overhead;
4. Gradient Checkpointing: This was implemented to trade-off computation memory, enabling the training of larger models within limited GPU memory;
5. Effective Batch Size: The batch size was set to powers of two (2^N) to align with hardware optimizations and ensure efficient memory utilization [17].

3.2. Dataset and Training Configuration

We used the CIDAR dataset [18] containing Arabic instructions mapped to their corresponding answer, which was collected using translated instructions from the ALPAGASUS dataset [19] and carefully examined and analyzed. The dataset has around 10.000 instructions, averaging around 366 tokens per instruction, with the longest one having 10,177 tokens. Using an instruction-based dataset can allow for practical uses, especially when implementing an AI agent where interactions with users are important. The models were fine-tuned using the TRL library provided by Hugging Face, as it is more compatible with Pytorch and can facilitate the use of the above-mentioned optimization techniques.

3.3. Collected Metrics

We mainly focused on the following metrics to evaluate and analyze fine-tuning performance for each model:

1. Peak GPU Usage: The maximum GPU memory utilization value during training measured in MB;
2. Training Time: Total time taken to complete the fine-tuning process;
3. Iteration Speed: The number of iterations completed per second, reflecting computational throughput;
4. Training Loss: The final loss value, monitoring convergence and model performance.

With the use of all the mentioned techniques and tools, we can provide a comprehensive framework that will enable us to evaluate LLM training performance throughout multiple configurations.

In the following sections, we present insights into the trade-offs between model size, optimization strategies and memory.

4. Results

In this section, we will present the different observation throughout all fine-tuning processes, focusing mainly on GPU memory consumption, time consumption and convergence to obtain an idea of the difference between each configuration and obtain a good understanding of the recommended trade-offs when aiming to work with LLMs with only free resources.

4.1. Memory Consumption

The following graph represents the memory utilization in MB for each effective batch when using either mixed precision FP16 or BF16, which has been collected using the Pytorch profiler.

We can make multiple observation regarding these data. First, we can see that for Gemini and Flan-T5, we could go up to 64 effective batch size since their parameter number is below 1B, which when reached we can start seeing memory allocation difficulties; that is also the case for DeepSeek, Qwen and Instella, where we see a limit that we cannot overcome. We can conclude that a 3B parameter is the highest that we can go when fine-tuning a model. Another key observation is the influence of the model's architecture on the GPU's consumption. We can see that Qwen reached its limit before Instella, even though

the latter has higher parameters, making the choice of the model's architecture also an important point that will influence the memory needed for fine-tuning. Moreover we can clearly conclude, based on the graphs, the memory difference between FP16 and BF16, with FP16 consuming less memory across all effective batch sizes, with one exeption for DeepSeek when using an effective batch size of 2. We could not explain this discontinuity but we can assume that for most cases, FP16 is the best option for memory efficient training.

4.2. Time Consumption

The following graphs (Figures 1–3) represent the time taken for a fine-tuning process to finish, giving us an idea of how feasibly a model can be trained within a prefered time interval. Knowing that we fine-tuned using 10,000 mapped instructions, we can approximate the time that it will take to for a larger dataset, enabling us to plan for a training more efficently.

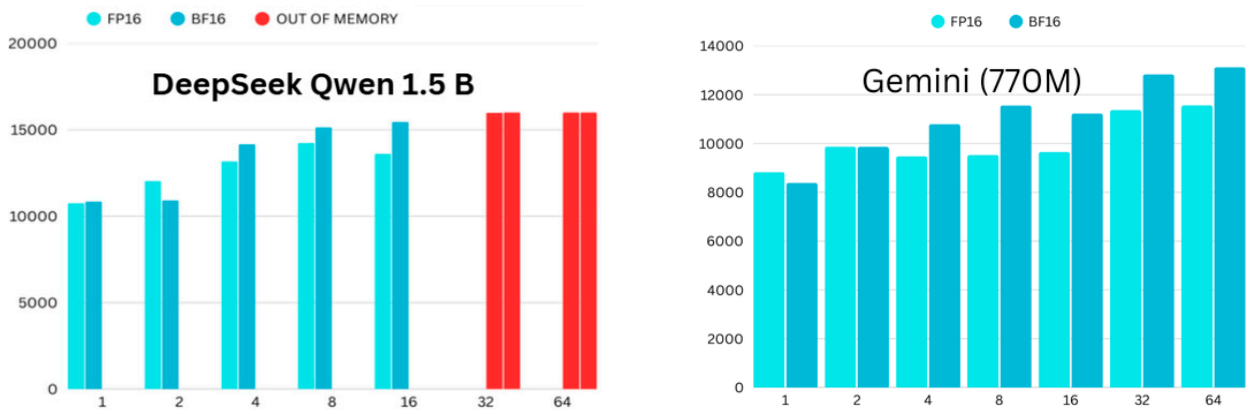

Figure 1. Max memory allocation in MB for the effective sizes going from 1 to 64 for DeepSeek and Gemini.

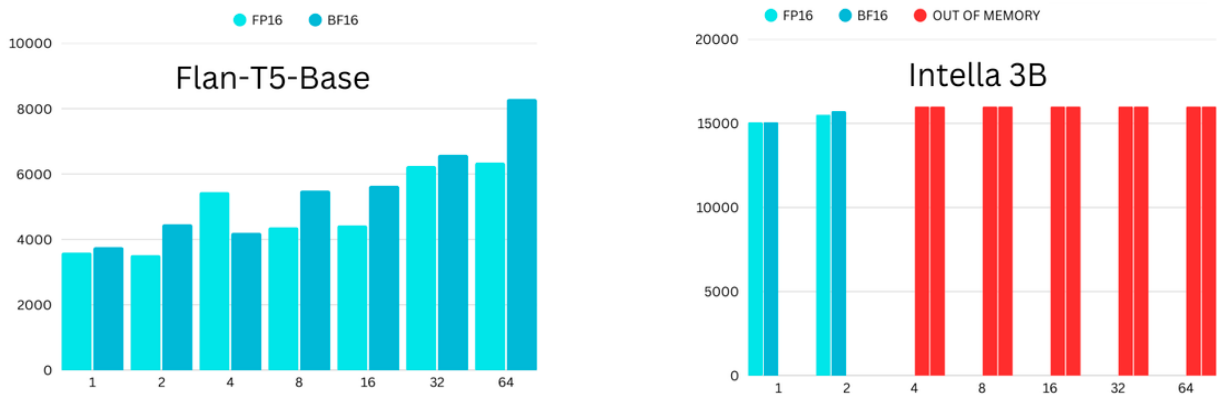

Figure 2. Max memory allocation in MB for the effective sizes going from 1 to 64 for Flan-T5-base and Instella.

The first important observation from Figures 4–6 is the efficency of the FP16 mixed precision, since throughout all the fine-tuning processes, it was considerably quicker across all tests, except for Flan-T5 when using an effective batch size of 4. We do not know the cause of that expection but we can safely assume that for quicker training, FP16 is the best choice. Also, the reason for the graph reaching zero for the model above 1B parameters is because of memory capacity reaching the maximum GPU limit. The precise time it took for each model is displayed in the following table.

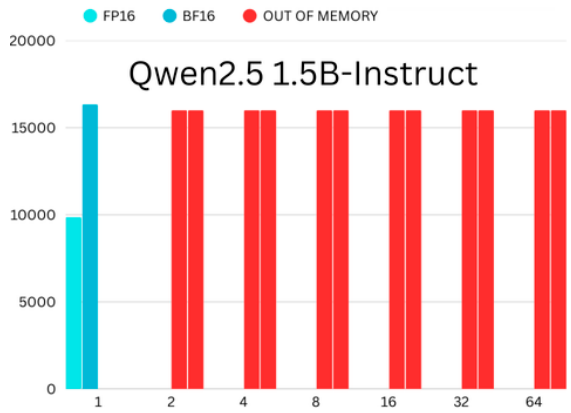

Figure 3. Max memory allocation in MB for the effective sizes going from 1 to 64 for Qwen.

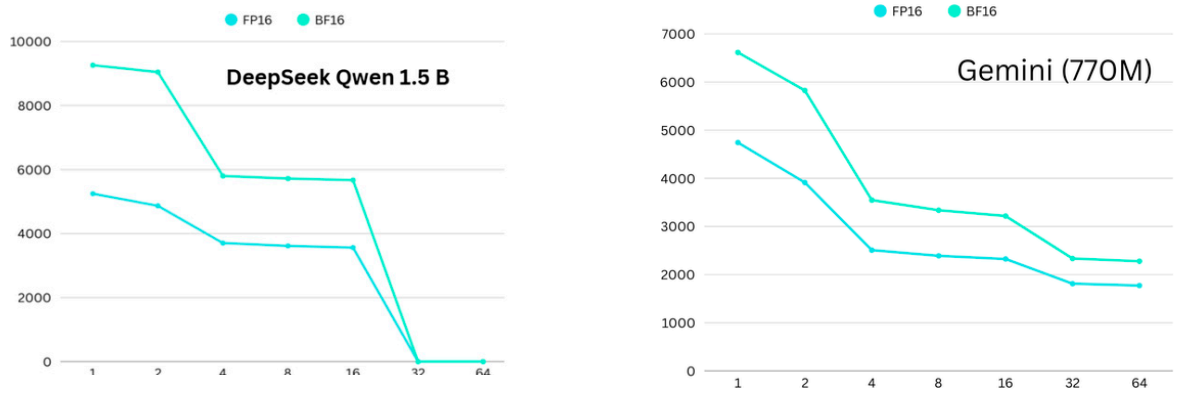

Figure 4. Time consumption in seconds for DeepSeek and Gemini.

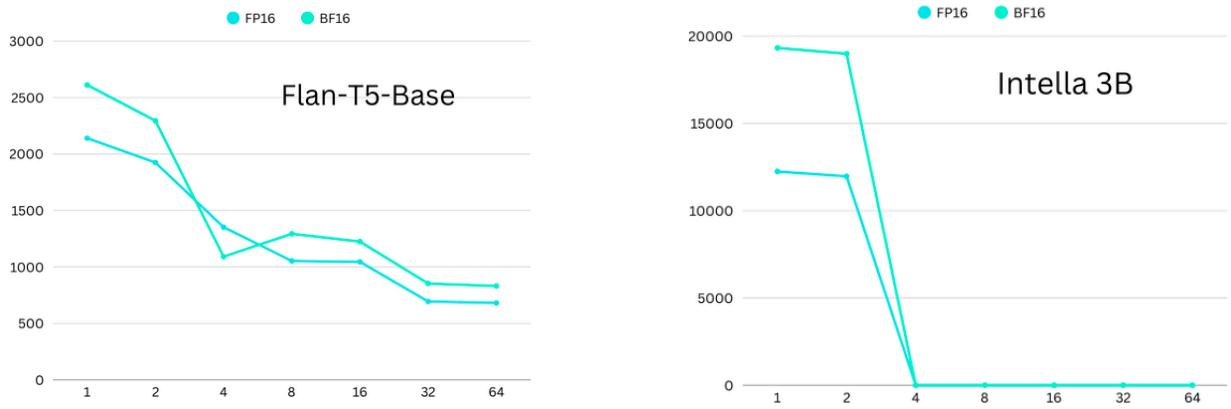

Figure 5. Time consumption in seconds for Flan-T5 and Instella.

The highest and lowest time records shown in Table 1 are reached, respectively, using the highest and lowest effective batch sizes.

Table 1. Precise time consumption for each model, indicating lowest and highest time record.

Model	DeepSeek	Gemini	Flan-T5	Instella	Qwen
Highest (FP16)	1 h 27 m 32 s	1 h 19 m 04 s	35 m 40 s	3 h 24 m 09 s	1 h 24 m 43 s
Highest (BF16)	2 h 34 m 19 s	1 h 50 m 15 s	43 m 32 s	5 h 22 m 10 s	2 h 37 m 03 s
Lowest (FP16)	59 m 22 s	29 m 33 s	11 m 21 s	3 h 19 m 31 s	1 h 24 m 43 s
Lowest (BF16)	1 h 34 m 30 s	37 m 58 s	13 m 51 s	5 h 16 m 39 s	2 h 37 m 03 s

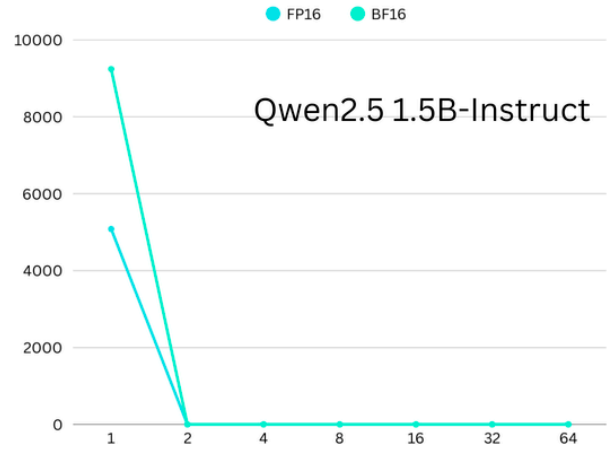

Figure 6. Time consumption in seconds for Qwen.

4.3. Convergence Differences

This section shows the convergence of each model on each configuration by observing the evolution of the loss across all the steps of a fine-tuning process; this can show us how accurate the training becomes for the dataset used. It is worth mentioning that the reason we did not specify the mixed precision used for the configuration combination is because the results obtained for both FP16 and BF16 are nearly the same, making these two parameters irrelevant in terms of convergence.

The first thing we notice from Table 2 is the increase in the loss relative to higher batch size, meaning that even though we want rapid training, we still need to take into consideration the convergence that struggles to find an optimum for that batch size. In other words, it is important to choose the effective batch size depending on the time constraint and the accuracy desired. Another observation is the irregularity of the loss for both Gemini and Flan-T5. This is probably because they do not have the vocabulary of the Arabic words used in the datasets. As for Instella and Qwen, there is not much data since the fine-tuning stops early.

Table 2. Loss difference between the multiple configurations used for fine-tuning.

Model	DeepSeek	Gemini	Flan-T5	Instella	Qwen
Effective size 1	2.5252	0.009	0.0063	1.0631	1.6211
Effective size 2	5.3273	0.0119	0.0721	1.0945	None
Effective size 4	5.5068	0.161	0.0116	None	None
Effective size 8	11.7354	0.0256	0.0188	None	None
Effective size 16	24.3662	0.0376	0.0257	None	None
Effective size 32	None	0.0624	0.0329	None	None
Effective size 64	None	0.128	0.0749	None	None

5. Discussion

The results of our analysis provide valuable insights into the possible trade-off when fine-tuning LLMs on one single GPU. Let us summarize the key observations and their implications for optimizing training workflows.

Our findings indicate that the optimum model size for single GPU training lies at around 3B parameters, since we observed difficulties when working with models of these sizes. Memory becomes hard to manage, even with all the optimization techniques we used, and prior work on memory-efficient training [20] claims the same thing, which highlights the challenges of scaling LLMs on limited hardware. Another key observation is the efficiency of the mixed precision FP16, which consumes less memory and requires less time than BF16 while still maintaning the same convergence as the latter, making

FP16 the best choice for training models in resource-constrained systems. Moreover, we do see that the model's architecture influences the memory, with Instella going up to two effective batch sizes and Qwen only going up to one, meaning that choosing the architecture carefully might lead to better memory management. In terms of convergence behavior, on all fine-tuning steps, the loss remained the same when using either FP16 or BF16, which means that mixed precision does not impact on the final model's accuracy. However, we do see an increase in the loss the higher the effective batch size, which can be attributed to the reduced gradient variance associated with larger batch sizes, leading to lower convergence and high loss values during early stages of training [21]. While larger batches improve throughput and hardware utilization, they may require careful tuning of learning rates and optimization schedules to maintain model performance.

When planning for a training workflow, we suggest carefully considering the above-mentioned observations by finding the best trade-offs that will work best on specific use cases. This will save time and prevent memory problems from arising when running the training script. A lot of individuals find it difficult, even impossible, to train LLMs with their own setup or by using free online resources and we want to show that it is not the case and encourage innovation no matter the situation and resources used. Moreover, in future work, we want to provide additional data and more techniques that will prove to be useful when training models on limites resources, so that it does remain an obstacle to innovation.

6. Conclusions

In this study, we conducted a evaluation of the training performance of multiple large language models using single GPU systems, focusing on memory efficiency, training time, convergence bhavior and the impact of batch sizes and mixed precision. By leveraging state-of-the-art optimization techniques such as mixed-precision training (FP16 and BF16, Adam optimization, gradient checkpointing and Parameter efficient Fine-Tuning), we demonstrated that it is feasible to train models with up to 3B parameters on publicly available GPUs.

Our study also revealed that FP16 offers significant memory saving and faster training times compared to BF16, while still maintaining the same convergence behavior across on fine-tuning steps, making it a practical choice for resource-constrained environments. Also, we observed that convergence behavior becomes slower the high the batch size is, making it important to carefully tune the training parameters in order to achieve the desired accuracy.

Author Contributions: Conceptualization, M.B.j. and L.E.; Methodology, M.B.j. and L.E.; Software M.B.j.; Validation L.E.; Formal analysis M.B.j.; Investigation M.B.j.; Data curation M.B.j.; Writing—original draft, M.B.j.; Writing—review & editing M.B.j. and L.E. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Data are available in this manuscript.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. DeepSeek-AI. DeepSeek-V3 Technical Report. *arXiv* **2025**, arXiv:2412.19437.
2. Radford, A.; Narasimhan, K.; Salimans, T.; Sutskever, I. Improving Language Understanding by Generative Pre-Training. 2018. Available online: https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf (accessed on 5 January 2025).

3. Devlin, J.; Chang, M.-W.; Lee, K.; Toutanova, K. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *arXiv* **2019**, arXiv:1810.04805.
4. Raffel, C.; Shazeer, N.; Roberts, A.; Lee, K.; Narang, S.; Matena, M.; Zhou, Y.; Li, W.; Liu, P.J. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. *arXiv* **2020**, arXiv:1910.10683.
5. Micikevicius, P.; Narang, S.; Alben, J.; Diamos, G.; Elsen, E.; Garcia, D.; Ginsburg, B.; Houston, M.; Kuchaiev, O.; Venkatesh, G.; et al. Mixed Precision Training. *arXiv* **2018**, arXiv:1710.03740.
6. Chen, T.; Xu, B.; Zhang, C.; Guestrin, C. Training Deep Nets with Sublinear Memory Cost. *arXiv* **2016**, arXiv:1604.06174.
7. Wu, Y.; Schuster, M.; Chen, Z.; Le, Q.V.; Norouzi, M.; Macherey, W.; Krikun, M.; Cao, Y.; Gao, Q.; Macherey, K.; et al. Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation. *arXiv* **2016**, arXiv:1609.08144.
8. Hinton, G.; Vinyals, O.; Dean, J. Distilling the Knowledge in a Neural Network. *arXiv* **2015**, arXiv:1503.02531.
9. Han, S.; Pool, J.; Tran, J.; Dally, W.J. Learning both Weights and Connections for Efficient Neural Networks. *arXiv* **2015**, arXiv:1506.02626.
10. Wolf, T.; Debut, L.; Sanh, V.; Chaumond, J.; Delangue, C.; Moi, A.; Cistac, P.; Rault, T.; Louf, R.; Funtowicz, M.; et al. Hugging Face's Transformers: State-of-the-Art Natural Language Processing. *arXiv* **2020**, arXiv:1910.03771.
11. Paszke, A.; Gross, S.; Massa, F.; Lerer, A.; Bradbury, J.; Chanan, G.; Killeen, T.; Lin, Z.; Gimelshein, N.; Antiga, L.; et al. PyTorch: An Imperative Style, High-Performance Deep Learning Library. *arXiv* **2019**, arXiv:1912.01703.
12. Reddi, V.J.; Cheng, C.; Kanter, D.; Mattson, P.; Schmuelling, G.; Wu, C.-J.; Anderson, B.; Breughe, M.; Charlebois, M.; Chou, W.; et al. MLPerf Inferrence Bechmark. *arXiv* **2020**, arXiv:1911.02549.
13. Coleman, C.; Narayanan, D.; Kang, D.; Zhao, T.; Zhang, J.; Nardi, L.; Bailis, P.; Olukotun, K.; Ré, C.; Zaharia, M. DAWNBench: SMASH: One-shot Model Architecture Search through HYperNetworks. *arXiv* **2017**, arXiv:1708.05344.
14. NVIDIA. NVIDIA T4 GPU Architecture. 2020. Available online: https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/tesla-t4/t4-tensor-core-datasheet-951643.pdf (accessed on 10 January 2025).
15. Hu, E.J.; Shen, Y.; Wallis, P.; Allen-Zhu, Z.; Li, Y.; Wang, S.; Chen, W. LoRA: Low-Rank Adaptation of Large Language Models. *arXiv* **2021**, arXiv:2106.09685.
16. Dettmers, T.; Lewis, M.; Belkada, Y.; Zettlemoyer, L. 8-bit Adam Optimizer Via Block-Wise Quantization. *arXiv* **2022**, arXiv:2110.02861.
17. NVIDIA. 2023. Available online: https://docs.nvidia.com/deeplearning/performance/index.html (accessed on 15 January 2025).
18. Alyafeai, Z.; Almubarak, K.; Ashraf, A.; Alnuhait, D.; Alshahrani, S.; Abdulrahman, G.A.Q.; Ahmed, G.; Gawah, Q.; Saleh, Z.; Ghaleb, M.; et al. CIDAR: Culturally Relevant Instruction Dataset for Arabic. *arXiv* **2024**, arXiv:2402.03177.
19. Chen, L.; Li, S.; Yan, J.; Wang, H.; Gunaratna, K.; Yadav, V.; Tang, Z.; Srinivasan, V.; Zhou, T.; Huang, H.; et al. ALPAGASUS: Training a better ALPACA with fewer data. *arXiv* **2024**, arXiv:2307.08701.
20. Narayanan, D.; Shoeybi, M.; Casper, J.; LeGresley, P.; Patwary, M.; Korthikanti, V.; Vainbrand, D.; Kashinkunti, P.; Bernauer, J.; Catanzaro, B.; et al. Efficient Large-Scale Language Model Training on GPU Clusters Using Megatron-LM. *arXiv* **2021**, arXiv:2104.04473.
21. Hoffmann, J.; Borgeaud, S.; Mensch, A.; Buchatskaya, E.; Cai, T.; Rutherford, E.; de Las Casas, D.; Hendricks, L.A.; Welbl, J.; Clark, A.; et al. Training Compute-Optimal Large Language Models. *arXiv* **2022**, arXiv:2203.15556.