# CS 7643: Advancing Question Answering with Decoder-Only Causal Language Models: Efficient Fine-Tuning and Inference Techniques for Large-Scale QA

Andrey Shor
ashor6@gatech.edu

Micaela McCall
mmccall73@gatech.edu

Artem Maryanskyy
a.maryanskyy@gatech.edu

## Abstract

*Large Language Models (LLMs) represent the forefront of current natural language processing technologies, capable of undertaking a vast array of tasks from text generation to complex reasoning. While these models are generally pre-trained on extensive, task-agnostic datasets, adapting them to specific tasks remains a significant challenge due to the computational demands of traditional fine-tuning processes. This study investigates several efficient fine-tuning methodologies applied to three publicly available pre-trained LLMs, aiming to evaluate and compare their effectiveness and efficiency. Through a systematic comparison of these methods, we assess the speed and performance of each, providing insights into the most effective strategies for task-specific model adaptation. Our results indicate that certain Parameter-Efficient Fine-Tuning (PEFT) methods, particularly QLoRA and Adalora, show promise in enhancing the question-answering capabilities of LLMs with reduced computational overhead.*

## 1. Background

Large language models (LLMs), large-scale deep learning models using a transformer architecture, are the current state-of-the-art for processing and understanding written language. Trained on massive amounts of text-based data, these models can be used to perform a number of natural language processing (NLP) tasks, such as text generation, summarization of long texts, translation, and even complex reasoning tasks [26]. Because these models are so versatile, it has become customary to pre-train LLMs on general, task-agnostic data and publish them for the use of multiple downstream tasks.

In this project, we experimented with a variety of approaches to adapting pre-trained LLMs for task-specific uses. While the use of pre-trained LLMs has become standard in the development of various NLP applications, there are existing challenges in the adaptation process. This process, called fine-tuning, can be extremely computationally heavy when performed as a standard training process, i.e., requiring the updating of billions of model parameters. Much research and experimentation is being focused on the problem of how to increase efficiency of the fine-tuning process as the demand for and use of pre-trained LLMs grows. Given the abundance of new and creative approaches in this area, there is a need for thorough and specific direct comparisons in order to discover the optimal combination of methods for the task-specific adaptation of pre-trained LLMs. In this project, we provide some of these direct comparisons by employing a selection of efficient fine-tuning methods on three publicly available pre-trained LLMs. By comparing the speed of each method selected and the performance of the resultant fine-tuned models, we aim to contribute to the search for optimal fine-tuning approaches.

As mentioned, there are many new and successful approaches to this problem, e.g. Z. Wan et al. categorizes LLM efficiency techniques into model-centric, data-centric, and framework-centric [26]. While current practice is to use one of these existing methods, the resource demand is still the most considerable cost of using LLMs. This limits use of these extremely performant models to those entities with sufficient financial resources, and drives attempts to find more cost-effective and environmentally friendly approaches. Our project is situated within this context, and a successful contribution to this area of research would help the field meet the growing demands for quicker inference and training, as well as accessibility edge devices with less computing power.

In this project, we employ a selection of efficiency approaches (described below in the Methods) to fine-tune each of three pre-trained base LLMs: Llama [32] , Meta [28], and Mistral [22]. We then compare the performance and inference speed of the resultant fine-tuned models. All fine-tuning experiments were done using the UnifiedQA [23] dataset, which is described below.

### 1.1. Data

UnifiedQA is a dataset created by the Allen Institute for AI, consisting of a collection of different question-answering datasets that span four formats: yes/no, multiple choice, extractive (the question is a paragraph and the answer is a substring from the paragraph), and abstractive (the question is a paragraph and the answer is something about that paragraph but is more than just a substring) [23]. Allen Institute for AI argues for the utility of training question-answering models on questions with a variety of formats since models should understand the underlying reasoning of a question and answer [23]. The UnifiedQA datasets are provided in a text-in / text-out framework, where instances are questions (encoded as text) and labels are answers (encoded as text) [23]. We perform fine-tuning using 11 datasets from UnifiedQA, which together have 462,516 instances. See supplementary for a description of each dataset as well as the number of samples in train, dev, and test splits.

## 2. Approach

In this project, we perform experiments by applying efficient fine-tuning methods on three publicly available pre-trained LLMs and comparing the speed and performance of these methods and the trained models produced.

### 2.1. Models

The pre-trained models we use are all available on HuggingFace. We use the 7 Billion parameter version of Llama (available at https://huggingface.co/meta-llama/Llama-2-7b), the 2 Billion parameter version of Gemma (https://huggingface.co/google/gemma-2b), the 7 Billion parameter version of Gemma (available at https://huggingface.co/google/gemma-7b), and the 7 Billion parameter version of Mistral (https://huggingface.co/mistralai/Mistral-7B-v0.1).

All three models are pre-trained generative LLMs that use a decoder-only transformer architecture. In comparison to encoder-decoder architecture for question-answering, in which a model explicitly learns about the dependencies between tokens in the question, via the encoder, in order to produce answers, via the decoder, decoder-only architectures generate output token-by-token by attending only to positions prior to the position of the current token [**?**]. Even though these decoder-only LLMs generate text simply as next-word prediction, they display impressive emergent properties that allow them to be applied to many types of tasks out-of-the-box (see In-context learning below), and with fine-tuning can learn to answer questions in the format desired [32]. Efficient Fine-Tuning and Inference Metho

- The optimization was ADAM 8bit, the learning rate was 2e-5, with a linear learning rate scheduler that increases from 0 to 2e-5 over 30 warm-up steps and then decreases back to 0 [3].
- The max number of steps is 200 [4].
- The per-device train and eval batch sizes are both 8 and 4 [4].

### 2.2. Efficient Fine-Tuning and Inference Methods

We performed experiments with the selection of efficiency methods described below. All of the mentioned methods are existing methods that have been described in academic publications and have been proven to result in performant models [26]. While none of the approaches we tried were novel in that sense, we believed that comparing these approaches across different models would result in a successful exploration of efficient fine-tuning approaches.

Many of these approaches can be considered under the umbrella of Parameter Efficient Fine Tuning Methods (PEFT), which improve efficiency by reducing the number of model parameters that must be updated during gradient-descent-based fine-tuning, and thereby reduce requisite memory and computation [28].

1. Low Rank Adaptation (LoRA) introduces trainable decomposition matrices into the model and effectively performs low-rank decomposition on model parameter matrices while preserving essential information [21].

2. Quantization LoRA (QLoRA) is similar but additionally quantizes the weights of the pre-trained network, meaning it converts weights from high-precision data types to low-precision data types (such as from float-32 to int-8), which compresses the model [21]. This approach can also be applied on its own to reduce required compute and memory after training [26].

3. Adaptive LorRA (AdaLoRA) addresses one shortcoming of LoRA of assuming a constant importance of each weight matrix across layers by pre-specifying the rank of the decomposition matrix [24]. AdaLoRA adaptively assigns more or fewer rank (more or fewer parameters) to matrices with more or less importance [24].

4. Adapter-based Tuning takes the approach of adding adapter modules (new, randomly-initialized layers) to the transformer-based architecture and updating the adapter weights while keeping the pre-trained weights frozen [21]. While there are a myriad of ways to do this, we experiment with an approach called Infused Adapter by Inhibiting and Amplifying Inner Activations (IA)3, which introduces a set of learned vectors to rescale the model's activations, namely the keys and values in attention layers and inner products of feed-forward layers [25].

5. Prompt-tuning reduces trainable parameters by adding discrete (or manually written) prompt tokens to the input and performing fine-tuning by updating the parameters associated with those prompts [33]. There are various ways to select prompts, and the exploration of these approaches is referred to as prompt tuning. One

| Model | Question Tag | Answer Tag | Other Tags | Resource |
|---|---|---|---|---|
| Gemma | ‘Prepend “¡start_of_turn¿user” and append ”¡end_of_turn¿”‘ | Prepend ”¡start_of_turn¿model” and append ”¡end_of_turn¿” | Prepend ”¡bos¿” to question | https://huggingface.co/goo 7b/discussions/62 |
| Llama | Prepend ”Input:” | Prepend ”Output:” | Prepend ”¡s¿” to question | https://huggingface.co/docs neuron/en/tutorials/fine_tun or https://github.com/mallorb |
| Mistral | Prepend ”[INST] ” and append ” [/INST]” | N/A | Prepend ”¡s¿” to question | https://www.promptinggui 7b |

Table 1. Data Preprocessing.

| Approach | Hyperparameter | Description |
|---|---|---|
| LoRA, QLoRa, AdaLoRA | r | Rank; the size of the low-rank matrices used to decompose the weight matrix. |
| | alpha | Scaling factor applied to the learned decomposition matrices. |
| | bias | Whether “none”, “all”, or only the LoRA bias parameters should be trained. |
| | dropout | The dropout probability for LoRA layers. |
| AdaLoRa | target_r | The target average rank of incremental matrix. |
| | init_r | The initial rank for each incremental matrix. |
| | tinit | The steps of initial fine-tuning warmup. |
| | tfinal | The step of final fine-tuning. |
| | deltaT | The time internval between two budget allocations. |
| QLoRA Quantization | load_in_4bit | Whether to quantize the model to 4-bits when you load it set [1]. |
| | bnb_4bit_compute_dtype | The computational type which might be different than the input time. Use torch.bfloat16 for speedups [1]. |
| | bnb_4bit_use_double_quant | Whether to use a nested quantization scheme to quantize the already quantized weights set [1]. |
| (IA)$^3$ | target_modules | The layers to apply rescaling to [12,17]. |
| Prompt-tuning | prompt_tuning_init | The initialization of the prompt embedding [2]. |
| | prompt_tuning_init_text | The text to initialize the prompt embedding [2]. |
| | tokenizer_name_or_path | The name or path of the tokenizer. Applied to the prompt_tuning_init_text [2]. |

Table 2. Parameter Efficient Fine-Tuning Methods Hyperparameters

of these approaches, called chain-of-thought prompting, involves adding a prompt to the input asking the model to explain its step-by-step reasoning, thereby guiding the model to decompose the underlying reasoning into its constituent steps [27].

Another approach to efficient inference is In-Context Learning (ICL). ICL methods bypass gradient-descent-based fine-tuning entirely by leveraging the emergent property of LLMs that they are able to perform on previously unseen tasks after seeing some examples of that task [29]. Depending on how many (training) examples of the new task are fed to the model, the approach may be referred to as Zero-Shot, One-Shot, or Few-Shot ICL. The implementation of ICL also involves inserting prompts into the input. For example, in the zero-shot case, we simply inserted the prompt “Answer the following question.” In the k-shot cases, we inserted the prompt “Answer the following question. Follow these examples:”, followed by the example question-answer pairs in the model-specific chat format.

Finally, the last category of approaches we experiment with enhanced LLM efficiency at the algorithm level [26]. Speculative Decoding is an algorithm that speeds up sampling from LLMs by computing tokens in parallel using several smaller, approximation models that create a number of guesses at prefixes which are then evaluated by the larger model and chosen if they don’t change the target distribution [20]. We implement speculative decoding by using a 2 billion parameter version of the Gemma model, fine-tuned with QLoRA to create speculative prefixes for the 7 billion parameter Gemma to evaluate directly in the inference/text

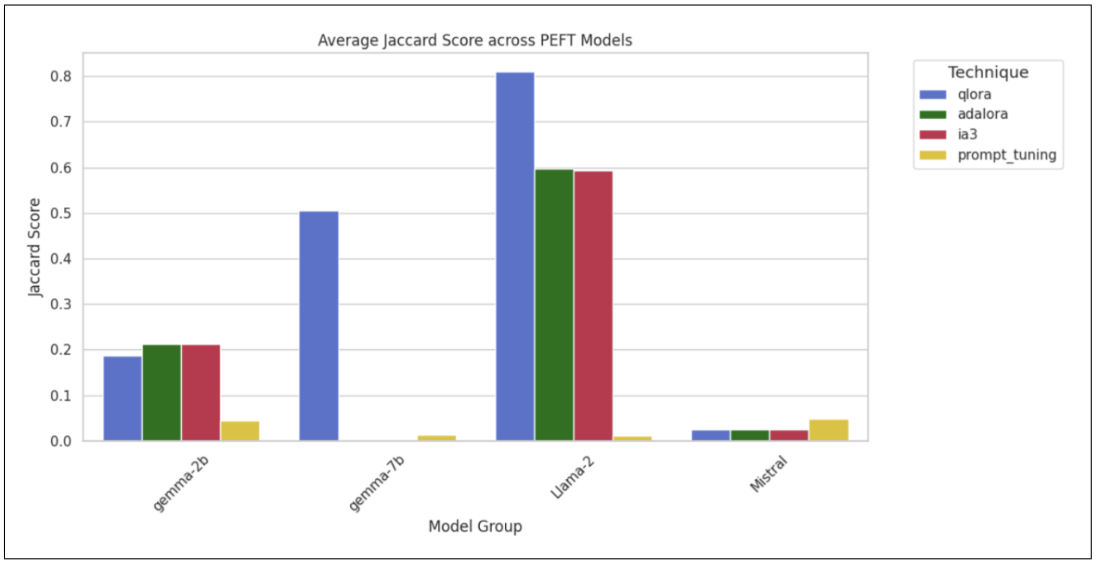

Figure 1. Average Jaccard Distance Across PEFT Models

generation stage.

### 2.3. Data Preprocessing

The raw UnifiedQA datasets were downloaded from Google storage using a script provided by Allen AI [23]. Basic preprocessing was performed to generate lowercase text .json files with 'id', 'question', and 'answer' keys containing lists for each data sample. An additional step of preprocessing was required to prepare the data for each of the three models, since they each expect a different format in terms of start token/ input and output for chatting with the LLM. With this step, we encountered our first challenge since this model-specific pre-processing is not something that is widely documented and therefore was not something we had anticipated. Please refer to Table 1 for the required model-specific data preprocessing as well as a link to the HuggingFace discussions in which we found the information for each model. Finally, the model-specific datasets were tokenized using a HuggingFace- developed procedure that uses the pre-trained tokenizer associated with a pre-trained model hosted on HuggingFace (as were all the models we used).

### 2.4. Training (fine-tuning) and Hyperparameters

After preprocessing and tokenizing the UnifiedQA data, we implemented a Python CLI tool that performed the following steps: Load the specified model (Gemma-2b, Gemma-7b, Llama-7b, and Mistral-7b) and corresponding tokenizer from HuggingFace. Load a configuration for the specified PEFT approach, which specifies required parameters and hyperparameters that vary from approach to approach. Train the model using the PEFT hyperparameters and the general training hyperparameters mentioned below. At a specified number of steps, checkpoint the trained model and save to a specified file location. When training is complete, save the trained model and the training losses at the above-specified number of steps to a specified file location.

See Table 2 for a description of each approach-specific hyperparameter. The combinations of approach-specific hyperparameters used (and resultant performance metrics) are reported in the Experiments and Results section.

We kept the general training hyperparameters constant across all experiments in order to have more time to run various PEFT experiments. These training hyperparameters are as follows, and were selected based on values that are generally good for this type of task [5].

### 2.5. Hardware Requirements

In order to do all the fine-tuning experiments, we used Lambdalabs [6], which is a GPU-as-a-service provider that rents out GPUs. However, we ran into our first roadblock when we tried to create an A100 GPU instance on Lambdalabs and found that they were all unavailable because they were being used by other people (first-come-first-serve policy). We ended up researching another similar service called Hyperstack [7], but then circling back to Lambdalabs.

After the PEFT Training CLI was developed, we moved it to Lambdalabs and implemented four bash scripts to run it. The four experiments we ran with these bash scripts were QLora, Quantized Adalora, Quantized IA3, and Quantized Prompt Tuning. We had to run the models in quantized versions as the hardware requirements we utilized (1 x A100 GPU on Lambda Labs) did not have enough memory to be

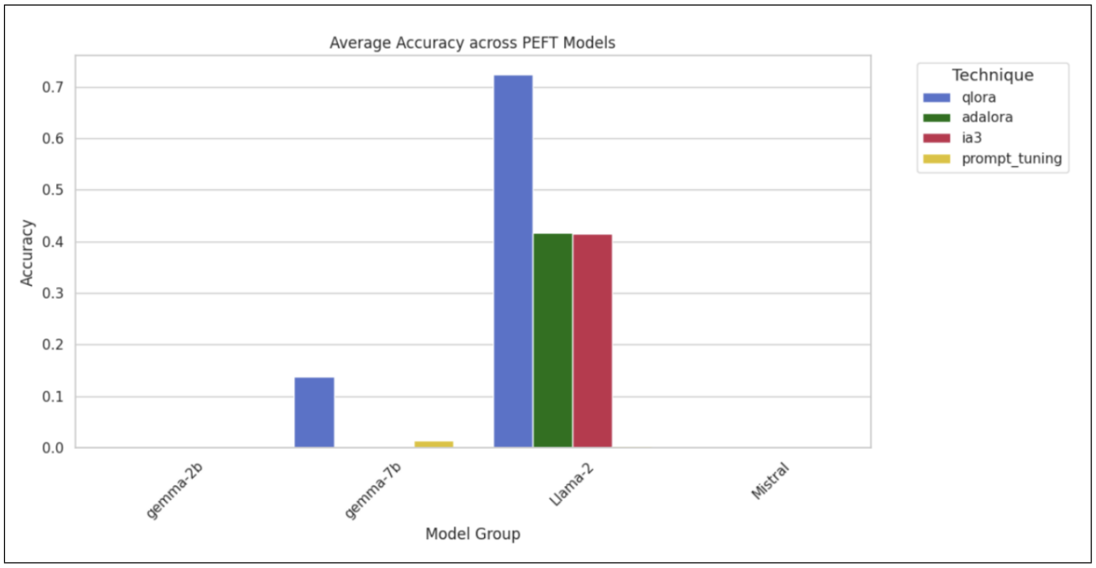

Figure 2. Average Accuracy Across PEFT Models

able to handle the unquantized variants during training. Due to the described hardware constraint, we were unable to run LORA on the A100 GPU for each of the models.

We used the same A100 GPU instance for inference with the fine-tuned PEFT models, as well as for ICL and speculative decoding.

## 3. Experiments and Results

We implemented each PEFT, ICL, and inference-time efficiency method described above and evaluated each one by measuring the performance of the resultant models on test data and the inference-time efficiency while generating predictions. The results of these experiments are recorded in Table 4.

### 3.1. Inference and Evaluation

We evaluated each of our fine-tuned models on the test dataset, which contains samples across the 11 UnifiedQA datasets in supplementary material material. While the total number of test samples was over 30,000, we limited the prediction to 1,000 samples (questions) due to the constraints of the L4 GPU available on Google Colab. This is a statistically representative sample size to get a sense of the quality of the generated predictions. For each sample, the question text, preprocessed to have the model-specific chat tags, was fed to the model and the generated text was collected as the prediction. The answer was then stripped out of that prediction, which was also formatted according to the model-specific chat format, and compared to the ground-truth (human generated) answers to each question. The metrics used to evaluate the quality of the predictions, as well as the inference-time efficiency, are described below.

### 3.2. Metrics

The metrics we used to evaluate the success of each method are accuracy, perplexity, and GPU throughput. Perplexity is the exponentiated average negative log-likelihood of a sequence [8]. The log-likelihood of a sequence is the sum of the likelihoods of each item in the sequence conditioned on the previous items in the sequence. The perplexity basically captures the degree of uncertainty of the model when seeing new data, and the lower the perplexity the better [9]. We calculated perplexity by exponentiating the training loss (cross entropy loss) at the end of training for each fine-tuning method.

Accuracy measures how many correct outputs the model produces [10]. We calculated Jaccard similarity between each answer generated during testing with the ground truth answer. Jaccard similarity is the number of common tokens between the two strings divided by the total length of the longer string [11]. A Jaccard similarity of 1 would indicate a perfect match and thus count as a “correct” output. Because this is a stringent criteria for success in natural language understanding, we also report average Jaccard similarity across the test set for each fine-tuning approach on each model.

GPU Throughput is simply the number of output tokens the model generates per second and is higher for models with higher inference-time efficiency [12]. In order to measure throughput, we recorded the time required for each batch of inference tokens to be produced (latency in seconds) and then divided the number of produced tokens by that latency [13].

| Approach | Test Accuracy | Test Average Jaccard Similarity | GPU Throughput (Tokens per second) | Training Perplexity | Training Cross Entropy Loss |
|---|---|---|---|---|---|
| QLoRA | 0.0 | 0.1878 | 169.6 | 24.6407 | 3.2044 |
| AdaLoRA | 0.0 | 0.2133 | 135.0 | 638.3585 | 6.4589 |
| $(IA)^3$ | 0.0 | 0.2125 | 180.3 | 35.6803 | 3.3529 |
| Prompt tuning | 0.0 | 0.0449 | 190.5 | 35.6803 | 3.5746 |

Table 3. Gemma 2B Results

| Approach | Test Accuracy | Test Average Jaccard Similarity | GPU Throughput (Tokens per second) | Training Perplexity | Training Cross Entropy Loss |
|---|---|---|---|---|---|
| QLoRA | 0.1371 | 0.5059 | 59.4 | 15.6348 | 2.7495 |
| $(IA)^3$ | 0.546 | 0.8265 | 27.2403 | N/A | N/A |
| Prompt tuning | 0.0 | 0.0449 | 190.5 | 245719205.2789 | 19.3197 |

Table 4. Gemma 7B Results

## 3.3. Challenges

The biggest challenge of this project was the hardware required to train these massive models. Our research had only somewhat prepared us for this issue; while we knew we'd need significant compute and time, the reality of training LLMs was much more challenging than expected in terms of the hardware required. This was the limiting factor in our experimentation and prevented us from producing all the results we had hoped for.

## 3.4. Discussion

Our study demonstrates that Parameter-Efficient Fine-Tuning (PEFT) methods applied to Causal Language Models (CLMs) can effectively adapt these models to a range of question-answering (QA) tasks. As depicted in Figures 1 and 2 the QLoRA method outperformed other techniques in both the Llama 2-7b and Gemma-7b models. Notably, Llama2-7b, fine-tuned using QLoRA, excelled in processing the UnifiedQA dataset, achieving a Jaccard Similarity of 80% and an accuracy of 72%. In contrast, Gemma-7b exhibited a respectable Jaccard Similarity of 50% but a considerably lower accuracy of 14%. One final experimentation that was done was utilizing Speculative Decoding for Inference increase. This was done using the Gemma family of models. We used the base Gemma-7b model as the main model along with the fine-tuned Gemma-2b (QLora) as the assistant model. Our results show that while the throughput of the output decreased to 27.3 tokens/sec, the Jaccard Similarity Score jumped up to 82.3% and the accuracy jumped to 54.6%, resulting in a model that was able to outperform our finetuned Llama2-7b (QLoRA).

The experimentation highlighted several intriguing results and areas of uncertainty regarding PEFT approaches. Across multiple models, prompt tuning resulted in catastrophic forgetting, with the models producing nonsensical outputs, thereby failing to address the questions in the test set. Additionally, all PEFT methods were ineffective with the Mistral-7b model, which consistently generated irrelevant content up to the maximum token limit. Unfortunately, due to a network connectivity issue, we lost the fine-tuned results for the Gemma-7b models using Adalora and IA3, and time constraints prevented the rerun of these experiments. It is also important to note that the reason the throughput for the Mistral-7b was so high was due to the way that throughput was calculated (number of tokens generated / latency). Since Mistral-7b generated the maximum number of tokens in a very short amount of time, the result is what can be found in Figure 3. The exact values of throughput for each model can be found in table 3-6.

Outside of Mistral-7b's throughput, the Gemma-2b model was the next fastest, followed by Gemma-7b and Llama2-7b. It is crucial to highlight that the throughput measurements for these models are somewhat skewed. The prompt-tuning method failed to facilitate effective learning of the UnifiedQA dataset, affecting the performance assessments.

## 3.5. Future Work & Direction

Given more time, our goal was to explore additional PEFT strategies, including In-Context Learning and LoRA, to further our understanding of PEFT's efficacy across different model architectures and tasks. To facilitate future research in this area, we have included the code for both zero-shot and k-shot In-Context Learning, as well as LoRA, in the supplementary materials. These resources are intended to support ongoing investigations into the potential and limitations of PEFT approaches in enhancing model perfor-

| Approach | Test Accuracy | Test Average Jaccard Similarity | GPU Throughput (Tokens per second) | Training Perplexity | Training Cross Entropy Loss |
|---|---|---|---|---|---|
| QLoRA | 0.7249 | 0.8098 | 30.8 | 8.3794 | 2.1258 |
| AdaLoRA | 0.4171 | 0.5973 | 39.8 | 76.3555 | 4.3354 |
| $(IA)^3$ | 0.4146 | 0.4146 | 50.9 | 9.2341 | 2.2229 |
| Prompt tuning | 0.0034 | 0.0115 | 33.7 | 9.5305 | 2.2545 |

Table 5. Llama 7B Results

| Approach | Test Accuracy | Test Average Jaccard Similarity | GPU Throughput (Tokens per second) | Training Perplexity | Training Cross Entropy Loss |
|---|---|---|---|---|---|
| QLoRA | 0.0 | 0.0242 | 2702.3 | 8.2194 | 2.1065 |
| AdaLoRA | 0.0 | 0.0242 | 1582.3 | 134.6663 | 4.9028 |
| $(IA)^3$ | 0.0 | 0.0242 | 1936.1 | 10.1767 | 2.3201 |
| Prompt tuning | 0.0 | 0.0242 | 114.6 | 11.1696 | 2.4132 |

Table 6. Mistral 7B Results

mance on complex QA tasks.

## References

[1] https://huggingface.co/docs/peft/developer/$_{g}uides/quantization$. 3, 8

[2] https://huggingface.co/docs/peft/en/task/$_{g}uides/prompt/_{b}ased/_{m}ethods?configurations = prompt + tuning.$ 3, 8

[3] https://huggingface.co/docs/transformers/main/en/main/$_{c}lasses/optimizer_{s}chedulestransformers.SchedulerType.$ 2, 8

[4] https://huggingface.co/docs/transformers/main/$_{c}lasses/trainertransformers.TrainingArguments.$ 2, 8

[5] https://www.philschmid.de/fine-tune-llms-in-2024-with-trl. 4, 8

[6] https://lambdalabs.com/blog/fine-tuning-metas-llama-2-on-lambda-gpu-cloud. 4, 8

[7] https://infrahub-doc.nexgencloud.com/docs/virtual-machines/virtual-machine-features. 4, 8

[8] https://huggingface.co/docs/transformers/en/perplexity. 5, 8

[9] https://medium.com/nlplanet/two-minutes-nlp-perplexity-explained-with-simple-probabilities-6cdc46884584. 5, 8

[10] https://huggingface.co/spaces/evaluate-metric/accuracy. 5, 8

[11] https://medium.com/@mayurdhvajsinhjadeja/jaccard-similarity-34e2c15fb524. 5, 8

[12] https://www.databricks.com/blog/llm-inference-performance-engineering-best-practices. 5, 8

[13] https://medium.com/@romxzg/maximizing-computing-power-a-guide-to-google-colab-hardware-options-a68469415291. 5, 8

[14] https://huggingface.co/docs/peft/v0.7.0/en/task/$_{g}uides/token - classification - lora.$ 8

[15] https://huggingface.co/docs/peft/v0.10.0/en/package/$_{r}eference/ia3peft.IA3Config.$ 8

[16] https://huggingface.co/docs/peft/main/en/task/$_{g}uides/clm - prompt - tuning.$ 8

[17] https://huggingface.co/docs/peft/main/en/task/$_{g}uides/clm - prompt - tuning.$ 8

[18] https://magazine.sebastianraschka.com/p/understanding-encoder-and-decoder. 8

[19] https://towardsdatascience.com/deploying-large-language-models-vllm-and-quantizationstep-by-step-guide-on-how-to-accelerate-becfe17396a2. 8

[20] Y. Li W. Chen-H. He C. Fan, J. Tian and Y. Jin. “chain-of-thought tuning: Masked language models can also think step by step in natural language understanding,”. arXiv.org, Oct. 18, 2023. https://arxiv.org/abs/2310.11721 (accessed Apr. 23, 2024). 3, 8

[21] P. Wallis Z. Allen-Zhu Y. Li S. Wang L. Wang E.J. Hu, Y. Shen and W. Chen. “lora: Low-rank adaptation of large language models,”. in Proc. Int. Conf. Learn. Representations, 2022. https://arxiv.org/abs/2106.09685. 2, 8

[22] A. Q. Jiang et al. “mistral 7b,”. arXiv.org, Oct. 10, 2023. https://arxiv.org/abs/2310.06825. 1, 8

[23] D. Khashabi et al. “unifiedqa: Crossing format boundaries with a single qa system.”. arXiv.org, Oct. 06, 2020. https://arxiv.org/abs/2005.00700. 1, 2, 4, 8

[24] Q. Zhang et al. “adaptive budget allocation for parameter-efficient fine-tuning,”. arXiv.org, Mar. 18, 2023. https://arxiv.org/abs/2303.10512. 2, 8

[25] Z. Liu et al. “deja vu: Contextual sparsity for efficient llms at inference time,”. arXiv.org, Oct. 26, 2023. https://arxiv.org/abs/2310.17157. 2, 8

[26] Z. Wan et al. “efficient large language models: A survey,”. arXiv (Cornell University), Dec. 2023. https://arxiv.org/abs/2312.03863. 1, 2, 3, 8

[27] M. Mohammed J. Mohta-T. Huang M. Bansal H. Liu, D. Tam and C. Raffel. “few-shot parameter-efficient fine-tuning is better and cheaper than in-context learning,”. in Proc. Adv. Neural Inf. Process. Syst., 2022. https://arxiv.org/abs/2205.05638. 3, 8

[28] S.-Z. J. Qin-X. Tao L. Xu, H. Xie and F. L. Wang. “parameter-efficient fine-tuning methods for pretrained language models: A critical review and assessment,”. arXiv.org, Dec. 19, 2023. https://arxiv.org/abs/2312.12148. 1, 2, 8

[29] A. Holtzman T. Dettmers, A. Pagnoni and L. Zettlemoyer. “qlora: Efficient finetuning of quantized llms,”. arXiv preprint 2023. https://arxiv.org/abs/2305.14314. 3, 8

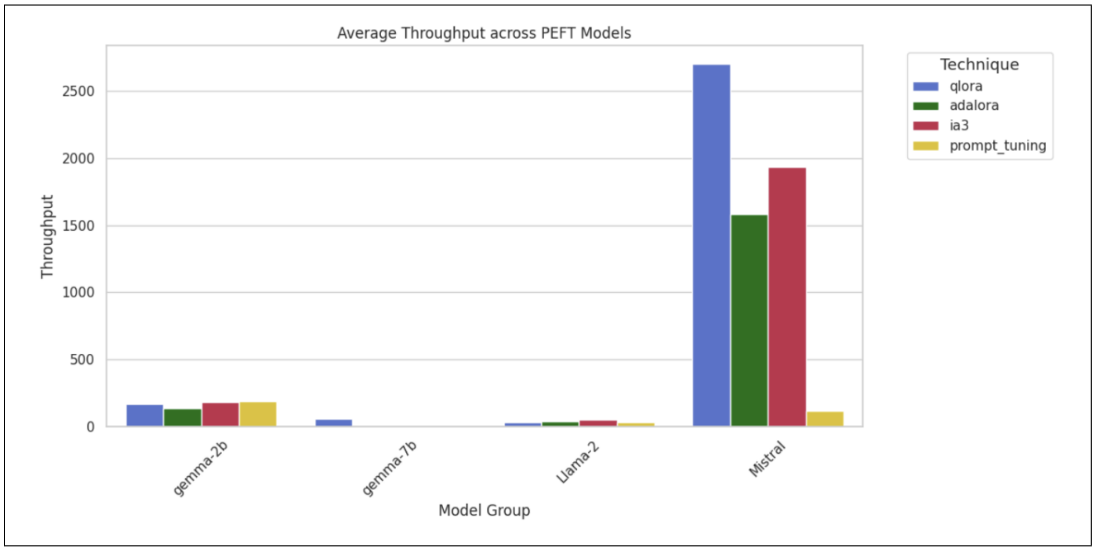

Figure 3. Average Throughput Across PEFT Models

| Student Name | Contributed Aspects |
|---|---|
| Andrey Shor | Downloaded and preprocessed UnifiedQA dataset. setup training pipeline for all models and for finetuning with Huggingface, PEFT, BitsandBytes Configs. Setup Lambda Labs instance machine. Setup data pipeline from lambda labs to GCP. Setup testing for training/finetuning of models. Created CLI for finetuning of models. |
| Micaela McCall | Implemented text generation and evaluation code for trained models, evaluated predictions using implemented metrics calculations, implemented inference efficiency methods, took lead on writing the report. |
| Artem Maryanskyy | Data preprocessing prompt engineering, result visualization, research on vm costs and needs, project kickoff coordination initial timeline planning. |

Table 7. Contributions of team members.

[30] A. Holtzman T. Dettmers, A. Pagnoni and L. Zettlemoyer. “qlora: Efficient finetuning of quantized llms,”. arXiv preprint 2023. https://arxiv.org/abs/2305.14314. 8

[31] Google Team and Google Deepmind. “gemma: Open models based on gemini research and technology.”. Available: https://storage.googleapis.com/deepmind-media/gemma/gemma-report.pdf. 8

[32] H. Touvron. “llama 2: Open foundation and fine-tuned chat models,”. arXiv.org, Jul. 19, 2023. https://arxiv.org/abs/2307.09288. 1, 8

[33] Z. Du M. Ding-Y. Qian Z. Yang X. Liu, Y. Zheng and J. Tang. “gpt understands, too,”. arXiv preprint 2021. https://arxiv.org/abs/2103.10385. 2, 8

[34] M. Kalman Y. Leviathan and Y. Matias. “fast inference from transformers via speculative decoding,”. arXiv.org, May 18, 2023. https://arxiv.org/abs/2211.17192. 8

[26] [23] [32] [31] [22] [28] [21] [30] [33] [29] [25] [27] [20] [34] [14] [1] [15] [2] [24] [16] [17] [6] [7] [5] [3] [4] [8] [9] [10] [12] [11] [**?**] [18] [13] [19] [**?**],