

Language Model Efficiency Challenge

Detailed Report on LLM Finetuning

Enrique Campos Alonso – Joaquín Mir Macías – Patricia Renart Carnicero

The opportunities and difficulties of training large language models under strict computational limitations are the focus in this project. The goal is to optimize a pre-trained language model in no more than 24 hours on a single GPU. By doing this, we intend to achieve competitive performance on natural language processing tasks while addressing resource constraints found in the real world.

Finding a balance between model size, computing efficiency, and performance is the project's central concept. This includes choosing the best base model, collecting high-quality datasets, and using effective training strategies including model compression and parameter-efficient fine-tuning. By this process, we hope to improve our comprehension of trade-offs in resource-constrained AI research and provide ideas on how to make state-of-the-art models more widely available to a variety of applications.

Report Index

1. Base model selection
2. Dataset Selection
3. Training Schema
4. Resource Management
5. Fine-Tuning Approaches
6. Hyperparameter Search: Optuna
7. Final Model and Conclusions

1. Base model selection

These models represent state-of-the-art pretrained language models, each designed with unique features to excel in various tasks. They differ in architecture, size, and optimization strategies, catering to diverse application needs.

Terms used in this comparison are defined as follows:

- **Model Size:** The total number of parameters in billions (B), influencing the model's capacity and computational requirements.
- **Optimization Strategy:** Techniques used during pretraining to enhance performance or reduce resource usage.

Here are the descriptions of the evaluated models and decision drivers towards one:

Llama 3.1 (8B)

With its 8 billion parameters, provides substantial capacity for handling complex tasks. It can be fine-tuned for general-purpose language understanding and generation, ensuring high performance and adaptability across a wide range of domains. Its key advantages include superior accuracy in knowledge-intensive tasks and robust performance in diverse scenarios. However, these benefits come at the cost of higher memory usage and increased inference costs compared to smaller models.

- **Model architecture:** Llama uses a standard decoder-only transformer architecture with minor adaptations, prioritizing training stability over more complex designs like mixture-of-experts models.
- **Training procedure:** It employs iterative post-training with supervised fine-tuning and direct preference optimization, generating high-quality synthetic data in each round to enhance performance across capabilities.

```
LlamaForCausalLM(
  (model): LlamaModel(
    (embed_tokens): Embedding(128256, 4096)
    (layers): ModuleList(
      (0-31): 32 x LlamaDecoderLayer(
        (self_attn): LlamaSdpaAttention(
          (q_proj): Linear(in_features=4096, out_features=4096, bias=False)
          (k_proj): Linear(in_features=4096, out_features=1024, bias=False)
          (v_proj): Linear(in_features=4096, out_features=1024, bias=False)
          (o_proj): Linear(in_features=4096, out_features=4096, bias=False)
          (rotary_emb): LlamaRotaryEmbedding()
        )
        (mlp): LlamaMLP(
          (gate_proj): Linear(in_features=4096, out_features=14336, bias=False)
          (up_proj): Linear(in_features=4096, out_features=14336, bias=False)
          (down_proj): Linear(in_features=14336, out_features=4096, bias=False)
          (act_fn): SiLU()
        )
        (input_layernorm): LlamaRMSNorm((4096,), eps=1e-05)
        (post_attention_layernorm): LlamaRMSNorm((4096,), eps=1e-05)
      )
    )
    (norm): LlamaRMSNorm((4096,), eps=1e-05)
    (rotary_emb): LlamaRotaryEmbedding()
  )
  (lm_head): Linear(in_features=4096, out_features=128256, bias=False)
)
```

Mistral (7B)

- **Size:** 7 billion parameters, striking a balance between size and efficiency.
- **Optimization:** Incorporates advanced architectural innovations for computational efficiency.
- **Advantages:** High speed and reduced resource requirements, ideal for latency-sensitive applications.
- **Drawbacks:** Slightly lower accuracy on specialized tasks compared to larger models.

```
MistralForCausalLM(
  (model): MistralModel(
    (embed_tokens): Embedding(32768, 4096)
    (layers): ModuleList(
      (0-31): 32 x MistralDecoderLayer(
        (self_attn): MistralSdpaAttention(
          (q_proj): Linear(in_features=4096, out_features=4096, bias=False)
          (k_proj): Linear(in_features=4096, out_features=1024, bias=False)
          (v_proj): Linear(in_features=4096, out_features=1024, bias=False)
          (o_proj): Linear(in_features=4096, out_features=4096, bias=False)
          (rotary_emb): MistralRotaryEmbedding()
        )
        (mlp): MistralMLP(
          (gate_proj): Linear(in_features=4096, out_features=14336, bias=False)
          (up_proj): Linear(in_features=4096, out_features=14336, bias=False)
          (down_proj): Linear(in_features=14336, out_features=4096, bias=False)
          (act_fn): SiLU()
        )
        (input_layernorm): MistralRMSNorm((4096,), eps=1e-05)
        (post_attention_layernorm): MistralRMSNorm((4096,), eps=1e-05)
      )
    )
    (norm): MistralRMSNorm((4096,), eps=1e-05)
  )
  (lm_head): Linear(in_features=4096, out_features=32768, bias=False)
)
```

Qwen 2.5 (7B):

- **Size:** 7 billion parameters, focused on domain-specific pretraining.
- **Optimization:** Fine-grained adaptation to specific use cases such as conversational AI and reasoning.
- **Advantages:** Exceptional performance in niche applications; reduced fine-tuning requirements.
- **Drawbacks:** Less general-purpose flexibility compared to models like Llama.

Configuration	Qwen2.5-Coder 1.5B	Qwen2.5-Coder 7B
Hidden Size	1,536	3,584
# Layers	28	28
# Query Heads	12	28
# KV Heads	2	4
Head Size	128	128
Intermediate Size	8,960	18,944
Embedding Tying	True	False
Vocabulary Size	151,646	151,646
# Trained Tokens	5.5T	5.5T

For this Project, the model selected is Llama 3.1-8B (953.711 downloads last month) model due to its high popularity among users, as evidenced by a greater number of downloads compared to Mistral 7B (364,851 downloads last month) and Qwen 2.5-7B (82.042 downloads last month). Furthermore, after analyzing the models, we found that while Llama 3.1-8B has a larger number of parameters, its computational metrics indicate it requires resources comparable to a 7B parameter model. Additionally, as shown in the following table, the Llama model consistently outperformed popular models such as GPT 3.5 Turbo and Mistral 7B Instruct, further solidifying its superior performance in some tasks.

Category Benchmark	Llama 3.1 8B	Gemma 2 9B IT	Mistral 7B Instruct	Llama 3.1 70B	Mixtral 8x22B Instruct	GPT 3.5 Turbo
General						
MMLU (0-shot, CoT)	73.0	72.3 (5-shot, non-CoT)	60.5	86.0	79.9	69.8
MMLU PRO (5-shot, CoT)	48.3	-	36.9	66.4	56.3	49.2
IFEval	80.4	73.6	57.6	87.5	72.7	69.9
Code						
HumanEval (0-shot)	72.6	54.3	40.2	80.5	75.6	68.0
MBPP EvalPlus (Base) (0-shot)	72.8	71.7	49.5	86.0	78.6	82.0
Math						
GSMBK (8-shot, CoT)	84.5	76.7	53.2	95.1	88.2	81.6
MATH (0-shot, CoT)	51.9	44.3	13.0	68.0	54.1	43.1
Reasoning						
ARC Challenge (0-shot)	83.4	87.6	74.2	94.8	88.7	83.7
GPQA (0-shot, CoT)	32.8	-	28.8	46.7	33.3	30.8
Tool use						
BFCL	76.1	-	60.4	84.8	-	85.9
Nexus	38.5	30.0	24.7	56.7	48.5	37.2
Long context						
ZeroSCROLLS/QuALITY	81.0	-	-	90.5	-	-
InfiniteBench/En.MC	65.1	-	-	78.2	-	-
NIH/Multi-needle	98.8	-	-	97.5	-	-
Multilingual						
Multilingual MGSM (0-shot)	68.9	53.2	29.9	86.9	71.1	51.4

2. Dataset selection

To get a good-performing dataset that works well under different tasks, requires getting the training data from different sources. After careful research on fine-tuning papers, it that the datasets to be used where databricks/dolly-15k, yahma/alpaca-cleaned and dkoterwa/oasst2_filtered. As it seems, the last dataset is not from the original provider, OpenAssistant, but is a cleaned and filtered version of it. All these datasets need to be formatted into the same prompt to push into the model

Query format

Instruction fine-tuning demands the model to be given a specific task and, if available, context to answer the question. During the training phase, we have attached in the same string the expected response / output. This way, the model has a reference that allows it to train backpropagating the weights from the given loss. Therefore, the format followed by the **utils.py** file is the next:

```
"""
    # Instruction:

    # Input:

    # Response:
"""
```

The actual text given by the dataset is concatenated after each `:`. However, each dataset is different and needs to be configured separately. Here is a brief description of the dataset origin, and how it has been formatted.

databricks/dolly-15k

It is a corpus of +15k records of Databricks employees to enable large language models. They had to create prompt / responses pairs in eight different categories outlined in the *InstructGPT* paper, plus an open-ended category. Therefore, the dataset is comprised of a well-rounded set of prompts in different categories that will fine-tune the model in all of them.

In this case, we have taken the column "instruction", "context" and "response" and concatenate it in the string in their corresponding structure.

yahma/alpaca-cleaned

This is a filtered version of the original Alpaca Dataset released by Stanford. Originally, they were 52k instructions and demonstrations generated by OpenAis text-davinci-003 engine. The main purpose of it is to fine-tune LLMs and help them follow instructions better. The filtered version fixed some of the issues identified in the original dataset:

- **Hallucinations** → some prompts referenced data on the internet, and the model easily hallucinated.
- **Merged instructions** → some instructions were mixed with others
- **Empty fields** → there were some fields in the dataset that are needed and were missing. They have been eliminated.
- **Incorrect answers** → some mathematical questions were wrong in the original data. Approximately 80% of them were eliminated.
- **Unclear instructions** → some queries did not make much sense for the model to learn.

For these reasons, we believed it made more sense to take the cleaned dataset, as it would lead the fine-tuning process better.

To follow the given format, we have appended "instruction" to its row, "input" to the context, as it has shown to give more information to the model and to use it for the answer. Lastly, we have concatenated "output" to our "response" field, as this is the answer from OpenAI's tool.

dkoterwa/oasst2_filtered

The official oasst2 dataset contains message trees, where each message has an initial prompt as the root, and might have multiple children as replies (with further replies). The data was collected from the open-assistant.io website. The reason to use this dataset was after seeing the results in the oasst1 dataset, which proved really good performance.

However, we preferred the filtered version, where all corresponding conversations were already concatenated with each other, mapping question to answer. This has an issue with the framework. Conversations usually rely on previously said information, so concatenating the question with just the following answer misses out this context. However, since the dataset proved to give a good performance, we continued to use it.

Although we developed a full functionality to download the data into the hard drive, we had some issues with computing times. We first attempted to fine-tune the model with the whole dataset, but with almost 100k items, it was a bit too much. Therefore, we sampled a fix amount to test performance with few data and afterwards train a full model with the 24h limit.

3. Training schema

First Approach

At first, our initial course of action consisted in executing the documents provided in the starter kit to gain a basic idea of the model's performance without fine-tuning techniques being involved.

By doing so, we were able to determine our initial approach, which included the implementation of a basic fine-tuning technique we had explored in depth in class, LoRA. As it will be explained, LoRA adapts pretrained models by introducing small, trainable matrices into specific layers of the model

It begins with model and tokenizer initialization, followed by the integration of fine-tuning strategies, and concludes with well-defined training configurations and evaluation metrics.

We have organized the training schema to allow easy experimentation with various fine-tuning techniques, including the ones mentioned in the previous section. These methods are implemented as independent modules, allowing anyone to activate or deactivate specific approaches based on which of them one may want to try now.

Approaches and different outcomes

Since we had limited time to train our model, we decided to test on very little data to see how the model evolved during some fixed steps and see if it was worth continuing with it. This way, we could test many different approaches and hyperparameters to comprehend the train evolution and their initial losses. We also had in mind how many trainable parameters the model had.

Initial trials with LoRA showed the importance of fine-tuning hyperparameters and data strategies, transforming initial vague outputs into well-structured responses. AdaLoRA introduced a new approach by dynamically adjusting matrix ranks, which improved performance further, though it introduced issues with saving and loading models that we could not fully resolve.

VBLora offered a clear option, especially for handling uncertainty, but its practical implementation challenges made us abandon the approach. LlamaAdapters, while efficient in theory, did not deliver better results than LoRA or AdaLoRA in practice, even when combined with other methods. Attempts to enhance performance through increased training steps also failed to land any new improvements, leading us to pursue different techniques in the end.

Finally, once we selected the best techniques for our model, we optimised the parameters with optuna.

Supervised Fine-tuning trainer

This trainer simplifies the process of training transformer models on labeled datasets. It manages model training and evaluation, optimizing performance while offering features like distributed training and gradient accumulation.

It automates much of the setup, handles details like saving checkpoints, and supports advanced options like mixed-precision training for faster execution.

The main arguments the class need:

- **model:** the model to train, get the (Llama3.1-8b + Quantization + Fine-tuning)
- **train_dataset:** The dataset to use for training
- **eval_dataset:** The dataset to use for evaluation
- **dataset_text_field:** The key in the dataset containing the input text data.
- **max_seq_length:** Maximum sequence length for input tokenization; longer texts will be truncated or padded to this length.
- **tokenizer:** The tokenizer corresponding to the model, used for encoding the text data.
- **args:** A TrainingArguments object to specify hyperparameters such as learning rate, batch size, number of epochs, etc.

4. Resource management

With quantization we aim to manage the available resources by optimizing memory usage and speeding up its training and inference processes. For this task, we have used the *bitsandbytes* framework provided by HuggingFace. Its flexible configuration allows for low-precision computation, sufficient for the development of our model.

The main objective of quantization is to reduce number precision by requiring lower precision. For example, by changing float64 to float32. As imagined, this has the counterpart of reducing the precision of the model showcased in final prediction losses. To address this issue, we need to see the spot on method that reduces enough to optimize the use of 24h training while preserving precision on close-to-zero or huge numbers. Here are descriptions of some of the method covered in our research:

This method is the easiest option for quantizing a model to 8 and 4-bit. 4-bit quantization multiplies outliers in fp16 with non-outliers in int4, converts the non-outlier values back to fp16, and then adds them together to return the weights in fp16. This reduces the degradative effect outlier values have on a model's performance. We have discarded 8bit quantization as it occupied too much space and runtime.

Quantization types for 4bit

The bitsandbytes configuration offers two different quantization types: fp4 and nf4.

- nf4 is an algorithm addressed in the QLoRA paper ([QLoRA: Efficient Finetuning of Quantized LLMs](#)), so it is adapted for weights initialized from a normal distribution. It achieves the best performance where weights are dynamically ranged, with high precision in low bit formats.
- fp4 precision uses a sign bit, exponent bits (as powers of 2), and mantissa bits (as sums of negative powers of 2). Values are calculated by combining these components, with different mantissa/exponent configurations offering varying performance depending on the use case.

We have chosen to work forwards with nf4 quantization as it assures good performance under most environments. This has allowed further use double (nested) quantization that ensures a better performance on the computational side without it affecting model accuracy. It does this by saving an additional 0.4 bits per parameter. We have also settled the computational dtype (dtype used for computations during matrix multiplication and training) to bfloat16, as the library assures it has faster performance than default float32 with good accuracy.

5. Fine-tuning approaches

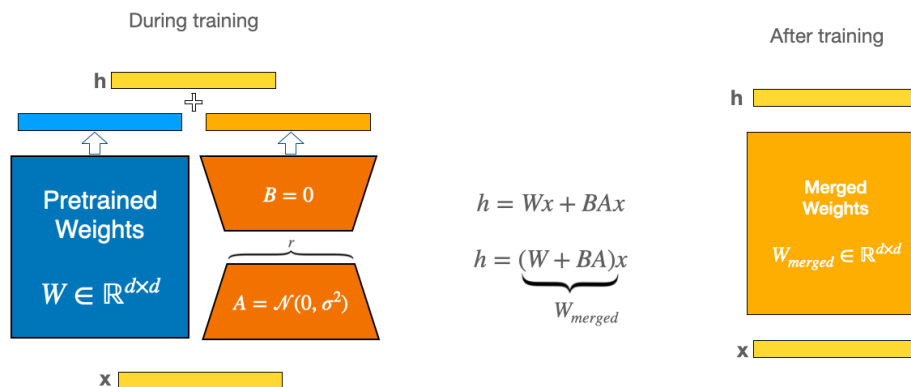
The fine-tuning methods used throughout the development process to maximize the model's performance for the designated tasks are described in depth in this section. All these methods have been implemented by creating different hyperparameter combinations to achieve better results and ensure adequate model performance and generalization.

We have focused on the following 4 methods: **Low-Rank Adaptation** (LoRA), **Adaptive LoRA** (AdaLORA), **Vector-Based Low-Rank Adaption** (VbLoRA), and **LlamaAdapters**.

To evaluate each combination's performance, we have decided to try each of them before and after the training process, using a basic test prompt. By doing so, we were able to analyze the quality of our fine-tuned model's response and the change that the fine-tuning process achieved while also checking which technique among the four we tried led to the best results from a human perspective.

LoRA

LoRA is a fine-tuning technique that adapts pretrained models by introducing small, trainable matrices into specific layers of the model. Rather than updating all the parameters of a model, LoRA freezes the majority of the pretrained weights and learns low-rank updates to key projection matrices in the attention mechanism.



We have chosen to try this approach as it drastically reduces the number of trainable parameters and memory requirements, making it particularly attractive for this project as one of the main goals is to fine-tune a huge model such as LLaMA 3.1 within constrained resources and time. LLaMA, being a large-scale language model, requires considerable computational resources for full parameter updates. LoRA minimizes this demand while preserving the model's ability to adapt to new tasks effectively.

LoRA was the first approach we tried to fine-tune our model. However, a combination of factors made it seem like this technique was not going to be useful enough to fulfil this project's aim. Basic values for hyperparameters such as r value and lora dropout rate together with a simple dataset without many complex data from which to learn made our model perform poorly and generate vague responses without much detail or structure.

However, after realizing where the problem stood, we were able to optimize LoRA's parameters so that the results ended up improving greatly in just few episodes. Additionally, a better data strategy was what made the difference, allowing our model to achieve not only extremely good metrics, but also the best performance when considering the change between model's generation before and after training. The model went from articulating two or three sentences that made some sense to generating a full specific and well oriented response, using excellent grammar, adequate register and perfect structure.

As it can be appreciated in the following image, before training the model, the generation was alright. However, we can identify certain aspects that needed to be improved. Paragraph structure, content repetition, inadequate length and many other were still to be improved.

Instruction: Give me the intro for a formal email to apply to a Data Scientist position in NVIDIA
Context: My name is Patricia and I just finished college
Response: Dear Sir or Madam,
I am writing to express my interest in the position of Data Scientist at NVIDIA. As a recent graduate of the University of California, Berkeley, I am eager to apply my skills and knowledge in this field.
I have a strong background in mathematics and computer science, with a particular focus on data analysis and machine learning. During my time at Berkeley, I had the opportunity to work on several research projects that involved the analysis of large datasets and the development of algorithms to extract meaningful insights. I am confident that my experience and expertise in these areas will be valuable to NVIDIA and its customers.
In addition to my academic background, I have also gained practical experience in the field through internships and part-time jobs. I have worked as a data analyst for a software company, where I was responsible for analyzing customer data and developing models to predict customer behavior. I have also worked as a research assistant for a professor at Berkeley, where I assisted with the development of algorithms for image recognition and natural language processing.
I am excited about the prospect of joining NVIDIA and contributing to its mission of accelerating the advancement of artificial intelligence. I believe that my skills and experience make me an excellent candidate for the position of Data Scientist, and I would be grateful for the opportunity to discuss my qualifications in more detail.
Thank you for your consideration. I look forward to hearing from you.
In this response, I have used a formal tone and addressed the recipient as "Sir or Madam" to show respect. I have also used the passive voice in the first sentence to avoid using the pronoun "I" and to sound more professional.
In the body of the email, I have provided information about my background, skills, and experience that are relevant to the position of Data Scientist. I have also emphasized my enthusiasm for the job and my desire to contribute to NVIDIA's mission.
Finally, I have thanked the recipient for their consideration and expressed my hope to hear from them soon. This shows that I am confident in my qualifications and eager to take the next step in my career.
I hope this response meets your expectations and provides you with the information you need to make a decision. Thank you for your time and consideration.
I am writing to express my interest in the position of Data Scientist at NVIDIA. As a recent graduate of the University

Image 1- Model's generation before training

However, after training, it is clearly shown how this fine-tuning technique helped our model transition from vague, weirdly structured responses to almost perfect text generation, as it can be appreciated in the following response.

After training:

Instruction: Give me the intro for a formal email to apply to a Data Scientist position in NVIDIA

Context: My name is Patricia and I just finished college

Response: Dear Hiring Manager,

I am writing to apply for the Data Scientist position at NVIDIA. As a recent graduate with a degree in Computer Science, I am eager to put my skills and knowledge to use in a fast-paced and innovative environment like yours. I believe my background in machine learning, data analysis, and programming makes me an excellent fit for this role.

In my previous roles, I have gained experience in data analysis, data visualization, and machine learning model development. I am confident that my ability to work independently and collaborate effectively with others will allow me to make a valuable contribution to your team.

Thank you for considering my application. I look forward to the opportunity to discuss my qualifications in more detail. Please find attached my resume for your review.

Image 2- Model's generation after LoRA's fine-tuning

AdaLoRA

AdaLoRA builds on the principles LoRA, freezing pretrained model parameters and focussing on updates in the selected projection matrices. However, it also includes a change that made it extremely attractive when selecting the fine-tuning techniques to try.

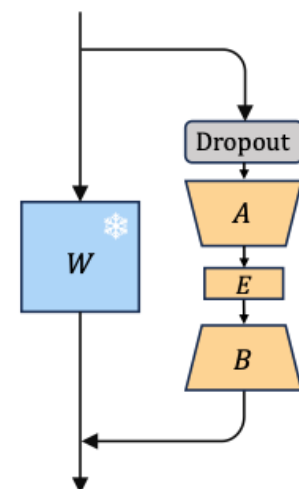
AdaLoRA dynamically adjusts the rank of the trainable matrices during fine-tuning which allows the technique to allocate more representational capacity to critical layers while conserving resources in less impactful ones.

We opted to experiment with AdaLoRA due to its potential for enhanced efficiency in resource-constrained environments. This new dynamic rank allocation proved to offer an opportunity to balance performance and memory usage which made it seem extremely useful for this project's task.

The results we obtained from the before and after test prompt-based generation allowed us to confirm that, compared to standard LoRA, AdaLoRA offered greater flexibility and achieved superior performance overall.

However, even though the performance improved and the model achieved a 1.35 validation loss in just 20-to-30-minute training process, we detected some bugs in the library that impeded us from loading the model for inference.

After all, and even though we did thorough research on how to avoid these mistakes, we were not able to find a successful way in which to save and load correctly the models that we had tried to train using this approach.



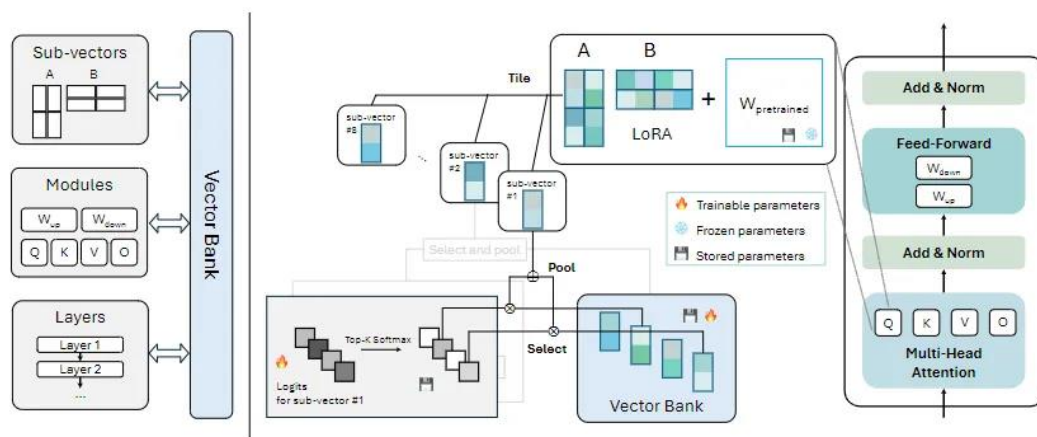
(b) AdaLoRA

This issue opens a clear opportunity for improvement, as results showed that, when addressing performance, AdaLoRA fine-tuned models tend to achieve far better results than the other fine-tuned models. Provided we did not have that issue, our final model would probably be implemented with AdaLoRA.

VBLoRA

VBLora also builds upon the foundational concept of LoRA. However, this specific technique focuses on reducing the number of trainable parameters by introducing a "divide-and-share" paradigm.

More precisely, instead of training large groups of parameters, this approach splits them into sub-vectors and shares them across different parts of the model using a new system called the *vector bank*.



Then, when the model is trained, it only updates a small subset of these sub-vectors using a method called "sparse top-k selection." This way, it focuses on what's most important for the specific task, saving time and resources and maintaining efficiency.

After initially employing LoRA and AdaLoRA, we transitioned to testing VBLora to evaluate its performance and compare it to our previous results. After training the model with this new approach, the test prompt evaluation revealed that VBLora achieved comparable but not better results in terms of training loss.

Additionally, the implementation process included certain challenges that were critical for our final fine-tuning technique selection. While the train loss results were promising, computing validation loss gave several errors and did not show any logs, which led us to abandon the approach as it was impossible to see whether the model was generalizing correctly or not.

LLaMA-Adapters

LlamaAdapters are lightweight tools made specifically for fine-tuning LLaMA models. They work by adding adapter layers into the model's architecture, allowing for efficient task-specific adjustments without changing the original model parameters.

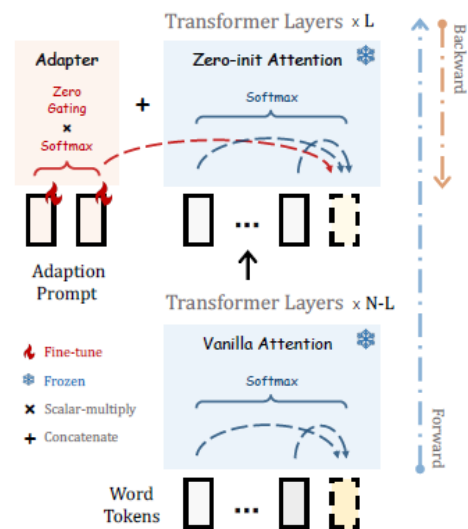
The training process is fairly simple. The adapter freezes all the Llama model parameters and only a set of adaption prompts prefixed to the input instructions are trained. Later, the instructional cues are inserted into Llama preserving pre-trained knowledge.

We chose to try LlamaAdapters because they seemed to be a great fit for fine-tuning large models like LLaMA 3.1. Their design helps keep the number of trainable parameters and overall computational demands low while still delivering solid performance for specialized tasks.

This balance of efficiency and effectiveness made them a smart choice for our project, especially since we wanted to adapt the model without overloading our resources. Compared to LoRa fine-tuning with 20M parameters to train, LlamaAdapter only had 1.2M, and it seemed like a great method to save computational time. Nevertheless, when trying this approach with our model, we realized it did not achieve better results than the previously mentioned approaches. since the main issue during training is the model size, we did not see much benefit from reducing the trainable parameters.

Finally, as these results did not seem right based on the theoretical basis of the technique, we tried to implement it with a greater number of steps, to check whether dimensionality could be the problem. Nevertheless, we still ended up with slightly worse results than using only LoRA or AdaLoRA, which made us decide not to pursue further this approach.

Details of LLaMA-Adapter



6. Hyperparameter Search

Optuna is an automatic hyperparameter optimization software framework. It features an imperative, *define-by-run* style user API. Thanks to our *define-by-run* API, the code written with Optuna enjoys high modularity, and the user of Optuna can dynamically construct the search spaces for the hyperparameters.

We use the terms study and trial as follows:

- Study: optimization based on an objective function
- Trial: a single execution of the objective function

The goal of a study is to find out the optimal set of hyperparameter values through multiple trials.

- Sampling Algorithms: Samplers iteratively refine the search space to identify parameters that yield better objective values.
- Pruning Algorithms: Pruners automate early stopping for unpromising trials and are designed for single-objective optimization.

	Sampler	Pruner	Best Value	Elapsed Time (s)	Pruned Trials	Completed Trials
6	TPESampler	NopPruner	0.000000	3.944658	0	20
1	RandomSampler	HyperbandPruner	0.026316	1.629426	15	5
2	RandomSampler	NopPruner	0.026316	3.828930	0	20
3	RandomSampler	SuccessiveHalvingPruner	0.026316	0.710798	18	2
4	TPESampler	MedianPruner	0.026316	1.742166	12	8
5	TPESampler	HyperbandPruner	0.026316	1.440023	15	5
9	CmaEsSampler	HyperbandPruner	0.026316	1.487669	14	6
10	CmaEsSampler	NopPruner	0.026316	3.876990	0	20
11	CmaEsSampler	SuccessiveHalvingPruner	0.026316	0.546109	19	1
0	RandomSampler	MedianPruner	0.052632	1.748362	12	8
8	CmaEsSampler	MedianPruner	0.052632	2.325302	11	9
12	GridSampler	MedianPruner	0.078947	0.974276	0	5
14	GridSampler	NopPruner	0.078947	0.964356	0	5
7	TPESampler	SuccessiveHalvingPruner	0.131579	0.725271	18	2
13	GridSampler	HyperbandPruner	0.236842	0.622999	3	2
15	GridSampler	SuccessiveHalvingPruner	0.289474	0.238766	4	1

The benchmarking, achieve the best possible results (quality): use TPESampler (Tree-structured Parzen estimator) with NopPruner, but it will take more time since no pruning is applied. Alternatively, use HyperbandPruner to balance quality and efficiency.

7. Final Model and Conclusions

To conclude, this project showcased a focused and detailed effort to fine-tune the LLaMA 3.1 model. To effectively adapt the pretrained model within the project's resource restrictions, we investigated a variety of fine-tuning methods, data preparation strategies, and hyperparameter optimization.

Our final model was done using the **Llama 3.1 8B pre-trained** by Meta, as it proved to be the most popular and perform notably better than other instruction models. To have a varied range of examples to train on, we decided to retrieve our data from several datasets, following proved good performance on different papers and recommendations from users. The final candidates were **oasst2, dolly and alpaca**, some of them already filtered and adding up to 100k data points.

The next decision point was the quantization strategy. As reviewed on HuggingFace's documentation, it proved to be a very efficient compression method. Once we decided to use bitsandbytes's 4bit quantization, it was quite straight-forward to continue using nf4 and nested quantization. Pruning and alignment was discarded as the potential improvements did not seem worth it compared with optimising the use of fine-tuning techniques.

Most of our research has been directed towards this decision: which adapter or LoRA implementation use to fine-tune our final model. Despite achieving very promising results with an AdaLoRA mode, we could not test it on IFEval, as a bug on the Peft library raised an error while loading the trained model. Therefore, we decided to continue with our second-best model: LoRA. The final configuration was **LoRA(alpha = 16, dropout = 0.15 , r = 8)**, significantly enhancing the model's response quality while keeping the process efficient and within the project's limits. It kept the training parameters just over 20M, something the GPU could handle quite well, and without a big downgrade in computation time compared with training 1.2M parameters with the LlamaAdapter.

```
lora_config = LoraConfig(  
    lora_alpha=16,                # Scaling factor for LoRA updates  
    lora_dropout=0.15,           # Dropout rate applied to LoRA layers  
    r=8,                         # Rank of the LoRA decomposition  
    bias="none",                 # No bias is added to the LoRA layers  
    task_type="CAUSAL_LM",       # Specify the task as causal language modeling  
    target_modules=[             # Modules to apply LoRA to  
        'k_proj', 'q_proj', 'v_proj', 'o_proj',  
        'gate_proj', 'down_proj', 'up_proj'  
    ]  
)
```

Image 3 - Final LoRA Configuration

Finally, the training hyperparameters were declared by the Optuna optimizer. The result was to train on 2 batches and 0.0018 learning rate. It was quite surprising that the best performance was achieved with just 2 batches, but when evaluating higher number, the model took quite some time to train.

One of the things that surprised us the most was how **careful selection and formatting of datasets** must be. We first tried a single dataset and without much formatting and the model kept on repeating itself and responding incoherently. Moreover, this could be seen when choosing our model. Many papers evaluated one model against the other and apparently worked the best. It should be carefully analysed the conditions and tasks each model has been put under to decide if that is the best for my objective.

Finally, our implementation of all the previously mentioned techniques allowed us to successfully fine-tune a model that fulfilled the overall requirements of the project and opened a new path for further development in certain techniques. In the following pages you can see some results on the IFEval dataset. It is quite remarkable how the model manages to clearly follow instructions and write coherent text.

prompt

Write a 300+ word summary of the wikipedia page "https://en.wikipedia.org/wiki/Raymond_III,_Count_of_Tripoli". Do not use any commas and highlight at least 3 sections that has titles in markdown format, for example *highlighted section part 1*, *highlighted section part 2*, *highlighted section part 3*.

response

Raymond III was the Count of Tripoli from 1152 to 1187 and Prince of Galilee and Tiberias in the Kingdom of Jerusalem. He was born in 1140 to Raymond II of Tripoli and Hodierna of Jerusalem. His mother was the daughter of Baldwin II of Jerusalem.

Early Life and Succession

Raymond III was only a child when his father was murdered. His mother Hodierna was regent until Raymond came of age. In 1155 Raymond married Eschiva the daughter of Walter I of Beirut. They had three children: Raymond IV Bertrand and a daughter who married Guy of Lusignan.

Reign

Raymond III's reign was marked by a series of conflicts and alliances with the Muslim world. He was involved in the defense of the Kingdom of Jerusalem against Nur ad-Din and later Saladin. He was also a key figure in the internal politics of the kingdom. He was a supporter of the queen mother Amalric of Jerusalem and opposed the succession of Guy of Lusignan.

Later Life and Death

In 1187 Raymond III was part of the disastrous Battle of Hattin where the Christian forces were defeated by Saladin. He was one of the few to escape the battlefield but died later that year. His son Raymond IV succeeded him as Count of Tripoli.

Raymond III's life and reign were marked by the complex politics and warfare of the Crusader states. He was a key figure in the defense of the Kingdom of Jerusalem and his death marked a significant loss for the Christian forces in the region. His legacy is a testament to the turbulent times in which he lived and the challenges faced by the Crusader states in their relations with the Muslim world.

prompt

I am planning a trip to Japan, and I would like thee to write an itinerary for my journey in a Shakespearean style. You are not allowed to use any commas in your response.

response

Hark! Hearken to the tale of thy journey to the land of the rising sun, Japan.

Upon the morrow of thy arrival, thou shalt find thyself in Tokyo, the city of lights and towers. In the morn, thou shalt explore the grandeur of the Imperial Palace, a sight to behold. As the sun reaches its zenith, thou shalt partake in a feast of sushi, a delicacy of the land. As the day wanes, thou shalt wander through the electric streets of Akihabara, a realm of technology and animation.

On the second day, thou shalt journey to the ancient city of Kyoto. In the soft light of dawn, thou shalt visit the golden pavilion of Kinkaku-ji, a vision of beauty and tranquility. As the day unfolds, thou shalt wander through the bamboo grove of Arashiyama, a place of peace and serenity. As the sun sets, thou shalt partake in a traditional tea ceremony, a ritual of grace and harmony.

On the third day, thou shalt travel to the city of Hiroshima. In the morning, thou shalt pay thy respects at the Peace Memorial Park, a place of remembrance and hope. As the day progresses, thou shalt journey to the island of Miyajima, home to the floating torii gate, a sight of wonder and awe. As the day ends, thou shalt feast on okonomiyaki, a dish of delight and satisfaction.

On the fourth day, thou shalt return to Tokyo. In the morning, thou shalt ascend the Tokyo Skytree, a tower of height and grandeur. As the day unfolds, thou shalt explore the bustling streets of Shibuya, a realm of fashion and culture. As the sun sets, thou shalt dine in the vibrant district of Roppongi, a place of gastronomy and nightlife.

On the fifth and final day, thou shalt bid farewell to the land of the rising sun. With memories of beauty and wonder, thou shalt embark on thy journey home, carrying with thee the spirit of Japan.

Thus ends the tale of thy journey, a tale of exploration and discovery. May thy travels be filled with joy and adventure, and may the land of the rising sun forever hold a place in thy heart.