Parameter-Efficient Fine-Tuning of LLMa-2 using Quantization Low-Rank Adaptation

by

Md. Tariqul Islam
23173006

A project submitted to the Department of Computer Science and Engineering
in partial fulfillment of the requirements for the degree of
M.Engg. in Computer Science

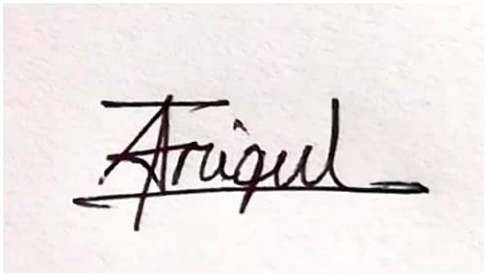
Department of Computer Science and Engineering
Brac University
November 2024

Declaration

It is hereby declared that

1. The project submitted is my own original work while completing degree at Brac University.
2. The report does not contain material previously published or written by a third party, except where this is appropriately cited through full and accurate referencing.
3. The project does not contain material which has been accepted, or submitted, for any other degree or diploma at a university or other institution.
4. I have acknowledged all main sources of help.

Student's Full Name & Signature:



Signature

Md. Tariqul Islam

Name

Approval

The project titled "Parameter-Efficient Fine-Tuning of LLMa-2 using Quantization Low-Rank Adaptation" submitted by

1. Md. Tariqul Islam (23173006)

Of Summer, 2024 has been accepted as satisfactory in partial fulfillment of the requirement for the degree of M.Engg. in Computer Science and Engineering on 14th November, 2024.

Examining Committee:

Supervisor:
(Member)

Moin Mostakim
Senior Lecturer
Department of Computer Science and Engineering
Brac University

Program Coordinator:
(Member)

Md Sadek Ferdous, PhD
Professor
Department of Computer Science and Engineering
Brac University

Head of Department:
(Chair)

Sadia Hamid Kazi, PhD
Chairperson and Associate Professor
Department of Computer Science and Engineering
Brac University

Abstract

Llama-2, an advanced neural network with huge potential in text generation, sentiment analysis and language understanding. This report focuses on the fine-tuning process for build chatbot on custom datasets, specification methods, hyperparameters and training strategies. Experimental results on Guanchu datasets show excellent adaptability of the model, outperforming the baseline model in human evaluation and achieving significant BERT scores for help and safety. The analysis includes an in-depth examination of LAMA-2's architecture, outlining strengths and suggesting areas for improvement. Parameter-efficient fine-tuning and quantization also investigate the transformative potential of LLMA-2 through low-rank adaptation. The objective is to strike a balance between model complexity and efficiency, addressing challenges in resource-constrained environments.

Keywords: Llama-2; Neural network; Hyperparameters; PEFT; Quantization Low-Rank Adaptation: Natural language processing

Acknowledgement

Firstly, all praise to the Great Allah for whom my project have been completed without any major interruption.
Secondly, to our co-advisor Mr. Moin Mostakim sir for his kind support and advice in our work. He helped us whenever we needed help.
And finally to our parents without their throughout sup-port it may not be possible. With their kind support and prayer we are now on the verge of our graduation.

Table of Contents

List of Figures

List of Tables

Nomenclature

The next list describes several symbols & abbreviation that will be later used within the body of the document

$BERT$ Bidirectional Encoder Representations from Transformers

GQA Grouped Query Attention

$LLMs$ Large Language Models

$LoRA$ Low Rank Adapter

$OASST$ OpenAssistant Conversations

$PEFT$ Parametric Efficient Fine-Tuning

$PLMs$ Pre-trained Language Models

PPO Proximal Policy Optimization

$RLAIF$ Introduce AI models using feedback

$RLHF$ Human Feedback Enhanced Learning

SFT Supervised Fine-Tuning

Chapter 1

Introduction

In recent years, LLMs have shown great potential in language understanding and creating tasks. These models, such as the LLMa-2, advance the state of the art in many applications. However, their extensive computational requirements and resource efficiency lead to challenges in terms of deployment, especially when computational resources are limited (Naveed et al., 2024). Fine-tuning techniques have been investigated to solve this problem. Efficiency is not just about adapting the pre-trained model to a specific task, but also about optimizing its performance while minimizing the computational load. In this context, the combination of quantization and low-order adaptation techniques provides an effective way to strike a balance between model performance and performance specificity (Gema et al., 2023).
Quantization will reduce the accuracy of the sample parameters, thus reducing the overall memory footprint and computational requirements. Lower-order transformations, on the other hand, use matrix factorization techniques to expand the representation model and thus support better thinking. The combination of quantization and low-pass transformation can improve performance without compromising the model's ability to capture complex descriptions (Liang et al., 2021). This report provides an in-depth study of the new LLMa-2 method used to improve the quality of negative effects through a combination of quantization and low-order transformation. By exploring this new method, we aim to demonstrate its effectiveness in increasing the validity of the model for deployment in limited space. Through a comprehensive analysis, we will evaluate the impact of quantization and low-level transformation on model performance, exploring the trade-off between computational efficiency and performance specifications. Goals of the project include: Impact of vulnerability performance on the complexity and resource consumption of the LLMa-2 model. Using quantized low-order adaptation to examine how quantized language optimization can improve the efficiency of LLMa-2 without sacrificing overall performance and using fewer resources.
The report is designed to provide a better understanding of the repair process. Chapter 2 presents the literature review supporting this research in the broader field of qualitative measurement and quantitative methods in NLP. Chapter 3 provides an overview of the method, explaining the steps involved in measuring the quality of the test and the evaluation of poor quality in determining LLMa-2. Section 4 presents and discusses the results in terms of performance improvement and their impact on performance. Finally, Chapter 5 concludes the report by summarizing the findings and offering recommendations for future research. In exploring the

intersection of dynamic scaling and quantization techniques for language structures, the insights gained from this research are expected to contribute to ongoing efforts to develop effective and efficient NLP models.

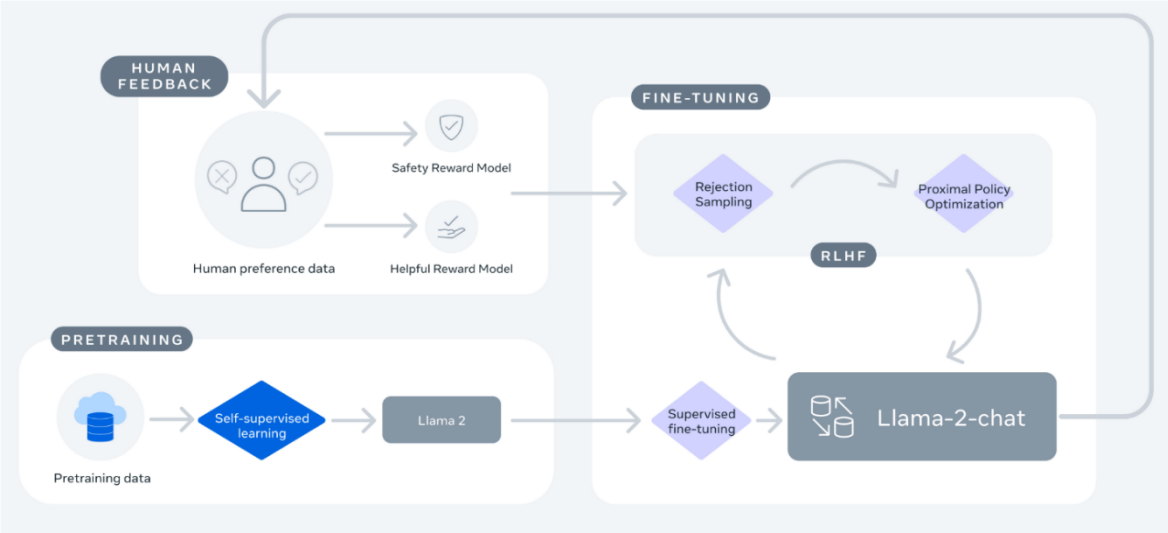


Figure 1.1: Fine-Tuning Process and LLaMA-2-Chat

The training process for LLaMA 2 is structured into three primary phases: pretraining, supervised fine-tuning, and reinforcement learning from human feedback (RLHF). During pretraining, LLaMA 2 is exposed to extensive datasets using self-supervised learning to understand language patterns. In the next phase, supervised fine-tuning is conducted to tailor the model for specific applications, such as conversational AI, which results in models like LLaMA-2-Chat. Lastly, RLHF is utilized, where human input is leveraged to create reward models that guide the model's development. Techniques like rejection sampling and Proximal Policy Optimization (PPO) help refine the model's safety and usefulness, enhancing the quality of its responses over time.

Chapter 2

Related Work

The quantization of LLMs has mostly concentrated on inference time quantization. The main methods for checking the quality of 16-bit LLM focus on checking abnormal features (e.g., SmoothQuant (Xiao et al., 2023) and LLM.int8(), although they are widely used, large language models (LLMs) require a large amount of GPU RAM for inference. The algorithm for Int8 matrix multiplication that the authors provide, LLM.int8(), is made especially for feed-forward and attention projection layers in transformers. This technique maintains full precision performance while cutting the memory required for inference in half. Understanding and navigating emergent characteristics in transformer language models is where the important discoveries are found. The authors create a two-step quantization process that combines a unique mixed-precision decomposition approach with vector-wise quantization. Inference in LLMs with up to 175B parameters may now be done without sacrificing efficiency thanks to LLM.int8(). (Dettmers et al., 2022)), while other methods use different methods. Lossy quantization studies the trade-off of the rounding (Pope et al., 2022; Dettmers & Zettlemoyer, 2023; Zeng et al., 2023) or how to improve the decision balance to improve the accuracy of quantization (Frantar et al., 2023). The only other study that examines backpropagation with quantized weights at a scale larger than 1B parameters is SwitchBack layers (Wolf et al., 2020). Although here used LoRA method freezes pre-trained model weights and injects trainable rank decomposition matrices into each layer of the Transformer architecture. By doing so, it achieves comparable or better performance than full fine-tuning while drastically reducing the number of trainable parameters. Unlike adapters, LoRA incurs no additional inference latency. (Hu et al., 2021), many other PEFT methods have been proposed, such as prompt tuning (Qin & Eisner, 2021), tuning embedding layer inputs (An et al., 2022), In order to train the model for new tasks, PEFT only requires a minimal number of parameters, and it not only provides improved accuracy but also significantly reduces computing costs, adjust hidden state (IA3) It achieves better performance with fewer extra parameters by scaling activations by learning vectors, a simple recipe called T-Few, applicable to new tasks without task-specific tuning or modifications. T-Few achieves super-human performance on completely unseen tasks, outperforming the state-of-the-art by 6% absolute on the RAFT benchmark. (Liu et al., 2022), add all layers (Houlsby et al., 2019), adjust bias (Zaken et al., 2022), learning mask over weights based on Fisher's data (Sung et al., 2021), and combination of various methods.A unique approach to fine-tuning large-scale language models that better balances the quantity of trainable

parameters with task performance is the Compacter technique. Compacter adds task-specific weight matrices to the weights of a pretrained model, as opposed to standard fine-tuning, which necessitates updating all model weights. These matrices are generated in an efficient manner by utilizing concepts from parameterized hypercomplex multiplication layers, low-rank optimization, and adapters. Compacter provides performance similar to ordinary fine-tuning on GLUE and exceeds it on SuperGLUE and low-resource settings by training only a small portion of the pretrained model's parameters (Mahabadi et al., 2021).The QLoRA technique offers a memory-saving finetuning method that is highly effective. It maintains the whole 16-bit finetuning work performance while enabling the finetuning of a 65B parameter model on a single 48GB GPU. To do this, QLoRA backpropagates gradients into Low Rank Adapters (LoRA) from a frozen, 4-bit quantized pretrained language model. On the Vicuna benchmark, the Guanaco model family—the best model family derived from QLoRA—performs better than all previously published models. Guanaco just needs 24 hours of fine-tuning on a single GPU to achieve 99.3% of ChatGPT's performance level. Through the use of double quantization, paged optimizers, and a new data type called 4-bit NormalFloat (NF4), QLoRA offers a number of advances to conserve memory without compromising speed. The LoRA adapter can achieve full 16-bit fine-tuning performance (Dettmers et al., 2023). Address the issue of effective generative inference for Transformer models, particularly in difficult situations involving big, complex models. The objective is to handle lengthy sequences with low latency inference. Based on application needs, the research chooses optimal multi-dimensional partitioning algorithms suited for TPU v4 slices and presents a straightforward analytical model for inference efficiency. The authors create a new Pareto frontier in terms of latency and model FLOPS utilization (MFU) tradeoffs for 500B+ parameter models by combining these methods with low-level improvements. They also show that proper segmentation allows scalability to up to 32x longer context durations. The suggested method preserves memory economy while achieving outstanding performance (Pope et al., 2022). Fine-Tuning the Instructions : Instruction finetuning employs input-output pairings of different data sources to finetune a pretrained LLM to create the output given the input as a prompt. This helps the LLM follow the instructions given in the prompt. Among the methods and resources are MetaICL (Min et al., 2022),

Table 2.1: Under-evaluation of the CrowS dataset. A lower score indicates less likelihood of creating an unbiased system. guanaco follows a different model from the LLaMA base model.

	LLaMA-65B	GPT-3	OPT-175B	Guanaco-65B
Gender	70.6	62.6	65.7	47.5
Religion	79.0	73.3	68.6	38.7
Race/Color	57.0	64.7	68.6	45.3
Sexual orientation	81.0	76.2	78.6	59.1
Age	70.1	64.4	67.8	36.3
Nationality	64.2	61.6	62.9	32.4

Continued on next page

Table 2.1: Under-evaluation of the CrowS dataset. A lower score indicates less likelihood of creating an unbiased system. guanaco follows a different model from the LLaMA base model. (Continued)

Disability	66.7	76.7	76.7	33.9
Physical appearance	77.8	74.6	76.2	43.1
Socioeconomic status	71.5	73.8	76.2	55.3
Average	66.6	67.2	69.5	43.5

MetaTuning (Zhong et al., 2021), InstructGPT (Ouyang et al., 2022), PromptSource (Bach et al., 2022), Self-instruct (Yao et al., 2022) , UnnaturalInstructions (Honovich et al., 2022) and Self-instruct-GPT-4 (Peng et al., 2023a).
Chatbots : Many learning tracking models have been developed based on conversational chatbots, often using Human Feedback Enhanced Learning (Christiano et al., 2023) or by generating data from existing models to introduce AI models using feedback (Bai et al., 2022). Methods and databases include AnthropicHH (Askell et al., 2021), Open Help (Köpf et al., 2023), LaMDA (Touvron, Lavril, et al., 2023), and Sparrow (Glaese et al., 2022). In this report do not use any additional learning, but best model, Guanaco, is fine-tuned against many interactions in the Open Assistant dataset designed to train RLHF (Köpf et al., 2023). A method that uses GPT-4 to evaluate chatbots instead of using expensive human annotations has been developed (Peng et al., 2023b).

Chapter 3

Methodology

In this section, I discuss the methodology of Llma-2 fine tuning model. This section begins by introducing the methods I used.

3.1 Dataset Description

In the optimization of the base model (Llama-2-7b-chat-hf from NousResearch as the model), use a small dataset called mlabonne/guanaco-llama2-1k. The data I used was a subset of Open Assistant datasets, the Guanaco dataset, a good set of Open Assistant datasets with about 10,000 dialogues. With the help of the PEFT library, can use the new QLoRA method to set the fit on top of the frozen 4-bit model. To provide a large-scale comparison of studies, OpenAssistant Conversations was published, a corpus of human-generated, human-annotated assistant conversations containing 161,443 words in 35 languages, annotated with 461,292 good ratings, producing more than 10,000 conversation trees with complete annotations. The corpus is the result of an international effort involving more than 13,500 volunteers. This file contains the messages tree. As the root node of every message tree, the first prompt message may have several child messages as replies, and these child messages may also have several replies.
A role attribute exists for every message, and it can be either "assistant" or "prompter". "Prompter" and "assistant" play opposite roles in discussion threads from prompt to leaf node.

Dataset Sample:

```
<s>[INST] In England, what vegetable is referred to as a rocket?
[/INST] The species name for "rocket" is "Eruca vesicaria", and
it's also sometimes called "eruca". However, other countries
have different names for it: * Arugula - America * Garden Rocket
or Rocket - Britain, Australia, South Africa, Ireland, New Zealand
Other names include "ruchetta", "rucola", "rucoli", "rugula",
"colewoort", and "roquette". The word transferred from Latin to
Italian to English, which might explain the variants with similar
pronunciation. </s>

<s>[INST] Cuál es el país más grande del mundo? [/INST] El país
```

```
más grande del mundo en superficie es Rusia, con más de 17.000.000
km², pues representa aproximadamente el 10% de la totalidad de la
superficie terrestre. </s>

<s>[INST] I want you to translate the json documents I provide into
a schema in YAML format [/INST] Sure, I can help with that! Please
provide the JSON document you would like me to translate into
YAML schema. </s><s>[INST] Sure: ‘‘‘json { "name": "John Doe",
"age": 30, "email": "johndoe@example.com", "address": { "street":
"123 Main St", "city": "Anytown", "state": "CA", "zip": "12345" },
"phoneNumbers": [ { "type": "home", "number": "555-555-1234" },
{ "type": "work", "number": "555-555-5678" } ] } ‘‘‘
[/INST] Sure thing! Here is how you can translate the given JSON
to YAML: ‘‘‘yaml name: John Doe age: 30 email: johndoe@example.com
address: street: 123 Main St city: Anytown state: CA zip: ’12345’
phoneNumbers: - type: home number: ’555-555-1234’ - type:
work number: ’555-555-5678’ ‘‘‘ </s>
```

3.2 Data Prepossessing

The prompt template for the conversation models in Llama 2 is as follows. I use the reformat dataset with the 1k sample (called mlabonne/guanaco-llama2-1k) dataset as instructed, using the Llama 2 template. The “guanaco-llama2-1k” dataset, which has been processed to comply with the Llama 2 prompt format and comprises 1000 samples, should be loaded from the Hugging Face hub. Load the tokenizer and pretrained base model as well.

```
<s>[INST] <<SYS>>
    system_prompt
<</SYS>>
    user_message
[/INST] Model answer </s>
```

3.3 Model Architecture

There are several elements that go into making LLaMA-2 useful.With a few adjustments, LLaMa-2 uses the LLaMA model architecture.

Normalization:

Layer normalization is used by the majority of transformer architectures and is done after each layer inside the transformer block. However, LLaMA substitutes this with a variation known as Root Mean Square Layer Normalization, or RMSNorm for short. This is a more straightforward form of layer normalization that has been demonstrated to enhance generalization and training stability (Zhang & Sennrich,

2019). The formula for RMSNorm is as follows.

$$\overline{a_i} = \frac{a_i}{\text{RMS}}, \text{ where RMS} = \sqrt{\frac{1}{n}\sum_{n=1}^{\infty} a_i^2} \tag{3.1}$$

The input vector **a** consists of n elements: $a_1, a_2, \ldots, a_n$. The Root Mean Square (RMS) of the elements in the vector **a** is calculated. Each element a_i in the input vector **a** is then divided by the RMS to normalize it. The input vector a is transformed to have a mean root square of 1. Each element is scaled according to the RMS of the entire vector. This normalization technique helps in stabilizing the training of neural networks by ensuring consistent scaling of inputs.
For LLaMA, a pre-normalization variation of RMSNorm is used, which means that, contrary to what the below picture illustrates, normalization is done before the main layers in a transformer block. RMSNorm outperforms layer norm by 10–50% while maintaining performance levels that are similar.

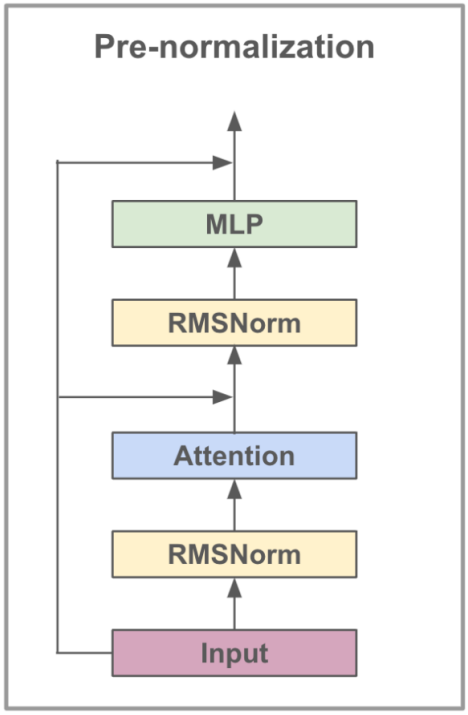


Figure 3.1: Pre-normalization variant of the normal transformer block

The pre-normalization variant of a transformer block introduces RMSNorm (Root Mean Square Normalization) before the attention and MLP layers, unlike traditional transformers that apply normalization after these operations. In this setup, inputs like word embeddings are first normalized by RMSNorm before being passed through the self-attention mechanism, which captures token relationships. Following attention, another RMSNorm step is applied to stabilize inputs for the Multi-Layer Perceptron (MLP). Residual connections are employed to combine the original inputs with the outputs, helping to maintain gradient flow and retain information, thereby improving training stability with pre-normalization.

Activation functions:

In its feed-forward layers, LLaMA models use the SwiGLU activation function, as opposed to the typical ReLU function used by most neural networks. One way to formulate the SwiGLU activation function is as follows.

$$SwiGLU(x) = Swish(xW).xV \tag{3.2}$$

where:

- x is the input to the activation function.
- W and V are weight matrices.

The sigmoid function applied to the scaled input βx is given by:

$$Swish(x) = x.Sigmoid(\beta x) \tag{3.3}$$

where:

- β is a parameter that can be learned or set as a constant.

.
SwiGLU is the element-wise product of two linear transformations (one with a Swish activation added to it) of the input x. Even when the amount of compute is held constant, this activation function has been found to produce better performance than other activation functions, despite requiring three matrix multiplications, making it more computationally expensive than a typical activation function like ReLU (Shazeer, 2020).

Rotary location Embedding (RoPE):

LLaMA models use a RoPE (Su et al., 2023) scheme, which seeks a balance between the absolute and relative location of each token in a sequence, in place of absolute or relative positional embeddings. With this position embedding method, relative location information is added straight into the self-attention procedure, while absolute position is encoded via a rotation matrix. Since RoPE embeddings work well on tasks with larger sequence lengths, several LLMs have embraced this technique (e.g., PaLM (Chowdhery et al., 2022) and Falcon (Institute, 2023)).
There are a few significant architectural modifications between LLaMA-2 and its predecessor. Initially, 4K tokens are used to train LLaMA-2 (before, LLaMA was trained using a 2K context length). Furthermore, as can be seen below, LLaMA-2 uses GQA (Ainslie et al., 2023) in every layer. This change is intended to accelerate the LLaMA-2 models' inference process.
A modified form of the typical multi-head causal self-attention is called GQA. In GQA, as previously said, we aggregate the N total self-attention heads into shared key and value heads. Using a common key and value projection over all N heads, this method interpolates between multi-query attention and vanilla multi-headed self-attention.
For pre-training, LLaMA and LLaMA-2 only use data from publicly available sources. This decision was taken specifically to guarantee that anyone with the necessary computing power could freely duplicate the pre-training procedure. However, LLaMA-2

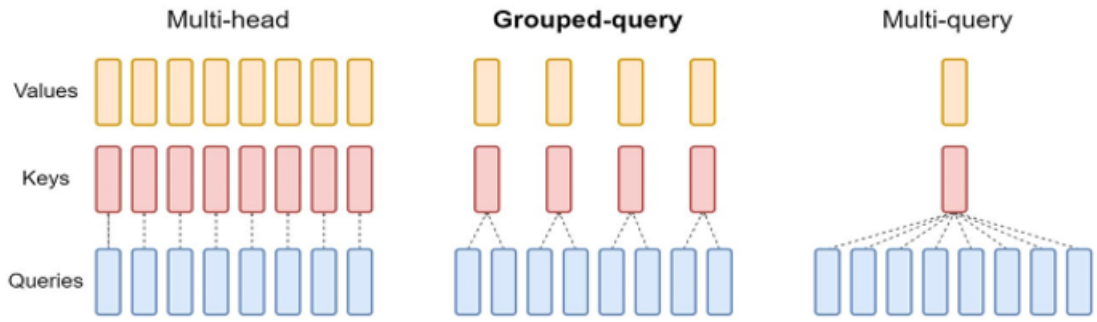


Figure 3.2: Overview of grouped-query method

raises the pre-training dataset size by 40% and adopts a new blend of pre-training data (i.e., sources that are known to be true and high-quality are sampled more strongly than in LLaMA). With this modification, the model may learn from more data during pre-training, which enhances LLAMA-2's knowledge base.
Important characteristics like helpfulness and safety that have to be promoted in the final model are highlighted by LLaMA-2. It's interesting to note that LLaMA-2 starts focusing on safety measures and data quality during the pre-training phase. Pre-training emphasizes more data from reliable or factual sources, while excluding sites that are known to include high quantities of personal information from the pre-training collection. To reduce the degree of bias within LLaMA-2, writers also assess a number of variables, including the representation of diverse demographic groups and toxicity levels; refer to the previous section for more information. To put it simply, LLAMA-2's pre-training dataset is meticulously curated and assembled.
The majority of open-source models were only aligned using supervised fine-tuning (SFT) on publicly accessible datasets before the release of LLaMA-2. Proprietary models, such as ChatGPT and GPT-4, on the other hand, typically take a more comprehensive approach, honing the LLM through copious amounts of conversation and human feedback. In an effort to bridge this gap, LLaMA-2 is refined using a sizable dataset in a way akin to proprietary models, resulting in the LLaMA-2-Chat model, which is tailored for applications including discussion; refer above. The alignment procedure is extremely private for most closed-source models; however, LLaMA-2 aims to increase transparency and simplify high-quality alignment for open-source research endeavors.

Alignment:

Teaching an LLM to provide an output that is in line with human needs is known as alignment. While the pre-training phase concentrates on precisely predicting the next token, alignment fine-tunes the model to achieve a range of various objectives; see below. Typically, SFT and/or RLHF are used. Remarkably, pre-training provides LLMs with the majority of their knowledge base, whereas alignment instructs them on the proper output style or structure.
Alignment, for instance, may be done to lessen hallucinations, steer clear of risky queries, adhere to thorough directions, and more. It is possible to synchronize many objectives at once. Regarding LLaMA-2, the authors concentrate on enhancing the subsequent standards while aligning: Helpfulness: The model complies with user demands and delivers the data they ask for. Safety: The model stays away

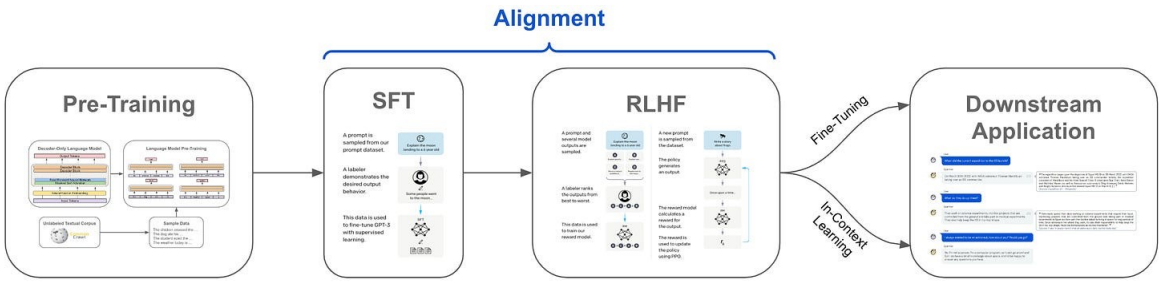


Figure 3.3: Overview of Teaching an LLM

from "unsafe" answers. Pursuing these objectives through alignment results in the creation of the LLaMA-2-Chat model, which is tailored for use cases including discussion. Since LLaMA-2-Chat is aligned and fine-tuned using both RLHF and SFT, we will now take a closer look at each of these fine-tuning stages separately.

Supervised Fine-Tuning (SFT):

SFT is the initial stage in the LLaMA-2 fine-tuning procedure. This method is easy to comprehend since it trains the model with a next-token prediction target applied to each of these instances' replies. It is based on a (supervised) next-token prediction objective that is almost the same as pre-training. This method of fine-tuning the model teaches the LLM to mimic behavior that is modeled in the answers of the fine-tuning dataset. As a result, any desired characteristics—or alignment goals—found in this data also appear in the model.

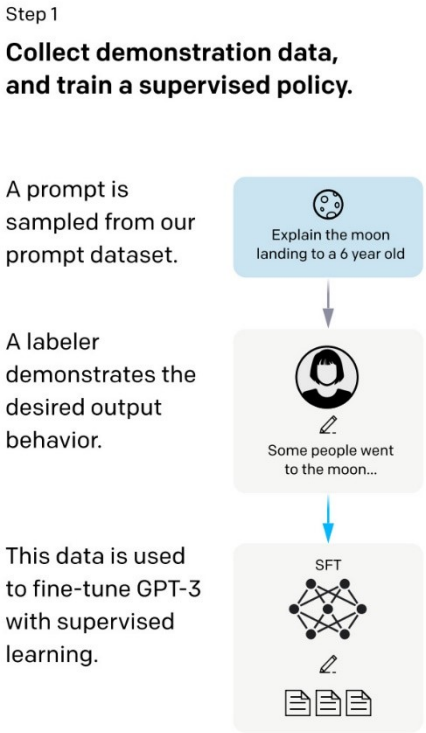


Figure 3.4: Supervised Fine-Tuning

Notably, LLaMA-2-Chat carries out two distinct SFT processes. While the second

step trains over a smaller, carefully curated dataset of significantly higher quality, the first phase trains over a bigger amount of publicly available data.
The first step in LLaMA-2's alignment procedure is fine-tuning over a publicly accessible SFT dataset. However, LIMA (Zhou et al., 2023) found that such public data is lacking in both quality and variety, which is problematic.
For the second stage of SFT, the small set of clear instructions utilized in (Touvron, Martin, et al., 2023) is quite efficient. It's interesting that the authors point out that gathering additional data for SFT has declining returns because i) the improved model can provide its own data for SFT and ii) adding more annotation to RLHF clearly improves the results. Overall, (Touvron, Martin, et al., 2023) shows that selecting a small dataset with stricter quality requirements is a useful approach for SFT, which is consistent with research from (Zhou et al., 2023).

Reinforcement Learning from Human Feedback (RLHF):

SFT is a very powerful tool, but managing and curating a dataset that really reflects alignment goals may be challenging. RLHF adopts a more direct strategy, adjusting the LLM based on input from people who have seen the model's results. Initially, human annotators are requested to create prompts for the LLMs, which may then be chosen according to alignment criteria (such safety and helpfulness). Subsequently, the LLM is employed to produce two or more replies for every prompt, which are then ranked by human annotators according to their level of excellence. In this case, the degree to which the answer satisfies the alignment criterion is usually used to evaluate "quality." After that, the LLM may be optimized via RLHF using these human preference scores as a basis. LLAMA-2 gathers information about human preferences using a binary protocol. Annotators develop two model replies for every prompt they write, and they select the preferred response based on safety and helpfulness standards. Interestingly, every human preference collecting case concentrates on a single alignment criterion. RLHF data is gathered in many batches. Every week, a fresh batch of data is gathered, and the authors of (Touvron, Martin, et al., 2023) iteratively refine the LLaMA-2-Chat model (using RLHF) in response to each new batch. As such, throughout the alignment procedure, several consecutive "rounds" of RLHF are carried out.
The image outlines two key steps in the Reinforcement Learning from Human Feedback (RLHF) process:
Step 2: Collect Comparison Data and Train a Reward Model – A prompt is given to the model (e.g., "Explain the moon landing to a 6-year-old"), which generates several outputs. A human labeler ranks these outputs by quality. This ranking data is then used to train a reward model, which learns human preferences by assigning higher rewards to better outputs.
Step 3: Optimize a Policy Using Reinforcement Learning (RL) – The model generates a new response to a different prompt (e.g., "Write a story about frogs"). The reward model evaluates the output and assigns a reward based on human preferences. This reward is used to update the model's policy using Proximal Policy Optimization (PPO), allowing the model to improve and align better with human feedback over time.

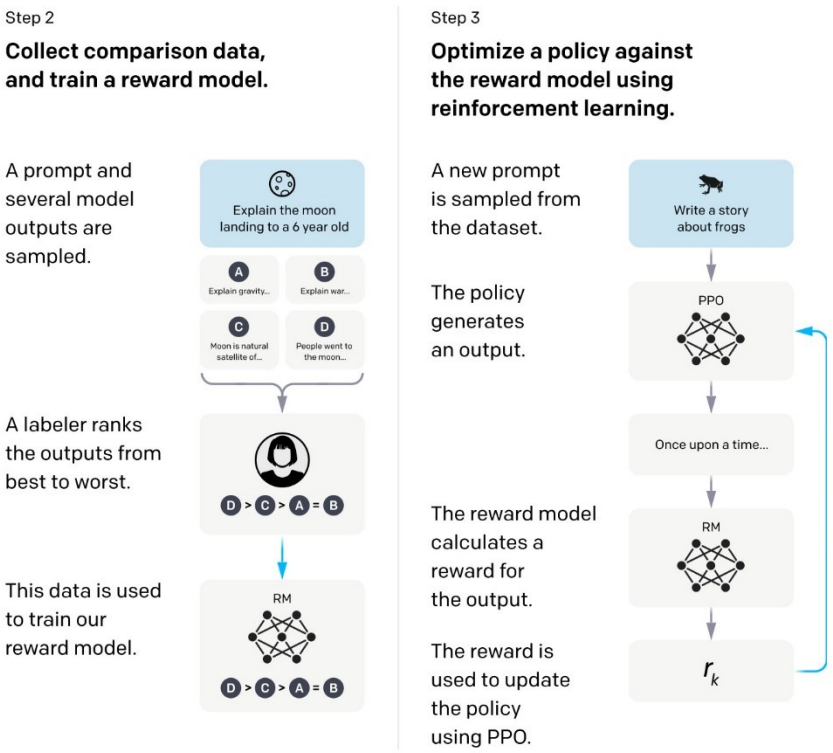


Figure 3.5: Reinforcement Learning from Human Feedback

Training the reward model:

A reward model is trained on this set of data following the collection of human input. The reward model predicts a preference score by using a prompt and response—along with the whole conversation history—as input. The architecture and weights of the reward model are often the same as those of the LLM; however, instead of a classification head for predicting the next token, a regression head is used for preference estimation, and the model is adjusted using preference data.
This figure illustrates two main approaches for training large language models (LLMs): Next-Token Prediction (left) and the Reward Model Structure (right). In Next-Token Prediction, a decoder-only transformer processes input tokens and their positional embeddings, applying masked self-attention to predict the next token in a sequence. The predicted token is then compared to the actual next token to calculate the loss, which is used to update the model. The Reward Model Structure shares a similar foundation but incorporates human feedback. After processing the tokens, it produces a sequence embedding, which is passed through a regression layer to generate a reward score. This score is based on human preferences, and the model computes loss by comparing predicted rewards to this feedback, optimizing its performance using reinforcement learning.
To maximize (automatically produced) human preference ratings across a large dataset, the LLM may be optimized using reinforcement learning thanks to the reward model's automation of generating an accurate preference score for prompt-response combinations. Using binary preference data, we create a training goal that requires the preferred example to score greater than the alternative in order to train the reward model.

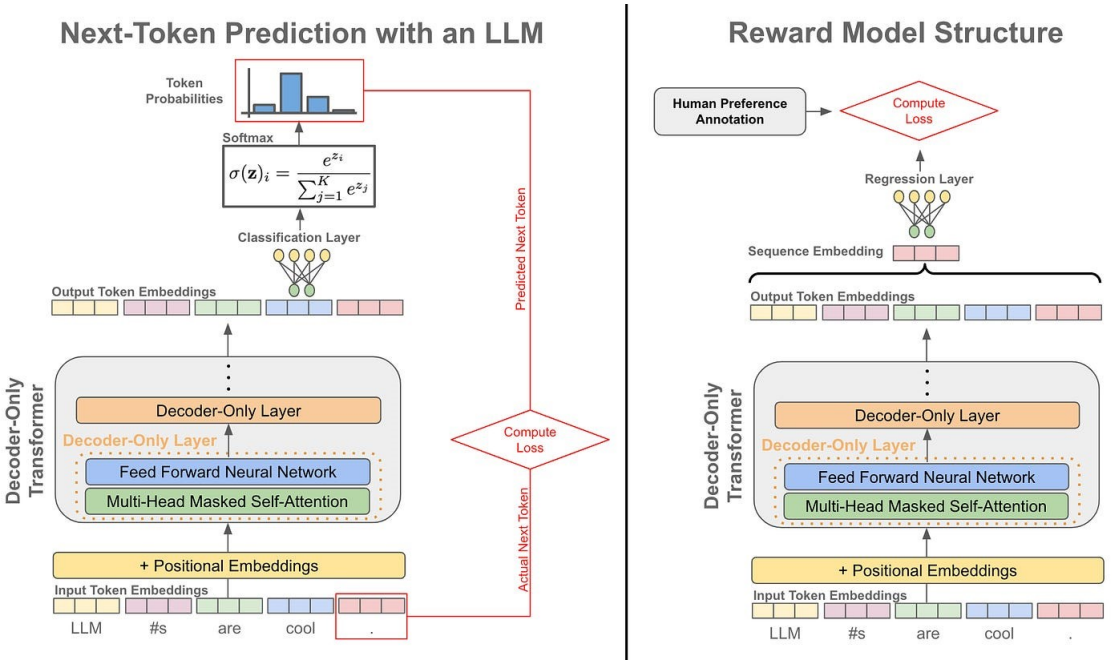


Figure 3.6: Reward model

$$\mathcal{L}_{\text{ranking}} = -\log(\alpha(r_\theta(x, y_c) - r_\theta(x, y_r))) \tag{3.4}$$

where:

- L_{ranking} is the ranking loss.
- r_θ is the reward model with parameters θ.
- x is the input prompt.
- y_c is the correct (preferred) response.
- y_r is the rejected (alternative) response.
- α is a scaling factor.

The loss function encourages the reward model to score the correct response higher than the rejected one by maximizing the difference $r_\theta(x, y_c) - r_\theta(x, y_r)$. The negative logarithm stabilizes training and penalizes incorrect rankings.
It is important to keep in mind, nevertheless, that the preference data gathered in (Touvron, Martin, et al., 2023) is not exclusively binary. We can identify replies that are substantially or moderately better than others thanks to our more detailed data. We may easily add a margin—that is, a set number assigned to each of the much better, better, somewhat better, or negligibly better categories—to the loss displayed above in order to capture this more specific information. When there is such a margin, the reward model is pushed to give greater score gaps across replies with significant preference differences.

$$\mathcal{L}_{\text{ranking}} = -\log(\alpha(r_\theta(x, y_c) - r_\theta(x, y_r) - m(r))) \tag{3.5}$$

where:

- L_{ranking} is the ranking loss.
- r_θ is the reward model with parameters θ.
- x is the input prompt.
- y_c is the correct (preferred) response.
- y_r is the rejected (alternative) response.
- α is a scaling factor.
- $m(r)$ is a margin function that accounts for significant preference differences.

The loss function encourages the reward model to score the correct response higher than the rejected one by maximizing the difference $r_\theta(x, y_c) - r_\theta(x, y_r)$, while ensuring significant score gaps for responses with substantial preference differences using $m(r)$. The negative logarithm stabilizes training and penalizes incorrect rankings.

3.4 Fine tuning model

Load the dataset as it is first defined. Although the dataset has already been preprocessed in this case, I normally reformat the prompt at this point. Setting up bits and bytes for 4-bit quantization follows.
Next, to speed up training, load the Llama 2 basic model in 4-bit precision using the compute dtype "float16" from Hugging Face. To address the fp16 problem, you should also load the tokenizer from Hugginface and change padding_side to "right."
Traditionally, PLMs had to have all of their parameters updated. This takes a lot of data and is computationally costly.The far more effective PEFT technique simply modifies a tiny portion of the model's parameters. Large LLM models may be effectively fine-tuned on consumer hardware while maintaining great performance thanks to 4-bit quantization via QLoRA. This significantly enhances usability and accessibility for practical applications.A pre-trained language model is quantized to 4 bits via QLoRA, after which the parameters are frozen. The model is then enhanced with a few trainable Low-Rank Adapter layers.
Gradients are backpropagated into just the Low-Rank Adapter layers during the fine-tuning process via the frozen 4-bit quantized model. Thus, just the adapters are changed, while the pretrained model as a whole stays constant at 4 bits. Besides, the model performance is not harmed by the 4-bit quantization.
The figure presents three fine-tuning strategies for transformer models: Full Fine-Tuning, LoRA, and QLoRA, each balancing resource consumption and performance. Full Fine-Tuning updates all model parameters at 16-bit precision, demanding significant memory and computational power. LoRA mitigates these demands by introducing trainable adapters at 16-bit precision, only updating part of the model, thus lowering memory and compute requirements. QLoRA goes a step further by quantizing the model to 4-bit precision, significantly reducing memory usage and utilizing paging to offload excess memory to the CPU. Both LoRA and QLoRA prioritize efficient parameter updates, making them ideal for fine-tuning large models on limited hardware resources.
In our situation, I use BitsAndBytes to achieve 4-bit quantization using NF4 type setting.

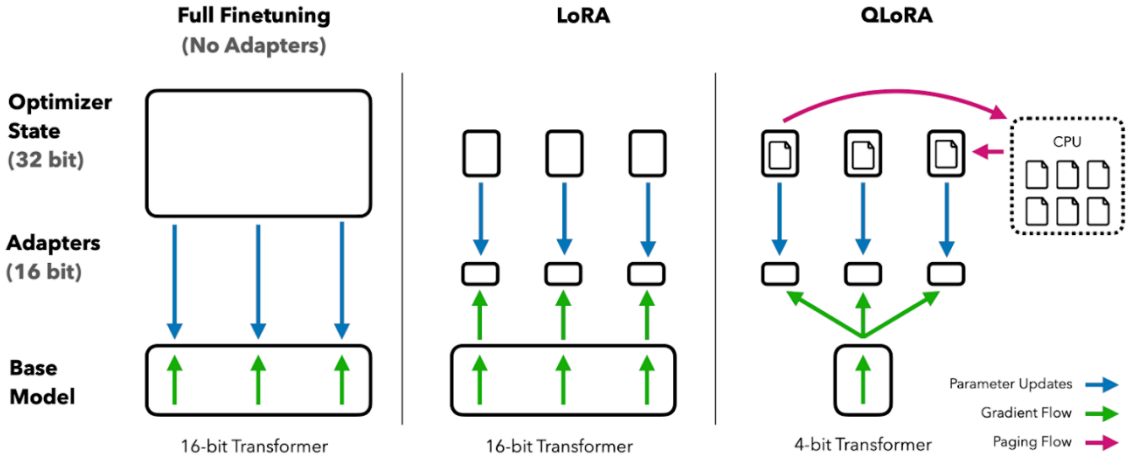


Figure 3.7: Fine tuning model

3.5 Model Train & Training Parameter

SFT is a key step in RLHF. The TRL library from HuggingFace provides an easy-to-use API to create SFT models and train them on your dataset with just a few lines of code. It comes with tools to train language models using reinforcement learning, starting with supervised fine-tuning, then reward modeling, and finally PPO.
We will provide SFT Trainer the model, dataset, Lora configuration, tokenizer, and training parameters. Finally, use .train() to fine-tune the Llama 2 model on a new dataset.
The list of hyperparameters that may be adjusted to enhance the training procedure is provided below:

Table 3.1: The list of hyperparameters & Description

Hyperparameters	**Description**
output_dir	The output directory is where the model predictions and checkpoints will be stored.
num_train_epochs	Two training epoch.
fp16/bf16	Disable fp16/bf16 training.
per_device_train_batch_size	Batch size per GPU for training.
per_device_eval_batch_size	Batch size per GPU for evaluation.
gradient_accumulation_steps	This refers to the number of steps required to accumulate the gradients during the update process.
gradient_checkpointing	Enabling gradient checkpointing.
max_grad_norm	Gradient clipping.
learning_rate	Initial learning rate.
weight_decay	Weight decay is applied to all layers except bias/LayerNorm weights.
Optim	Model optimizer (AdamW optimizer).
lr_scheduler_type	Learning rate schedule.
max_steps	Number of training steps.

warmup_ratio	Ratio of steps for a linear warmup.
group_by_length	This can significantly improve performance and accelerate the training process.
save_steps	Save checkpoint every 25 update steps.
logging_steps	Log every 25 update steps.

3.6 Evaluation process

An automated assessment statistic called BERTScore is used to gauge how well text generating algorithms perform. In contrast to other widely used approaches that calculate syntactical similarity at the token level, BERTScore concentrates on calculating semantic similarity between reference and hypothesis tokens.
Existing methods like ROUGE and BLEU metrics, which rely on the syntactic overlap between hypothesis and reference by taking into account unigrams, bigrams, etc., may be used to assess the quality of text generation algorithms. However, taking into account their limitation—the use of the same term in both the hypothesis and the reference—and their inability to interpret the semantics in question, Introducing BERTScore: the goal is to compare what you have made with what was intended to be generated by first understanding their respective meanings.

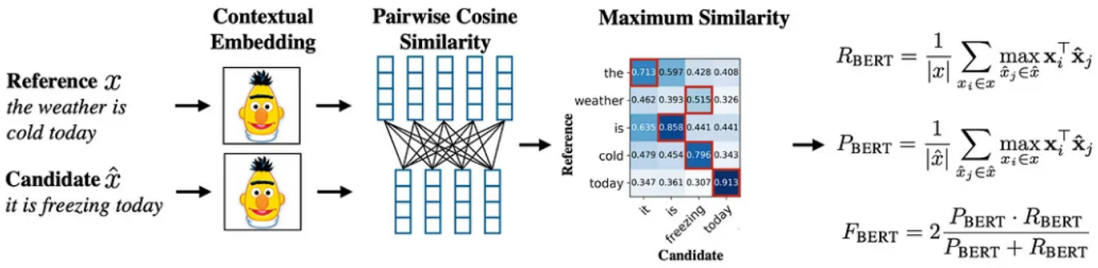


Figure 3.8: BERTScore: Evaluating Text Generation with BERT

The BERTScore method assesses text generation by comparing reference and candidate texts using BERT embeddings and cosine similarity. First, both the reference text (e.g., "the weather is cold today") and the candidate text (e.g., "it is freezing today") are passed through a pre-trained BERT model to generate contextual embeddings for each word, capturing their meanings within the sentence context. Then, pairwise cosine similarity is calculated between the embeddings of each word in the reference and candidate texts, forming a similarity matrix. Unlike traditional word matching, BERTScore evaluates the contextual meanings of words, making it particularly useful for tasks like translation, summarization, and paraphrasing, where semantic understanding is crucial.

BERTScore matches each token in x to a token in $\hat{x}$ to compute recall, and each token in $\hat{x}$ to a token in x to compute precision. Greedy matching is used to maximize the matching similarity score, where each token is matched to the most similar token in the other sentence. Precision and recall are combined to compute an F1 measure.

For a reference x and candidate $\hat{x}$ we can calculate the recall, precision, and F1 scores.
where:

- R_{BERT} is the recall score.
- P_{BERT} is the precision score.
- F_{BERT} is the F1 score.
- x_i and $\hat{x}_j$ are tokens from the reference and candidate sentences, respectively.
- $\cdot$ denotes the similarity score between tokens.

I also perform human evaluation on the generated text.

Chapter 4

System Design & Visualization

Discover detailed visualizations of GPU memory consumption and bit precision usage during LLaMA 2 operations and fine-tuning processes. The website provides real-time charts and analytics to help understand resource distribution and optimize model configurations. Use these insights to improve AI efficiency and overall performance.

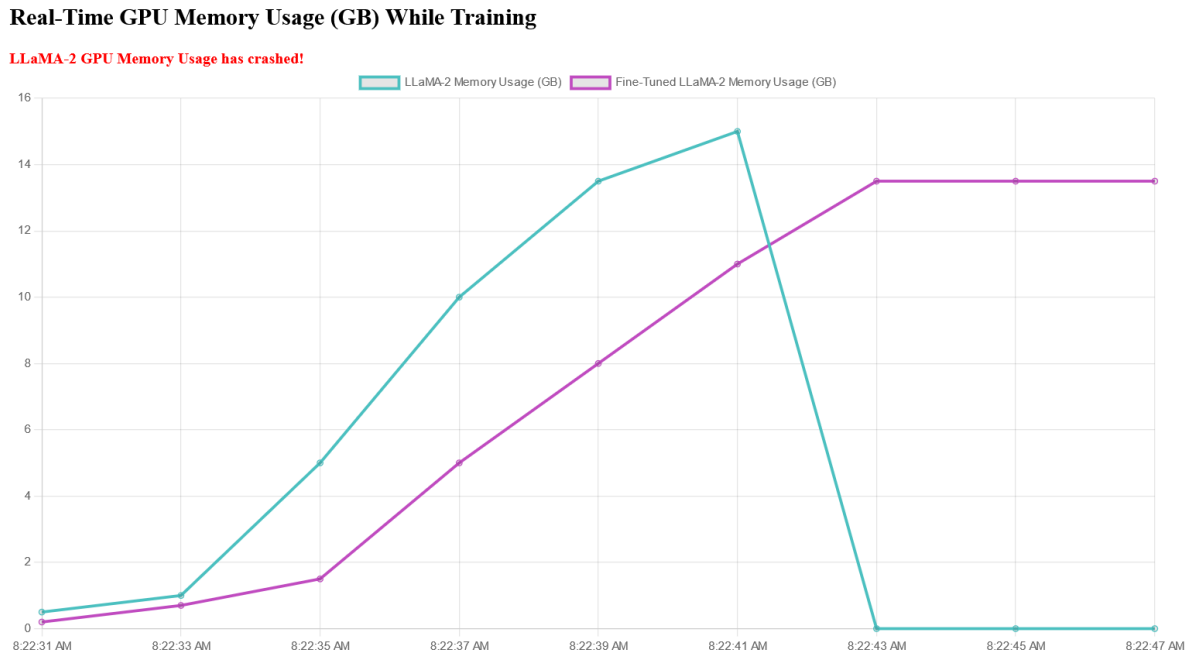


Figure 4.1: GPU memory for LLaMA-2 VS Fine-Tuned LLaMA-2 (using QLORA)

The graph shows the real-time usage of GPU memory for two scenarios: LLaMA-2 (in teal) and Fine-Tuned LLaMA-2 (in purple) during a training process. The x-axis represents time in seconds, while the y-axis represents memory usage in gigabytes (GB).
Here is the breakdown:
LLaMA-2 Memory Usage (Teal Line): Initially, increases steadily, reaching a peak around 14-15 GB. Suddenly, it drops to 0 GB, indicating a crash in GPU memory usage. This means that the model could no longer utilize GPU memory due to an issue (e.g., out-of-memory error).

Fine-Tuned LLaMA-2 Memory Usage (Purple Line): Starts with lower memory usage compared to LLaMA-2. Gradually increases and stabilizes at 13.5 GB, without any crashes complete the training process.

The warning message at the top, "LLaMA-2 GPU Memory Usage has crashed!", highlights the failure of LLaMA-2 when its memory usage abruptly drops to zero, while the fine-tuned model continues operating normally.

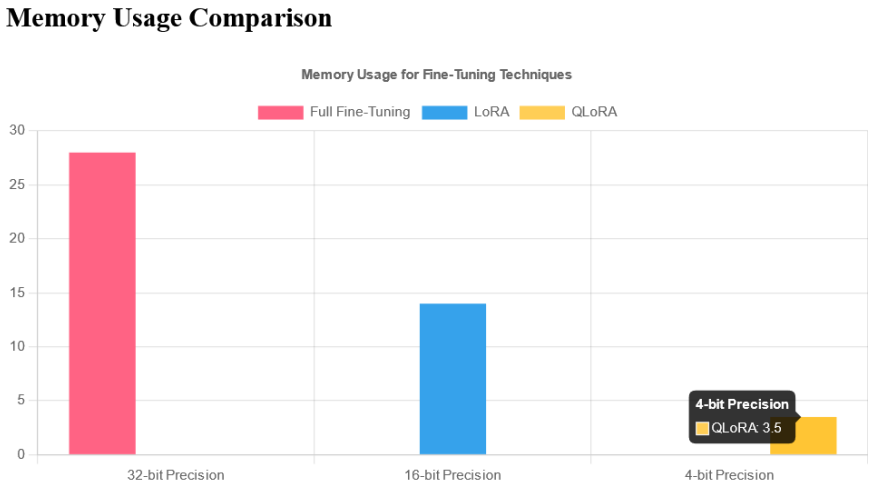


Figure 4.2: Memory Usage Comparison bit-wise

The chart shows comparing GPU memory usage across three fine-tuning techniques: Full fine tuning, LoRA (low-rank adaptation), and QLoRA (quantized LoRA). The x-axis represents different precision levels used during training, 32-bit precision, 16-bit precision, and 4-bit precision. The y-axis represents the usage of GPU memory in gigabytes (GB).

- **Full Fine-Tuning (in pink):** Shows the highest memory usage, particularly at 32-bit precision, where it reaches approximately 28-30 GB.
- **LoRA (in blue):** Significantly reduces memory usage, especially in 16-bit precision, to around 15 GB.
- **QLoRA (in yellow):** Demonstrates the lowest memory usage, achieving only 3.5 GB at 4-bit precision.

This visualization highlights the efficiency of QLoRA, especially with lower precision, in minimizing GPU memory usage during fine-tuning compared to the other techniques.
Demo of a website which creates using React and Django to visualize GPU memory usage in real time while training.
Here is a video link for GPU Utiliations: **Click here to watch the video!**

Chapter 5

Result Analysis

BertScore is a statistic that compares machine-generated text to ground truth or reference material to assess how well the text is. The measure is based on contextual embeddings produced by the well-known pre-trained language model BERT. Overall, these scores provide a quantitative assessment of the quality of the generated text in terms of precision, recall, and F1 score.

Table 5.1: Bert Scores in different candidate sets.

Scores for	Set 1 (llma2_400words)	Set 2 (llma2_200words)	Set 3 (finetune_llma2 _400words)	Set 4 (finetune_llma2 _200words)
Precision	0.874689	0.893544	0.866997	0.879702
Recall	0.888946	0.882551	0.884175	0.878541
F1 Score	0.881646	0.887907	0.875405	0.878989

In general, all candidate sets have relatively high scores across precision, recall, and F1 score, indicating good overall performance.

Human evaluation in generated text: For testing purposes, I asked some questions to both the pre-trained Llma-2 model and the model after fine-tuning. In some cases, the Llma-2 model provided no answer and displayed a blank response. However, after fine-tuning, the model was able to generate answers for these questions. When I prompted the pre-trained LLMA-2 model to generate a 400-word text answer for a particular question, it occasionally produced repeated sentences to reach the 400-word target. Similarly, when I asked the same model to generate a 200-word answer to the same question, it provided the same response without any modifications. In contrast, after fine-tuning, the model no longer exhibited the tendency to generate repeated sentences when aiming for a 400-word response. Furthermore, when generating a 200-word answer, it demonstrated the ability to modify sentences for improved diversity and content.
Question: Anaconda Distribution is the industry standard for data scientists developing, testing, and training on a single machine. Write a brief, 5 minutes, introduction about it.
Base Llma-2 with 400 words Answer:

```
Anaconda Distribution is a free and open-source distribution of the
Python programming language, which is widely used in data science and
```

scientific computing. It provides a comprehensive set of tools and libraries for data manipulation, visualization, and machine learning, as well as a consistent and easy-to-use interface for working with these tools.

Anaconda Distribution is designed to be easy to install and use, and it is available for both Windows and Linux operating systems. It includes a wide range of libraries and tools for data science, including NumPy, Pandas, Matplotlib, and Scikit-learn, as well as a number of other libraries and tools for data manipulation, visualization, and machine learning.

One of the key advantages of Anaconda Distribution is its ability to work seamlessly with a wide range of data sources, including CSV files, databases, and web APIs. It also provides a number of tools for data cleaning, transformation, and visualization, making it an ideal choice for data scientists who need to work with large and complex datasets.

In addition to its extensive library and toolset, Anaconda Distribution also provides a number of other features that make it an attractive choice for data scientists. For example, it includes a number of pre-built conda environments, which allow users to easily switch between different versions of Python and other libraries. It also provides a number of tools for debugging and testing code, as well as a number of other features that make it easier to work with Anaconda Distribution.

Overall, Anaconda Distribution is a powerful and flexible tool that is widely used in the data science community. Its comprehensive set of tools and libraries, combined with its ease of use and flexibility

Base Llma-2 with 200 words Answer:

Anaconda Distribution is a free and open-source distribution of the Python programming language, which is widely used in data science and scientific computing. It provides a comprehensive set of tools and libraries for data manipulation, visualization, and machine learning, as well as a consistent and easy-to-use interface for working with these tools.

Anaconda Distribution is designed to be easy to install and use, and it is available for both Windows and Linux operating systems. It includes a wide range of libraries and tools for data science, including NumPy, Pandas, Matplotlib, and Scikit-learn, as well as a number of other libraries and tools for data manipulation, visualization, and machine learning.

Llma2-fine-tuning with 400 words Answer:

Anaconda distribution is a powerful tool for data scientists and is widely used in the industry. It's an open source data science platform that combines a number of popular data science libraries into a single environment.

This makes it easy for users to develop, test, and train machine learning models without having to worry about installing and managing multiple libraries.

Anaconda distribution provides a lot of flexibility and customization options, allowing users to easily switch between different versions of popular data science libraries and to create custom environments. This makes it easy for data scientists to work on complex projects and to collaborate with other team members.

Some of the key features of Anaconda distribution include its ability to install and manage multiple versions of popular data science libraries, its flexibility in working with different programming languages, its ability to create custom environments, and its ability to perform efficient data processing and analysis. These features make it an ideal choice for anyone looking to work with big data.

Anaconda distribution also provides a number of tools and utilities to help users manage their projects and to collaborate with other team members. This includes tools for version control, data visualization, and data visualization.

In conclusion, Anaconda distribution is a powerful tool that has become the industry standard for data scientists. Its flexibility, customization options, and ability to work with multiple versions of popular data science libraries make it an ideal choice for anyone looking to develop, test, and train machine learning models on a single machine.

Anaconda distribution provides a lot of flexibility and customization options, allowing users to easily switch between different versions of popular data science libraries and to create custom environments. It also includes a number of tools and utilities to help

Llma2-fine-tuning with 200 words Answer:

Anaconda Distribution is a free and open-source distribution of the Python programming language, primarily geared towards data science and scientific computing. It is widely used in industries such as finance, healthcare, and government, and is considered to be the de facto standard for data science and machine learning.

Anaconda Distribution includes a comprehensive environment for data science, including a wide range of libraries and frameworks commonly used in data science, such as NumPy, SciPy, Matplotlib, and pandas. It also includes many other useful libraries and tools, such as Jupyter Notebook, Jupyter Lab, and Jupyter Hub.

One of the key benefits of Anaconda Distribution is its ability to provide a consistent environment across multiple platforms, incl-

uding Windows, MacOS

Chapter 6

Conclusion

In conclusion, although optimizing performance and offering significant advantages in terms of flexibility, fine-tuning LLaMA 2 models still necessitates careful consideration of resource needs, safety precautions, and responsible use. Fine-tuning a language model, such as the llma2 model, requires addressing several important considerations. Data bias is a major concern, as the use of biased data sets can perpetuate inequities in predictions. Overfitting for a given data set is a risk that highlights the need for fine-tuning the data and optimal training duration. Domain specificity is also a challenge. It is important to find the right balance while fine-tuning the model in a particular domain. The computational resources required for fine-tuning can be significant, especially for large models. It is important to choose the right criteria and align them with the goals of the work to avoid unwanted incentives.
Evaluate different bit-precisions, such as using 3-bit base models or different adapter methods. Besides LoRA, PEFT methods have also been shown to work well. However, it is unclear whether these methods scale to larger models. Finetuning after quantization seems to recover much of the information lost during quantization This can enable more aggressive quantization. For example, even a 3-bit GPTQ quantization of a basemodel with LoRA can give 16-bit full finetuning performance after finetuning (Bratanic, 2023). Function descriptions can improve fine-tuning efficiency by making the appearance of tokens in the answer more likely over the tokens present in the question, thus making the optimization problem simpler and fine-tuning more effective. However, there is a trade-off between using task descriptions to improve fine-tuning efficiency and the goal of shortening prompts. This trade-off becomes more prominent when LoRA is used as a fine-tuning strategy. Therefore, it may be necessary to experiment with different signals to optimize a fine-tuned model (Artur Niederfahrenhorst, 2023).
Safety and Responsibility: Red-teaming exercises are conducted on refined models to verify their safety through both internal and external efforts. Third parties carry out external adversarial testing across fine-tuned models to find performance gaps, and adversarial prompts are provided to aid in the process of fine-tuning the model. There is ongoing work to invest in safety through benchmarking and iterative safety fine-tuning methods. Fine-Tuning LLMs in Practice: Customers strongly desire to be able to fine-tune these LLMs using their own data in order to tailor them when employing generative AI models. It is simple to fine-tune these models in AzureML using either UI-based or code-based techniques. During the fine-tuning phase, pa-

rameters like batch size, learning rate, epochs, etc., can be changed. Furthermore, fine-tuning may be executed with lower compute and memory needs by turning on optimizations like LoRA, DeepSpeed, and ORT (Gharse, 2023).
VRAM and Fine-Tuning: Important considerations are the amount of VRAM required for fine-tuning models of different sizes and the duration of the fine-tuning process. This information is useful for understanding resource requirements for fine-tuning LLM, especially when considering cloud-based fine-tuning. VRAM usage and fine-tuning duration may vary depending on model size and fine-tuning dataset specifications (Gharse, 2023).

Bibliography

Houlsby, N., Giurgiu, A., Jastrzebski, S., Morrone, B., de Laroussilhe, Q., Gesmundo, A., Attariyan, M., & Gelly, S. (2019). Parameter-efficient transfer learning for nlp.

Zhang, B., & Sennrich, R. (2019). Root mean square layer normalization.

Shazeer, N. (2020). Glu variants improve transformer.

Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., Davison, J., Shleifer, S., von Platen, P., Ma, C., Jernite, Y., Plu, J., Xu, C., Scao, T. L., Gugger, S., ... Rush, A. M. (2020). Huggingface's transformers: State-of-the-art natural language processing.

Askell, A., Bai, Y., Chen, A., Drain, D., Ganguli, D., Henighan, T., Jones, A., Joseph, N., Mann, B., DasSarma, N., Elhage, N., Hatfield-Dodds, Z., Hernandez, D., Kernion, J., Ndousse, K., Olsson, C., Amodei, D., Brown, T., Clark, J., ... Kaplan, J. (2021). A general language assistant as a laboratory for alignment.

Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., & Chen, W. (2021). Lora: Low-rank adaptation of large language models.

Liang, T., Glossner, J., Wang, L., Shi, S., & Zhang, X. (2021). Pruning and quantization for deep neural network acceleration: A survey.

Mahabadi, R. K., Henderson, J., & Ruder, S. (2021). Compacter: Efficient low-rank hypercomplex adapter layers.

Qin, G., & Eisner, J. (2021). Learning how to ask: Querying lms with mixtures of soft prompts.

Sung, Y.-L., Nair, V., & Raffel, C. (2021). Training neural networks with fixed sparse masks.

Zhong, R., Lee, K., Zhang, Z., & Klein, D. (2021). Adapting language models for zero-shot learning by meta-tuning on dataset and prompt collections.

An, S., Li, Y., Lin, Z., Liu, Q., Chen, B., Fu, Q., Chen, W., Zheng, N., & Lou, J.-G. (2022). Input-tuning: Adapting unfamiliar inputs to frozen pretrained models.

Bach, S. H., Sanh, V., Yong, Z.-X., Webson, A., Raffel, C., Nayak, N. V., Sharma, A., Kim, T., Bari, M. S., Fevry, T., Alyafeai, Z., Dey, M., Santilli, A., Sun, Z., Ben-David, S., Xu, C., Chhablani, G., Wang, H., Fries, J. A., ... Rush, A. M. (2022). Promptsource: An integrated development environment and repository for natural language prompts.

Bai, Y., Kadavath, S., Kundu, S., Askell, A., Kernion, J., Jones, A., Chen, A., Goldie, A., Mirhoseini, A., McKinnon, C., Chen, C., Olsson, C., Olah, C., Hernandez, D., Drain, D., Ganguli, D., Li, D., Tran-Johnson, E., Perez, E., ... Kaplan, J. (2022). Constitutional ai: Harmlessness from ai feedback.

Chowdhery, A., Narang, S., Devlin, J., Bosma, M., Mishra, G., Roberts, A., Barham, P., Chung, H. W., Sutton, C., Gehrmann, S., Schuh, P., Shi, K., Tsvyashchenko, S., Maynez, J., Rao, A., Barnes, P., Tay, Y., Shazeer, N., Prabhakaran, V., ... Fiedel, N. (2022). Palm: Scaling language modeling with pathways.

Dettmers, T., Lewis, M., Belkada, Y., & Zettlemoyer, L. (2022). Llm.int8(): 8-bit matrix multiplication for transformers at scale.

Glaese, A., McAleese, N., Trebacz, M., Aslanides, J., Firoiu, V., Ewalds, T., Rauh, M., Weidinger, L., Chadwick, M., Thacker, P., Campbell-Gillingham, L., Uesato, J., Huang, P.-S., Comanescu, R., Yang, F., See, A., Dathathri, S., Greig, R., Chen, C., ... Irving, G. (2022). Improving alignment of dialogue agents via targeted human judgements.

Honovich, O., Scialom, T., Levy, O., & Schick, T. (2022). Unnatural instructions: Tuning language models with (almost) no human labor.

Liu, H., Tam, D., Muqeeth, M., Mohta, J., Huang, T., Bansal, M., & Raffel, C. (2022). Few-shot parameter-efficient fine-tuning is better and cheaper than in-context learning.

Min, S., Lewis, M., Zettlemoyer, L., & Hajishirzi, H. (2022). Metaicl: Learning to learn in context.

Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C. L., Mishkin, P., Zhang, C., Agarwal, S., Slama, K., Ray, A., Schulman, J., Hilton, J., Kelton, F., Miller, L., Simens, M., Askell, A., Welinder, P., Christiano, P., Leike, J., & Lowe, R. (2022). Training language models to follow instructions with human feedback.

Pope, R., Douglas, S., Chowdhery, A., Devlin, J., Bradbury, J., Levskaya, A., Heek, J., Xiao, K., Agrawal, S., & Dean, J. (2022). Efficiently scaling transformer inference.

Yao, Z., Aminabadi, R. Y., Zhang, M., Wu, X., Li, C., & He, Y. (2022). Zeroquant: Efficient and affordable post-training quantization for large-scale transformers.

Zaken, E. B., Ravfogel, S., & Goldberg, Y. (2022). Bitfit: Simple parameter-efficient fine-tuning for transformer-based masked language-models.

Ainslie, J., Lee-Thorp, J., de Jong, M., Zemlyanskiy, Y., Lebrón, F., & Sanghai, S. (2023). Gqa: Training generalized multi-query transformer models from multi-head checkpoints.

Artur Niederfahrenhorst, R. A., Kourosh Hakhamaneshi. (2023). Fine-tuning llms: Lora or full-parameter? an in-depth analysis with llama 2.

Bratanic, T. (2023). Knowledge graphs llms: Fine-tuning vs. retrieval-augmented generation.

Christiano, P., Leike, J., Brown, T. B., Martic, M., Legg, S., & Amodei, D. (2023). Deep reinforcement learning from human preferences.

Dettmers, T., Pagnoni, A., Holtzman, A., & Zettlemoyer, L. (2023). Qlora: Efficient finetuning of quantized llms.

Dettmers, T., & Zettlemoyer, L. (2023). The case for 4-bit precision: K-bit inference scaling laws.

Frantar, E., Ashkboos, S., Hoefler, T., & Alistarh, D. (2023). Gptq: Accurate post-training quantization for generative pre-trained transformers.

Gema, A. P., Daines, L., Minervini, P., & Alex, B. (2023). Parameter-efficient fine-tuning of llama for the clinical domain.

Gharse, S. (2023). Introducing llama 2 on azure.

Institute, T. I. (2023). Introducing falcon llm.

Köpf, A., Kilcher, Y., von Rütte, D., Anagnostidis, S., Tam, Z.-R., Stevens, K., Barhoum, A., Duc, N. M., Stanley, O., Nagyfi, R., ES, S., Suri, S., Glushkov, D., Dantuluri, A., Maguire, A., Schuhmann, C., Nguyen, H., & Mattick, A. (2023). Openassistant conversations – democratizing large language model alignment.

Peng, B., Li, C., He, P., Galley, M., & Gao, J. (2023a). Instruction tuning with gpt-4.

Peng, B., Li, C., He, P., Galley, M., & Gao, J. (2023b). Instruction tuning with gpt-4.

Su, J., Lu, Y., Pan, S., Murtadha, A., Wen, B., & Liu, Y. (2023). Roformer: Enhanced transformer with rotary position embedding.

Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.-A., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., Rodriguez, A., Joulin, A., Grave, E., & Lample, G. (2023). Llama: Open and efficient foundation language models.

Touvron, H., Martin, L., Stone, K., Albert, P., Almahairi, A., Babaei, Y., Bashlykov, N., Batra, S., Bhargava, P., Bhosale, S., Bikel, D., Blecher, L., Ferrer, C. C., Chen, M., Cucurull, G., Esiobu, D., Fernandes, J., Fu, J., Fu, W., ... Scialom, T. (2023). Llama 2: Open foundation and fine-tuned chat models.

Xiao, G., Lin, J., Seznec, M., Wu, H., Demouth, J., & Han, S. (2023). Smoothquant: Accurate and efficient post-training quantization for large language models.

Zeng, A., Liu, X., Du, Z., Wang, Z., Lai, H., Ding, M., Yang, Z., Xu, Y., Zheng, W., Xia, X., Tam, W. L., Ma, Z., Xue, Y., Zhai, J., Chen, W., Zhang, P., Dong, Y., & Tang, J. (2023). Glm-130b: An open bilingual pre-trained model.

Zhou, C., Liu, P., Xu, P., Iyer, S., Sun, J., Mao, Y., Ma, X., Efrat, A., Yu, P., Yu, L., Zhang, S., Ghosh, G., Lewis, M., Zettlemoyer, L., & Levy, O. (2023). Lima: Less is more for alignment.

Naveed, H., Khan, A. U., Qiu, S., Saqib, M., Anwar, S., Usman, M., Akhtar, N., Barnes, N., & Mian, A. (2024). A comprehensive overview of large language models.