EE 599 ML Systems Final Project
Efficient Fine-tuning of LLM on a Single GPU

Instructor: Arash Saifhashemi
Submission By: Khushboo Khatri

1 Introduction

Transformer-based large language models (LLMs) like OpenAI's GPT-4 and ChatGPT have significantly expanded language generation and understanding capabilities. Trained on extensive web-scale datasets, these models excel not only in natural language processing but also in solving complex tasks through adept instruction handling and multi-step reasoning. This makes LLMs pivotal in advancing towards artificial general intelligence (AGI).

In our project, we focus on instruction tuning as a key method for adapting pretrained LLMs, using Meta AI's open-sourced LLaMA2 7B model. We'll investigate efficient fine-tuning techniques like gradient accumulation, checkpointing, mixed precision training, and low-rank adaptation, which are crucial for optimizing LLMs in resource-constrained environments, particularly for mitigating concerns regarding memory usage.

2 Background

In this section, your task is to review the LLM survey paper [1] and the LLaMA papers [2, 3] selectively, and provide comprehensive answers to the questions below. These questions will serve as a guide for reading and understanding the content of the papers. You may use Google or ChatGPT to assist in your research, but ensure that your answers are accurate and demonstrate a clear understanding of the concepts discussed in the papers.

1. What are the four major aspects of LLMs covered in the LLM survey paper?
   Answer:

   (a) Statistical language models (SLM)

   (b) Neural language models (NLM)

   (c) Pre-trained language models

   (d) Large Language Models (LLM)

2. What are the three major differences between LLMs and PLMs? What are the three typical emergent abilities for LLMs?
   Answer:
   The survey highlights three major differences between LLMs and PLMs:

   (a) LLMs display some surprising emergent abilities that may not be observed in previous smaller PLMs. These abilities are key to the performance of language models on complex tasks, making AI algorithms unprecedently powerful and effective.

   (b) LLMs would revolutionize the way that humans develop and use AI algorithms. Unlike small PLMs, the major approach to accessing LLMs is through the prompting interface (e.g., GPT-4 API). Humans have to understand how LLMs work and format their tasks in a way that LLMs can follow

(c) The development of LLMs no longer draws a clear distinction between research and engineering. The training of LLMs requires extensive practical experiences in large-scale data processing and distributed parallel training. To develop capable LLMs, researchers have to solve complicated engineering issues, working with engineers or being engineers.

3. Where are pre-training data from?
Answer:
Pre training data are largely from:

   (a) Books: BooksCorpus with 11000 novels and biographies and Project Gutenberg with 7,000 books including novels, essays, poetry, drama, history, science, philosophy and other books available in the public domain are some of the largest open source book collections used in the LLM pretraining. Also there are other book collections that are much larger but haven't been publicly released.

   (b) CommonCrawl: It is one of the largest open-source web crawling databases, containing a petabyte scale data volume, which has been widely used as training data for existing LLMs

   (c) Reddit Links: Highly up voted posts on reddit are often considered useful, and can be utilized to create high-quality datasets.OpenWebText which is an open source alternative of WebText is a well known corpus of highly upvoted links from Reddit. Alternatively Pushshift.io which is a real-time updated dataset that consists of historical data from Reddit.

   (d) Wikipedia: It is an online encyclopedia with a large volume of high-quality articles on diverse datasets.

   (e) Code: Github and StackOverflow are two major open-source public code repositories. Also Google has publicly released the BigQuery dataset [160], which includes a substantial number of open-source licensed code snippets in various programming languages, serving as a representative code dataset

   (f) Other Sources: Pile is a large-scale, diverse, and open source text dataset consisting of over 800GB of data from multiple sources, including books, websites, codes, scientific papers, and social media platforms. It is constructed from 22 diverse high-quality subsets. ROOTS is composed of various smaller datasets (totally 1.61 TB of text) and covers 59 different languages (containing natural languages and programming languages)

4. What are the three main types of instruction tuning datasets? What is the Alpaca dataset [4]? Pick a training sample and describe it.
Answer:
Three main types of Instruction Tuning (Supervised fine tuning) datasets are:

   (a) NLP task datasets

   (b) Chat datasets

   (c) Synthetic datasets

   Alpaca is a dataset of 52,000 instructions and demonstrations generated by OpenAI's text-davinci-003 engine. This instruction data can be used to conduct instruction-tuning for language models and make the language model follow instruction better.
   Example

   "instruction": "Create a classification task by clustering the given list of items.",

   "input": "Apples, oranges, bananas, strawberries, pineapples",

   "output": "Class 1: Apples, Oranges Class 2: Bananas, Strawberries Class 3: Pineapples",

   "text": "Below is an instruction that describes a task, paired with an input that provides further context. Write a response that appropriately completes the request. Instruction:
   nCreate a classification task by clustering the given list of items. Input:Apples, oranges, bananas,

strawberries, pineapples Response:Class 1: Apples, OrangesClass 2: Bananas, StrawberriesClass 3: Pineapples",

5. What are the pertaining data used by LLaMA? How many tokens are in the entire training set for training LLaMA?
   Answer:
   LLaMa was trained using Webpages like Wikipedia, OpenWebText and C4 upto the extent of 80 percent. 5 percent of data came from Code databases like BigQuery and the Github and 5 percent from books and news sources like the Book Corpus and Gutenberg project. Furthermore 3 percent data came from Scientific databases such as the Pile. 2 percent Conversational data like Stack Exchange and the Pile.
   65 Billion tokens were used for pretraining LLaMa.

6. Before pre-training, what is the procedure for data preprocessing?
   Answer:

   (a) Quality Filtering:

      i. Classifier-based: The former approach trains a selection classifier based on high quality texts and leverages it to identify and filter out low quality data

      ii. Heuristic-based: This approach is based on removing low-quality texts through a set of well defined rules, which can be summarized as follows.

         A. Language Filtering: If a LLM would be mainly used in the tasks of certain languages, the text in other languages can be filtered.

         B. Metric Filtering: Evaluation metrics about the generated texts, e.g., perplexity, can be employed to detect and remove unnatural sentences.

         C. Statitstic based Filtering: Statistical features of a corpus, e.g., the punctuation distribution, symbol-to-word ratio, and sentence length, can be utilized to measure the text quality and filter the low-quality data.

         D. Keyword Filtering: Based on specific keyword set, the noisy or unuseful elements in the text, such as HTML tags, hyperlinks, boilerplates, and offensive words, can be identified and removed.

   (b) De-duplication: Existing work has found that duplicate data in a corpus would reduce the diversity of Language models, which may cause the training process to become unstable and thus affect the model performance. DE-duplication can be doe on different granularities.

      i. Sentence Level: Low-quality sentences that contain repeated words and phrases should be removed, as they may introduce repetitive patterns in language modeling.

      ii. Document Level: Existing studies mostly rely on the overlap ratio of surface features (e.g., words and n-grams overlap) between documents to detect and remove duplicate documents containing similar contents

      iii. Set-level: To avoid the dataset contamination problem, it is also crucial to prevent the overlap between the training and evaluation sets, by removing the possible duplicate texts from the training set

   (c) Privacy Reduction: The majority of pre-training text data is obtained from web sources, including user-generated content involving sensitive or personal information, which may increase the risk of privacy breaches.

      i. Rule Based Approach: Use Keyword spotting, to detect and remove PII such as names, addresses, and phone numbers

      ii. De-duplication: Researchers also find that the vulnerability of LLMs under privacy attacks can be attributed to the presence of duplicate PII data in the pre-training corpus. Therefore, deduplication can redduce privacy risks to some extent.

(d) Tokenization: Tokenization is also a crucial step for data preprocessing. It aims to segment raw text into sequences of individual tokens, which are subsequently used as the inputs of LLMs. There are three representative tokenization methods.

7. What is tokenization? Name a few tokenization algorithms. What is the tokenization method used by LLaMA?
Answer:
Tokenization aims to segment raw text into sequences of individual tokens, which are subsequently used as the inputs of LLMs. In traditional NLP research (e.g., sequence labeling with conditional random fields), word-based tokenization is the predominant approach, which is more aligned with human's language cognition. However, wordbased tokenization can yield different segmentation results for the same input in some languages (e.g., Chinese word segmentation), generate a huge word vocabulary containing many low-frequency words, and also suffer from the "out-of-vocabulary" issue. Thus, several neural network models employ character as the minimum unit to derive the word representation (e.g., a CNN word encoder in ELMo).
Recently, subword tokenizers have been widely used in Transformer based language models, typically including:

(a) Reuse Existing Tokenizer

(b) Sentence Piece

(c) Byte-level BPE

Also there are three Tokenization algorithms:

(a) BPE was originally proposed as a general data compression algorithm in 1994 [221], and then adapted to NLP for tokenization [222]. It starts with a set of basic symbols (e.g., the alphabets and boundary characters), and iteratively combine frequent pairs of two consecutive tokens in the corpus as new tokens (called merge). For each merge, the selection criterion is based on the co-occurrence frequency of two contiguous tokens: the top frequent pair would be selected. The merge process continues until it reaches the predefined size. Further, Byte-level BPE has been used to improve the tokenization quality for multilingual corpus (e.g., the text containing non-ASCII characters) by considering bytes as the basic symbols for merge. Representative language models with this tokenization approach include GPT-2, BART, and LLaMA.

(b) WordPiece tokenization: WordPiece was a Google internal subword tokenization algorithm. It was originally proposed by Google in developing voice search systems [223]. Then, it was used in the neural machine translation system in 2016, and was adopted as the word tokenizer for BERT in 2018. WordPiece has a very similar idea with BPE by iteratively merging consecutive tokens, whereas taking a slightly different selection criterion for the merge. To conduct the merge, it first trains a language model and employs it to score all possible pairs. Then, at each merge, it selects the pair that leads to the most increase in the likelihood of training data. Since Google has't released the official implementation of the WordPiece algorithm, HuggingFace gives a more intuitive selection measure in its online NLP course: a pair is scored by dividing the co-occurrence count by the product of the occurrence counts of two tokens in the pair based on training corpus.

(c) Unigram tokenization: Unlike BPE and WordPiece, Unigram tokenization [225] starts with a sufficiently large set of possible substrings or subtokens for a corpus, and iteratively removes the tokens in the current vocabulary until the expected vocabulary size is reached. As the selection criterion, it calculates the yielded increase in the likelihood of training corpus by assuming that some token was removed from current vocabulary. This step is conducted based on a trained unigram language model. To estimate the unigram language model, it adopts an expectation–maximization (EM) algorithm: at each iteration, we first find the currently optimal tokenization of words based on the old language model, and then re-estimate the probabilities of unigrams to update the language model. During this procedure, dynamic programming algorithms (i.e., the Viterbi algorithm) are used to efficiently find the optimal decomposition way of a word given the language model. Representative models that adopt this tokenization approach include T5 and mBART.

Byte Pair Encoding(BPE) was the tokenization algorithm used in LLaMa.

8. What are the main three types of transformer architecture? Name a few models for each type
   Answer:

   (a) Encoder-decoder Architecture. The vanilla Transformer model is built on the encoder-decoder architecture [22], which consists of two stacks of Transformer blocks as the encoder and decoder, respectively. The encoder adopts stacked multi-head self-attention layers to encode the input sequence for generating its latent representations, while the decoder performs cross-attention on these representations and autoregressively generates the target sequence. Encoder-decoder PLMs (e.g., T5 [82] and BART [24]) have shown effectiveness on a variety of NLP tasks. Model using encoder-decoder architechture: Flan-T5

   (b) Causal Decoder Architecture. The causal decoder architecture incorporates the unidirectional attention mask, to guarantee that each input token can only attend to the past tokens and itself. The input and output tokens are processed in the same fashion through the decoder. Models using this architechture: GPT series, OPT, BLOOM and Gopher

   (c) Prefix Decoder Architecture. The prefix decoder architecture (a.k.a., non-causal decoder [244]) revises the masking mechanism of causal decoders, to enable performing bidirectional attention over the prefix tokens [245] and unidirectional attention only on generated tokens. In this way, like the encoder-decoder architecture, the prefix decoders can bidirectionally encode the prefix sequence and autoregressively predict the output tokens one by one, where the same parameters are shared during encoding and decoding. Instead of pre-training from scratch, a practical suggestion is to continually train causal decoders and then convert them into prefix decoders for accelerating convergence. Models using this architechture: GLM 130B, U-PaLM

9. Why are LLMs mainly decoder-only architecture?
   Answer:
   By pre-training with the LM objective, it seems that causal decoder architecture can achieve a superior zeroshot and few-shot generalization capacity. Existing research has shown that without multi-task fine-tuning, the causal decoder has better zero-shot performance than other architectures [29]. The success of GPT-3 [55] has demonstrates that the large causal decoder model can be a good fewshot learner. In addition, instruction tuning and alignment tuning discussed in Section 5 have been proven to further enhance the capability of large causal decoder models [66, 67, 69]. • Scaling law has been widely observed in causal decoders. By scaling the model size, the dataset size, and the total computation, the performance of causal decoders can be substantially improved. Thus, it has become an important strategy to increase the model capacity of the causal decoder via scaling. However, more detailed investigation on encoder-decoder models is still lacking, and more efforts are needed to investigate the performance of encoder-decoder models at a large scale.

10. What is position embedding? Name a few position embedding algorithms.
    Answer:
    Position embedding is mathematical representation, especially in transformers, that capture the position or order of tokens in a sequence. Few position embedding algorithms are:

    (a) Rotary Position Embeddings (RoPE): In Llama paper, they have used RoPe method. By encoding token positions through rotation in the embedding space, Rotary Positional Embeddings provide an alternative to conventional positional embedding techniques. By taking this approach, the model performs better on tasks that call for a thorough comprehension of sequence structure and relationships because it can better capture and utilize the relative positioning of tokens.

    (b) Absolute Position Embeddings: This traditional approach involves adding a unique vector to each input token embedding, where the vector encodes the position of the token in the sequence. The position embeddings can be learned during training or fixed (e.g., based on sine and cosine functions of different frequencies, as proposed in the original transformer paper by Vaswani et al.)

    (c) Relative Position Embeddings: Relative position embeddings concentrate on the separation between token pairs within a sequence rather than storing the absolute position of each token. This method is advantageous for jobs where the relative positioning of tokens is more essential than their absolute positions because it can capture the relative order of words more flexibly.

11. What is language modeling? What is the difference between causal language modeling and masked language modeling?
Answer:
One of the fundamental tasks in natural language processing (NLP) is language modeling, which is estimating the probability distribution of different linguistic sequences. It basically involves guessing the following word in a sentence based on the ones that came before it. Many NLP applications, such as text production, speech recognition, and machine translation, depend on this capacity. The aim of a language model is to learn the fundamental structure and nuances of a language in order to comprehend and produce writing that is similar to that of a person.
Key differences between causal language modeling and masked language modeling are:

    (a) Causal Language Modeling:
    CLM involves predicting the next word in a sequence given all the previous words. It represents a word sequence's probability distribution moving forward, that is, from left to right (or right to left, but consistently in one direction). GPT (Generative Pre-trained Transformer) series models are famous examples of CLM. They generate text by predicting the next word iteratively.

    (b) Masked Language Modeling:
    A certain portion of input tokens are randomly masked. The goal of the model is to infer the original identity of these masked tokens from the context that the sequence's unmasked tokens give. The model takes a bidirectional approach to prediction, taking into account both left and right context.BERT (Bidirectional Encoder Representations from Transformers) is a pioneering model using the MLM approach.

12. How to generate text from LLMs [5]? Explain different decoding strategies.
Answer:
Various decoding strategies, each with their own advantages and appropriateness for different applications, can be used to direct the text production process. Here is a summary of various well-known decoding techniques based on the Hugging Face text generating guide.

    (a) Greedy Decoding: In greedy decoding, at each step, the model selects the word with the highest probability as the next word. It's a straightforward approach that simply picks the most likely next word at each time step. (citing from huggingface guide). Though, the major drawback of greedy search though is that it misses high probability words hidden behind a low probability word as can be seen in attached image : We will get output as ”the nice woman” but ”the dog has” should be selected which will be taken care in Beam Search.

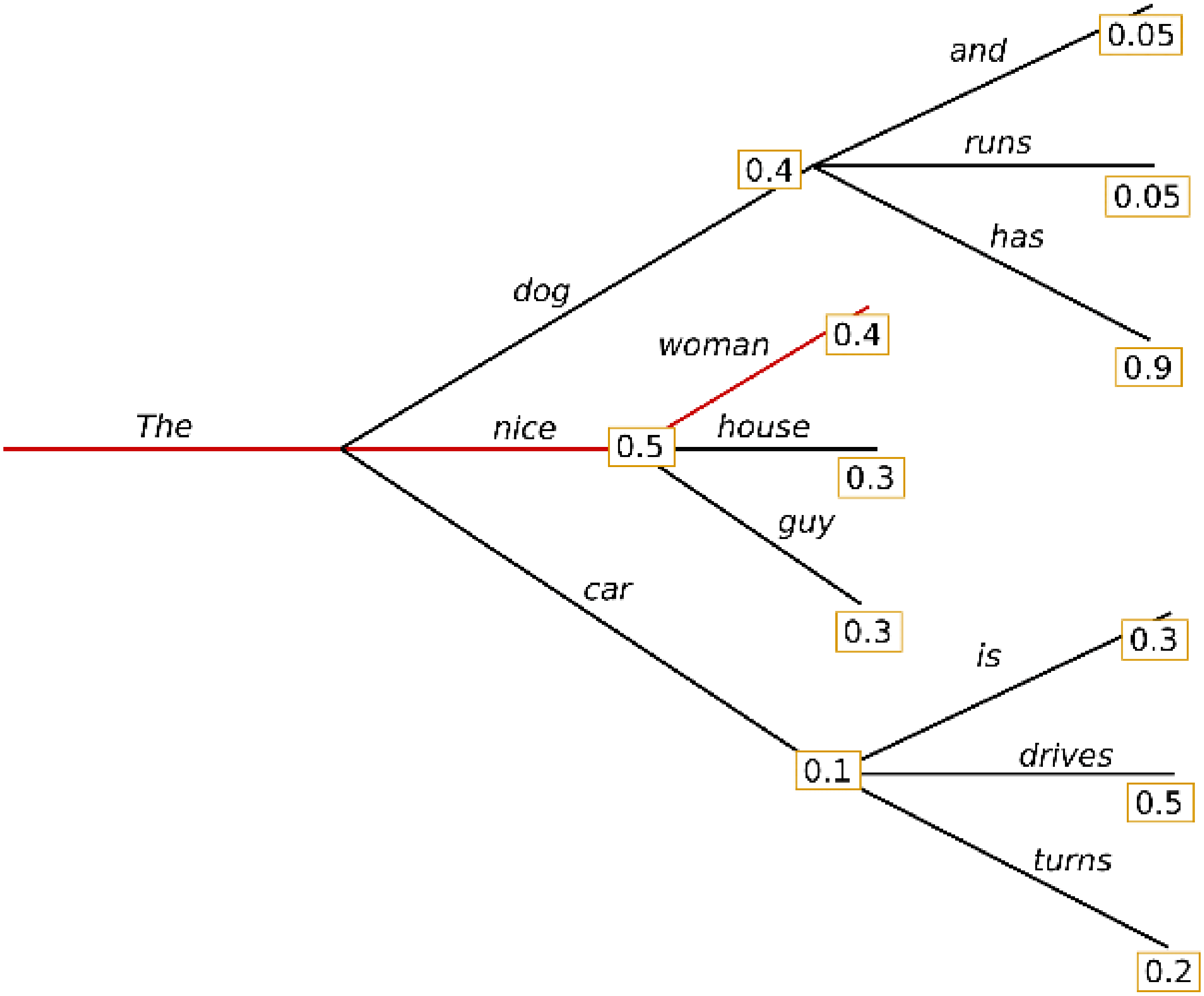
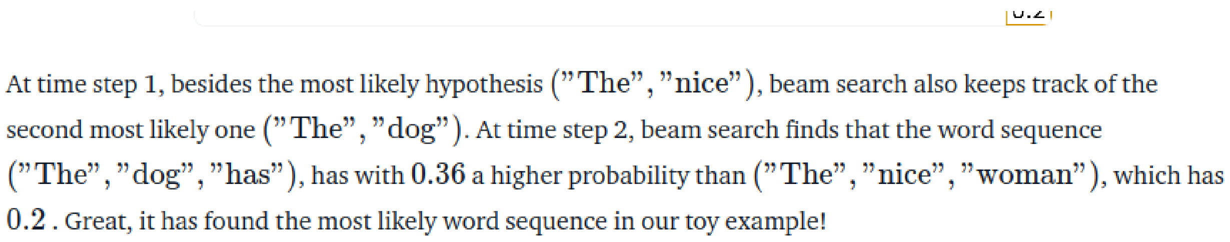


Figure 1: Greedy Search Example

(b) Beam Search: Beam search expands on greedy decoding by considering multiple possible next words (the ”beam width”) at each step and keeping a fixed number of most likely sequence candidates at each step (num_beams). Consider figure 1, and check cited figure 2 from hugging face guide.



At time step 1, besides the most likely hypothesis (”The”, ”nice”), beam search also keeps track of the second most likely one (”The”, ”dog”). At time step 2, beam search finds that the word sequence (”The”, ”dog”, ”has”), has with 0.36 a higher probability than (”The”, ”nice”, ”woman”), which has 0.2 . Great, it has found the most likely word sequence in our toy example!

Figure 2: Beam Search Huggingface output example

(c) Top-K Sampling: Top-K sampling selects K most likely candidates from among the pool of potential next words at each step, and then draws a random sample from this subset according to the candidates' probability distribution. This adds diversity and randomness to the text that is generated.

(d) Top-p (nucleus) sampling: Nucleus sampling, or top-p sampling, dynamically chooses the smallest group of words whose cumulative probability is greater than a predetermined p. Then, in order to maintain a balance between randomness and relevance, the model samples from this set.

13. What is instruction tuning, and why is it important?
Answer:
A language model (LM), which has been pre-trained on a large corpus of text data, is subjected to additional training (fine-tuning) on a dataset created especially to educate the model how to follow instructions or carry out tasks that are stated in natural language. This procedure is known as instruction tuning. This procedure is critical to improve the model's comprehension and execution of tasks, thus strengthening its "task agnostic" capabilities.
It is important because:

(a) Improves Task Generalization: Through training on a variety of tasks and instructions, the model improves its ability to generalize across activities that it was not specifically trained on. This implies that it does not require task-specific training data in order to execute new tasks on the basis of instructions.

(b) Facilitates Multi-Task Learning: Multiple jobs can be handled by models that have been fine-tuned with instructions, eliminating the need for distinct models for each activity. The computational and resource overhead related to deploying several specialized models is decreased by this multitasking capability.

14. What is alignment tuning, and why is it important?
Answer:
In order to make sure that a model's outputs closely match human values, intents, or planned outcomes, alignment tuning focuses on modifying a model's objectives, training data, and learning process. It's especially crucial for the creation of large language models (LLMs), which are used to produce text that resembles that of a human, respond to inquiries, and carry out other operations involving the comprehension and production of language. Also citing from paper (The Unlocking Spell on Base LLMs: Rethinking Alignment via In-Context Learning) " The alignment tuning process of large language models (LLMs) typically involves instruction learning through supervised fine-tuning (SFT) and preference tuning via reinforcement learning from human feedback (RLHF)." It is important because:

(a) Enhances Safety and Trustworthiness: AI systems must generate results that are consistent with social norms and human ethical standards in order to be safely incorporated into society and a variety of industries (such as healthcare, banking, and education). By ensuring that AI systems' actions and outputs do not violate these norms, alignment tuning plays a critical role in fostering trust in these systems.

(b) Facilitates Generalization: Models are more adaptable and efficient in a variety of applications when they are properly aligned, as this improves their ability to generalize from training data to real-world circumstances.

15. What is parameter-efficient fine-tuning, and why is it important? What is the difference between prefix tuning and prompt tuning?
Answer:
Methods for adjusting large pre-trained models to particular tasks without retraining the model as a whole are known as parameter-efficient fine-tuning techniques. These techniques become vital in situations when computational resources are sparse or where time, memory, or energy limitations make it unfeasible to adjust every parameter of an extremely large model. These strategies make it possible to effectively adapt pre-trained models to new tasks or domains by changing a small portion

of the model's parameters.
Both prefix tuning and prompt tuning are parameter-efficient techniques, but they apply different methods to incorporate task-specific adjustments into pre-trained models.

(a) Prefix tuning: Prefix tuning is the process of feeding each transformer layer in the model's input a series of continuous vectors, or prefixes. In order to direct the model's attention and intermediate representations in a way that is advantageous for the goal task, these prefixes are learned during the fine-tuning phase.
The pre-trained weights are kept constant as the model analyzes these task-specific prefixes in addition to the actual input sequences. This technique modifies the internal attention mechanisms of the model to efficiently guide its behavior.

(b) Prompt tuning: Conversely, prompt tuning entails adding a set of learnable tokens, or the prompt, to the input data. Following that, the model is adjusted to modify these prompt tokens in a way that enhances performance on the intended job. Prompt tuning, in contrast to prefix tuning, modifies the model's basic data representations at the input level.
By signaling the model for the particular job at hand, these cues are improved throughout training, allowing the model to efficiently use its prior knowledge in a novel setting. Preserving the general capabilities acquired during pre-training, the other parameters of the model stay fixed.

16. What is prompt engineering, and why is it important? What is zero-shot and few-shot demonstrations?
Answer:
The process of creating and improving the input prompts that are sent to a language model in order to produce particular or desired outcomes is known as prompt engineering. This procedure is important because the structure of the input prompt can have a big impact on how well a language model performs, particularly when it comes to tasks like text production, question answering, or any kind of language understanding. A more accurate, pertinent, and contextually suitable answer from the model can result from effective rapid engineering. It entails structuring the prompt to direct the model's comprehension of the activity at hand and produce results that are consistent with the user's wishes.
Zero-shot and few-shot demonstrations:

(a) Zero-shot demonstration: The model is given a task in the zero-shot setting, but no explicit examples of how to complete it are provided. To provide a suitable response, the model has to rely on its prior knowledge and the details in the prompt. Zero-shot learning evaluates the model's capacity to apply lessons learned during training to brand-new, untested tasks by using just the prompt's description.

(b) Few-shot Demonstrations: Few-shot learning, as opposed to zero-shot learning, consists of giving the model a limited number of instances, or shots, to illustrate the task at hand. The prompt contains these examples so that the model has a point of reference for what is anticipated. Few-shot learning uses these instances as a guide to improve the model's performance on the assignment. It is especially helpful when fine-tuning the model's outputs to be more task-specific or when zero-shot performance might not be sufficient for the task at hand.

17. What is the difference between in-context learning and chain-of-thought prompting?
Answer:

(a) In-context learning: The term "in-context learning" describes a model's capacity to modify its behavior or predictions without explicit retraining or fine-tuning, based on examples given inside the prompt itself. Through the use of pretrained knowledge ingrained in the model's parameters, this method enables the model to generalize across tasks and adapt to prompt-specific instructions. Large language models such as GPT-3 are good examples of in-context learning because they allow the model to be trained to accomplish a job given a few examples in the input prompt.

(b) Chain-of-Thought Prompting: A method called chain-of-thought prompting is intended to enhance language models' ability to perform on challenging reasoning assignments. Instead of asking for the answer straight away, the prompt must be framed to contain a methodical thinking process

that leads to the solution. By using this technique, the model is able to ”think aloud” or follow a line of reasoning that is similar to how people solve problems. Chain-of-thought prompting, as opposed to direct questioning or conventional prompting techniques, might assist models in handling tasks that call for multi-step reasoning, deeper comprehension, or complicated inference more successfully by explicitly outlining the reasoning steps.
ALso citing the idea from (, When faced with a difficult issue, people tend to divide it up into smaller, more manageable chunks. For instance, there are usually several steps involved in solving a complicated arithmetic equation, all of which are necessary to get the final, accurate solution. By asking an LLM to simulate this method of breaking down a problem and solving it step by step, chain-of-thought prompting essentially asks the model to ”think out loud,” as opposed to just offering a direct solution.

18. What are the three basic types of ability evaluation for LLMs? What is perplexity [6]? What is hallucination?
Answer:
For LLMs, the three primary ability evaluation categories are usually understanding, reasoning, and generation capacities. While creation is creating content that is cohesive and pertinent to the context, understanding entails understanding and interpreting information. Reasoning calls for logical conclusions or inferring information that is not expressly expressed.

A statistic called perplexity is used to assess language models, most especially the classical autoregressive or causal language models. It is the model's capacity to predict tokens and is defined as the exponentiated average negative log-likelihood of a sequence. Reduced confusion suggests that the model is performing better in terms of prediction.

In language models, the term ”hallucination” describes the production of false or absurd information that is not based on the input data. Text creation challenges present a challenge when the model generates content that appears convincing but is factually wrong.

19. What is human alignment, and why is it important?
Answer:
The creation of AI systems and algorithms that comprehend and behave in accordance with human values, objectives, and ethical considerations is commonly referred to as ”human alignment.” This idea seeks to guarantee that AI systems make choices and behave in ways that advance human well being and moral principles.
We have talked about alignment tuning and its importance in ques 14, pointing here again. In order to make sure that a model's outputs closely match human values, intents, or planned outcomes,alignment tuning focuses on modifying a model's objectives, training data, and learning process. It's especially crucial for the creation of large language models (LLMs), which are used to produce text that resembles that of a human, respond to inquiries, and carry out other operations involving the comprehension and production of language.

It is important because:

(a) Prevent Misalignment: As AI systems become more powerful and autonomous, it is essential to make sure that their objectives are in line with human values in order to avoid unforeseen or undesirable consequences.
(b) Ethical Considerations: Concerns about privacy, justice, and autonomy are among the moral and ethical issues that can be resolved by integrating AI systems with human ethics and values.
(c) Adoption and Trust: Because users believe that human-aligned AI systems are acting in their best interests, they are more likely to be trusted and embraced by them.
(d) Safety and Security: Preventing AI from acting in an undesired or hazardous manner requires that AI systems be in line with human aims.

20. Name a few comprehensive evaluation benchmarks for the evaluation of LLMs.

Answer:

Some of the comprehensive evaluation benchmarks for the evaluation of LLMs are:

1. MMLU: Multi-tasking benchmark designed to assess general language understanding.
2. BBH: Benchmark focusing on behavioral aspects of language models.
3. TyDiQA: Evaluation framework for question answering tasks across typologically diverse languages.
4. MGSM: Benchmark for measuring models' abilities in multi-step reasoning and problem-solving.

5. AlpacaFarm evaluation: AlpacaFarm makes the RLHF process accessible to everyone by providing a simulator that replicates the RLHF process quickly (24h) and cheaply (200)

3 Optimization Methods

3.1 Gradient Accumulation

In standard neural network training, we typically divide our data into smaller chunks called mini-batches and process them sequentially. The network generates predictions for each batch, and we compute the loss by comparing these predictions to the actual targets. Afterward, we perform a backward pass to calculate gradients, which are used to adjust the model's weights in the direction of these gradients.

Gradient accumulation alters this last step of the training process. Instead of updating the model's weights after processing each individual batch, we save the gradient values and continue to the next batch, accumulating the new gradients. Weight updates are deferred until the model has processed several batches.

The primary purpose of gradient accumulation is to simulate a larger effective batch size. For instance, imagine you intend to use a batch size of 32 images, but your hardware can handle only up to 8 images without running out of memory. In this scenario, you can work with smaller batches of 8 images and update the weights after every 4 batches by accumulating gradients from each batch in between. To ensure the accumulated gradient remains equivalent to the non-accumulated gradient on a batch of 32 images, you must divide your loss value on each mini-batch of 8 images by 4. Consider a linear model with MSE loss, and mathematically explain why this division step can yield identical results.

Your answer:

Consider a linear model where the output prediction $\hat{y}$ for each input vector x is given by the linear equation $\hat{y} = Wx + b$. The Mean Squared Error (MSE) loss for a set of predictions is:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2$$

where $\hat{y}_i$ is the predicted value and y_i is the actual value for the i-th observation. The gradient of the MSE loss with respect to the weights W is:

$$\nabla_W \text{MSE} = \frac{2}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)x_i$$

In the context of gradient accumulation, if we divide the dataset into smaller batches due to hardware limitations, we can accumulate the gradients from each batch and then update the weights. This approach requires adjusting the gradient computation as follows to ensure that the accumulated gradient reflects the average gradient across the entire dataset:

$$\begin{aligned}\nabla_W \text{MSE}_{\text{accumulated}} &= \frac{1}{k}\sum_{j=1}^{k}\left(\frac{2}{m}\sum_{i=1}^{m}(\hat{y}_{ji} - y_{ji})x_{ji}\right)\\ &= \frac{2}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)x_i\end{aligned}$$

This adjustment ensures that the scale of the gradient update is consistent with the full batch size, making the effect of gradient accumulation equivalent to processing a larger batch in one go.

It's crucial to remember that gradient accumulation could not always provide the same outcomes, particularly for deep neural network architectures that use batch normalization or other batch-wise regularization techniques. The statistics used for batch normalization are determined within each batch, and these statistics may become inconsistent as gradients are accumulated. Explain the behavior of the batch normalization layer mathematically, as well as the reasons that gradient accumulation in this instance produces different outcomes.

Your answer:

One method for enhancing artificial neural networks' speed, functionality, and stability is batch normalization. By deducting the batch mean and dividing by the batch standard deviation, BN normalizes the output of an earlier activation layer. Mathematically, for a given feature x, the batch-normalized $\hat{x}$ is computed as:

$$\hat{x} = \frac{x - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$$

where:

- μ_B is the mean of the batch.
- σ_B^2 is the variance of the batch.
- ϵ is a small constant added for numerical stability.

After normalization, BN then scales and shifts the normalized value with learnable parameters γ and β, respectively:

$$y = \gamma\hat{x} + \beta$$

This scaling and shifting allow the network to undo the normalization if it becomes counterproductive.

The core issue with applying gradient accumulation in the context of batch normalization arises from BN's reliance on per-batch statistics. When gradients are accumulated across mini-batches, each mini-batch has its batch statistics (μ_B and σ_B^2), which are used for normalization. This process means that the normalized values in each mini-batch differ based on the specific batch's data, leading to inconsistencies when these gradients are accumulated.

Accumulating gradients without updating these statistics throughout the collected batches indicates that the normalization is not similar to that of a larger batch because BN modifies the activation output based on the mean and variance of the current batch. When contrasted to training with the intended larger batch size, this disparity may result in non-identical outcomes because the normalizing effect of BN is not applied consistently throughout the theoretically bigger batch.

In conclusion, gradient accumulation can replicate bigger batches for gradient updates but it is not a precise simulation of the batch normalization procedure that takes place across these larger batches. This restriction may have an effect on the model's performance and training stability, particularly in deep neural networks where batch normalization is essential to regularization and training dynamics.

3.2 Gradient Checkpointing

Gradient checkpointing is another memory-saving technique that strikes a balance between memory and computation during neural network training. While it's well understood that storing all activations and intermediate results from the forward pass is essential for performing backward propagation, this process can lead to a significant increase in memory usage, often referred to as the "memory blow-up" problem. When monitoring memory consumption during model training, you'll notice that it gradually accumulates and reaches its peak after completing the forward pass. Explain why model inference does not have such "memory blow-up" problem.

Your answer:

The reason model inference does not experience the "memory blow-up" problem is that, unlike training, inference involves only the forward pass. During inference, the model applies the learned weights to new data to make predictions. There's no need to store activations for the backward pass because the model is not updating its weights – it's simply applying them. Therefore, the memory footprint remains much smaller, as it only needs to accommodate the activations produced during the single forward pass.

In mathematical terms, if we consider the memory required for storing activations as (M), and the number of layers in the network as (L), then during training, the memory required would be proportional to $L \times M$ for all activations. However, during inference, the memory required remains (M), as we only need to store activations for the current layer being computed. This is why there's no "memory blow-up" during inference.

Instead of retaining all activations and intermediate results in memory throughout the forward pass, gradient checkpointing offers an alternative approach. It strategically selects checkpoints within the network where memory efficiency takes precedence over computation time. When the algorithm encounters these checkpoints during the subsequent backward pass, it does not rely on stored activations because they were not stored in memory in the first place. Instead, it recalculates them by re-executing the forward pass up to the specific checkpoint.

To optimize memory usage and forward computation steps using gradient checkpointing in a feed-forward neural network composed of n layers, let's consider two strategies: Imagine a simple feed-forward neural network with n layers, each generating activations of the same size. Upon completing the forward pass, the algorithm retains all activations, resulting in an $O(n)$ memory requirement for this network. In other words, it consumes n units of memory. In terms of computation, the forward pass requires $O(n)$ steps, assuming that each layer requires 1 unit of computation.

Now, the question revolves around determining the optimal placement of checkpoints to strike a balance between memory and computation. A poor strategy would be discarding all activations during the forward pass and recomputing them later when needed. Conversely, a more general and optimal strategy suggests discarding activations every $\sqrt{n}$ steps. What are the memory requirement and forward computation steps for the two strategies in big O notation? Check this medium post [7] for more illustrations.

Your answer:

If we discarded all activations during the forward pass and recomputed them later when needed then that would lead to a memory requirement of O(1).
While if we discard the activations every $\sqrt{n}$ steps then we get a memory requirement of O(n)

In practical scenarios, neural networks often feature layers with varying activation sizes, and the computational complexity of each layer's forward pass can differ significantly. Furthermore, the forward graph of these networks may not adhere to a strictly sequential pattern. Consequently, the task of strategically placing checkpoints to optimize memory usage and computation steps remains an active area of research and attention.

3.3 Low-Rank Adaptation (LoRA)

Parameter-efficient Fine-tuning (PEFT) is a technique in NLP that enhances the performance of pre-trained language models on specific tasks. It freezes most of the pre-trained model's parameters and only fine-tunes a subset of layers, reducing computational and memory demands and training time, while retaining a good performance as possible. The idea of reducing the number of trainable parameters has its roots in transfer learning within computer vision tasks, where traditionally, only the final fully connected layer was updated.

Before we dive into the technical aspect of Low-Rank Adaptation (LoRA), one must revisit the transformer architecture and understand the mechanism of self-attention. In addition, LoRA uses the concept of truncated Singular Value Decomposition (SVD). Please study and understand the concepts of SVD and truncated SVD. Answer the following questions:

1. What is matrix rank?
   Your answer:

The rank of a matrix is a fundamental concept in linear algebra which gives an indication of how many dimensions are present in the vector space that its rows and columns span. A matrix's rank can be determined by counting the highest number of linearly independent row vectors or column vectors within the matrix, respectively.

The matrix rank idea aids in determining the bare minimum of parameters required to adequately capture the important details of the data transformations occurring inside the Transformer layers in the context of LoRA. LoRA keeps the fine-tuning procedure computationally fast while collecting the necessary information and variances by restricting the updates to a low-rank approximation of the weight matrices.

2. What are three decomposed matrices by SVD?
   Your answer:
   Singular Value Decomposition (SVD) is a powerful matrix factorization technique in linear algebra that decomposes any given matrix into three specific matrices, showcasing the geometric and algebraic structure of the original matrix. The three decomposed matrices in SVD are: U,Σ Matrix, and V matrix.

   (a) U Matrix: The U matrix represents the rotation or reflection needed in the original space to align it with the principal axes of variation.

   (b) Sigma Matrix: These singular values are non-negative and can be thought of as the scaling factors that stretch or compress the data when it's projected onto the principal axes of variation.

   (c) V Matrix: It represents the rotation or reflection needed to align the principal axes of variation with the axes of the target space.

   The mathematical represention of SVM is: The SVD of mxn matrix A is given by the formula A $= U\Sigma V^T$

   U : an $m \times m$ matrix of the orthonormal eigenvectors of AA^T.

   V^T : the transpose of an $n \times n$ matrix containing the orthonormal eigenvectors of $A^T A$.

   Σ : a diagonal matrix with r elements equal to the square root of the positive eigenvalues of AA^T or $A^T A$ (both

3. U and V are orthogonal matrices. Why does it imply $UU^T = I, VV^T = I$?
   Your answer:
   Orthogonal matrices have a special property that directly implies why the product of an orthogonal matrix and its transpose equals the identity matrix.

   The element in the ith row and jth column of UU^T, denoted as $(UU^T)_{ij}$, is the dot product of the ith row of U and the jth column of U^T. Because U^T's columns are U's rows, this is essentially the dot product of the ith and jth rows of U.

   - If $i = j$, we're taking the dot product of a row with itself, which equals the sum of the squares of its elements. Since each row has unit length, this sum is 1.
   - If $i \neq j$, the dot product is 0 because the rows are orthogonal to each other (their dot product is zero).

   Therefore, $(UU^T)_{ij} = 1$ if $i = j$ and 0 otherwise, which is exactly the definition of the identity matrix I.

4. If a matrix $W \in R^{n \times n}$ is full rank, what is its rank?
   Your answer:
   Its rank is n.

The rank of a matrix is defined as the maximum number of linearly independent row vectors or column vectors in the matrix. For a square matrix W in $R^{n\times n}$ to be full rank means that all its rows and all its columns are linearly independent. Thus, there are no redundant rows or columns that can be expressed as a linear combination of the others.

5. Suppose a full rank matrix $W \in R^{n\times n}$ represents an image. After we apply SVD to this matrix, we modify the singular matrix by only keeping its top-k singular values and discarding the rest (i.e., set the rest of the singular values to zero). Then, we reconstruct the image by multiplying U, modified S, and V. What would the reconstructed image look like? What if you increase the values of k (i.e., keep more singular values)?

   Your answer:

   When you apply the Singular Value Decomposition (SVD) to an image represented by a matrix W and keep only the top k singular values, you're essentially performing a form of compression or dimensionality reduction on the image.

   The reconstructed image will approximate the original image, but it will have lost some information due to the removal of singular values. The quality of the reconstruction depends on how many singular values you retain (the value of k).

   Increasing the value of k will lead to a better quality reconstruction, as more singular values are retained, preserving more information from the original image.

6. If a matrix $W \in R^{n\times n}$ is low rank, what does its singular matrix look like?

   Your answer:

   If a matrix W is low rank, it means that it can be approximately represented by a matrix with fewer singular values than its size would suggest.

   For a matrix with low rank, its singular values will be concentrated towards zero after being sorted in descending order. This is because the rank of a matrix is determined by the number of non-zero singular values. In a low-rank matrix, most of the singular values will be close to zero, indicating that there is a lot of redundancy or correlation between the rows or columns of the matrix.

7. If the top-k singular values of a matrix $W \in R^{n\times n}$ are large, and the rest are near zero, this matrix W exhibits low-rank or near-low-rank behavior. Can you represent W by two low-rank matrices, A and B? If so, what are those two matrices' expressions in terms of U, S, and V? Do you think those two matrices are a good approximation of W (i.e., $W \approx AB$)?

   Your answer:

   Let's denote the original SVD decomposition of W as $W = USV^T$, where U is an $n \times n$ orthogonal matrix, S is an $n \times n$ diagonal matrix containing the singular values in descending order, and V is an $n \times n$ orthogonal matrix.

   If the top k singular values are large and the rest are near zero, we can approximate W by keeping only the first k columns of U and V and the first k singular values in S, and then multiplying them to get the approximation AB:

$$A = U[:, 1:k]S[1:k, 1:k]^{1/2}$$

$$B = S[1:k, 1:k]^{1/2}V^T[:, 1:k]$$

   Here, $U[:, 1:k]$ denotes the first k columns of U, $S[1:k, 1:k]^{1/2}$ denotes the square root of the first k singular values, and $V^T[:, 1:k]$ denotes the first k rows of V^T.

   These two matrices A and B are good approximations of W, especially when k is chosen appropriately based on the amount of energy captured by the top singular values. The product AB gives a low-rank approximation of W, and the approximation quality improves as more singular values are retained (i.e., as k increases).

8. The above operation is called truncated SVD. Under what situation do you think truncated SVD fails to make a good approximation? Think about the singular matrix.
   Your answer:
   Truncated SVD may fail to make a good approximation when the singular values do not exhibit a clear separation between dominant and negligible values

Now, you are equipped with the knowledge to understand LoRA. As one of the most effective PEFT techniques, proposed by Edward Hu et al. in 2021 [8], the success of LoRA is based on an important observation that both pre-trained weights and change of weights during fine-tuning for LLMs are low-rank matrices.

To illustrate this, consider representing the final optimal weights after fine-tuning as $W^* = W_0 + \Delta W$, where $W_0 \in R^{n\times n}$ represents the pre-trained weights, and $\Delta W \in R^{n\times n}$ captures the weight changes during fine-tuning. Notably, ΔW can be approximated as the product of two matrices, AB, with $A \in R^{n\times r}$ and $B \in R^{r\times n}$. This implies that during fine-tuning, we don't have to update the whole matrix W_0. Instead, we modify the forward pass in the layer, changing it from $y = xW_0$ to $y = xW_0 + xAB = x(W_0 + AB)$, as depicted in Figure 3. We freeze W_0 and only update A and B. Notice that since $r << n$, the total number of parameters of A and B is much smaller than that of W_0. To make the modified forward pass identical to the original one at the beginning of fine-tuning, A is randomly initialized in Gaussian distribution, and B is initially set to 0, so $AB = 0$.

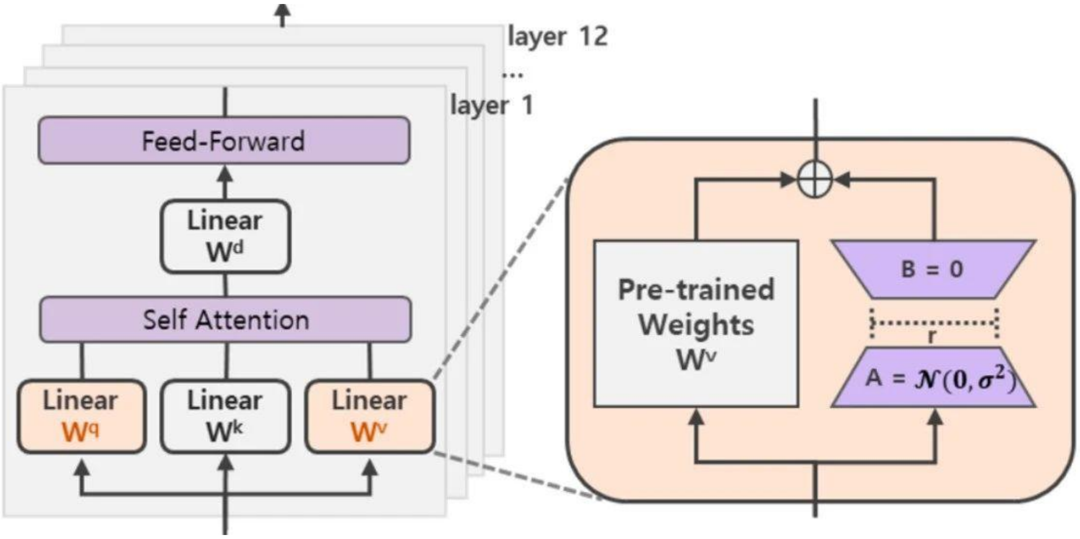


Figure 3: LoRA

Technically, we can insert A and B into any layers, such as the query, key, value, output projection layers, or the FFN layers in each attention block. However, the paper suggests that applying LoRA only to query and value projection layers can yield performance comparable to full parameter fine-tuning. The hyperparameter r typically takes on small values within the range of 4 to 32. By inserting A and B into specific layers and keeping the rest of the pre-trained weights frozen, LoRA achieves a substantial reduction in the total number of trainable parameters. For $r = 8$, the percentage of parameters that are subject to training accounts for less than 1% of the overall parameters in a typical LM. This remarkable reduction in the number of trainable parameters is a key factor in the efficiency of LoRA as a PEFT technique.

It seems like applying LoRA would add more parameters to the model, thus increasing forward pass costs and inference latency. How does LoRA paper overcome this drawback?

Your answer:

In the paper, to over this they have talked about-

1. Low Rank Matrix Decomposition - LoRA adds more trainable, considerably smaller low-rank matrices in place of directly altering the big weight matrices of the transformer layers. With a low-rank decom-

position, these matrices modify the original weights' behavior to suit the characteristics of new jobs without changing the original values. This method drastically lowers the amount of parameters that must be changed during adaption.

2. Parameter Efficiency- The approach is made to be as parameter-efficient as possible, which means that with fewer parameters, it can achieve notable gains in performance on the modified tasks. Because of its minimal computing overhead and ability to handle even very big models without becoming unaffordable, LoRA's efficiency is essential.

3. Adaptability with Minimal Overhead- LoRA's methodology enables LLMs to be used in real-world applications where response times are crucial by adapting them to new tasks with a minimum amount of overhead. LoRA strikes a compromise between performance and flexibility by leaving the majority of the pre-trained model unaltered and only making small, effective modifications.

3.4 Mixed Precision Training

Mixed precision training [9] is a technique used in large neural network training that combines the use of both 16-bit and 32-bit floating-point representations for different parts of the training process. This approach leverages the advantages of lower precision (16-bit) for some computations while still using higher precision (32-bit) where necessary.

As illustrated in Figure 4, during the forward pass, the model converts the FP32 weights to FP16 and computes FP16 activations, and in the backward pass, both activation gradients and weight gradients are calculated in FP16. Subsequently, the FP32 master copy of weights is updated with the FP16 weight gradients. The authors also claim that some operations should be read and written in FP16 but perform arithmetic in FP32 to maintain model accuracy. Read the paper and answer why we need an FP32 master copy of weights and why we need to scale up the loss.

Your answer: 1. In mixed precision training, we aim to strike a balance between memory efficiency and numerical stability. Using 16-bit floating-point numbers (FP16) can reduce memory usage and accelerate training due to their smaller memory footprint.

2. Challenges with FP16: FP16 has a smaller range of possible values compared to FP32, which can lead to precision issues. The weight update operation (e.g., w = w - lr * grad) can suffer from imprecision when gradients (grad) are much smaller than the weights (w).

3. The Solution: We maintain a master copy of the weights in FP32 alongside the model's FP16 weights.

4. During training: Compute the output using the FP16 model. Calculate the loss. Back-propagate gradients in half-precision (FP16). Copy the gradients to the FP32 master model. Update the master model (in FP32 precision). Copy the master model back to the FP16 model.

5. Scaling Up the Loss: When gradients are small (as in FP16), multiplying them by a small learning rate can result in updates that are too tiny to be accurately represented. By scaling up the loss (e.g., multiplying it by a constant factor), we ensure that gradients are large enough to avoid precision issues during weight updates.

FP16 operations offer significant speed advantages over FP32 operations, especially on modern GPUs equipped with NVIDIA's Volta and Ampere architectures. These architectures include dedicated hardware, known as Tensor Cores, designed to accelerate FP16 multiplication and accumulation into either FP16 or FP32 outputs. Thus, it is clear that mixed precision training can reduce computation time by utilizing FP16 computation units. In terms of memory consumption, according to the paper:

"Even though maintaining an additional copy of weights increases the memory requirements for the weights by 50% compared with single precision training, impact on overall memory usage is much smaller. For training memory consumption is dominated by activations, due to larger batch sizes and activations of each layer being saved for reuse in the back-propagation pass. Since activations are also stored in half-precision format, the overall memory consumption for training deep neural networks is roughly halved. "

Do you think this statement still holds for LLM fine-tuning?

Your answer: When it comes to Language Model (LLM) fine-tuning, there are some nuances to consider:

LLMs (such as BERT, GPT, etc.) are pre-trained on large corpora and then fine-tuned on specific downstream tasks. Fine-tuning typically involves training on smaller datasets with task-specific labels. The fine-tuning process may have different memory requirements compared to initial pre-training.

While activations still play a role in memory consumption during fine-tuning, the nature of LLM tasks (e.g., text classification, question answering) may lead to different activation patterns.

Fine-tuning often uses smaller batch sizes than pre-training. Smaller batch sizes reduce memory requirements for activations.

The size of the LLM (number of layers, hidden dimensions, etc.) impacts memory usage. Some fine-tuning tasks may require additional layers or modifications to the base LLM architecture, affecting memory.

The need for an FP32 master copy of weights and loss scaling still holds in LLM fine-tuning. However, the specific impact on memory consumption depends on the fine-tuning setup.

In summary, while the overall memory consumption reduction due to activations remains relevant, the specifics for LLM fine-tuning may vary based on the task, dataset, and model configuration .

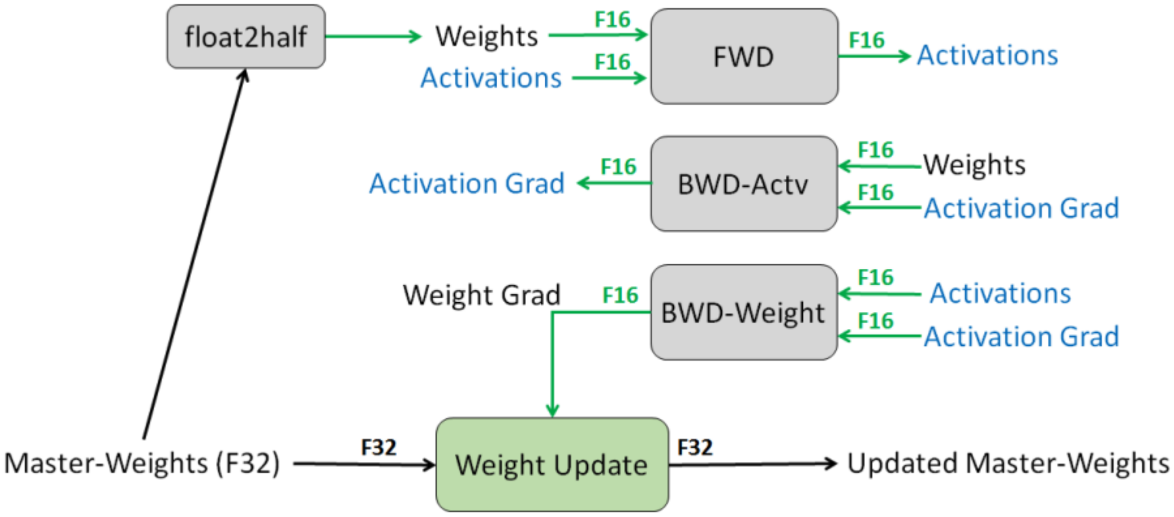


Figure 4: Mixed precision training

4 Phase 1: Preliminary Study

Complete all questions in Sections 2 and 3, and upload your answers as a PDF document named `phase_1` to GitHub. It's strongly advised to complete this phase as soon as possible to allow enough time for the subsequent phases.

5 Phase 2: LLaMA2 Model Inference

In this phase, we focus on understanding how the LLaMA2 7B model generates text and making specific changes to its code. Our goal is to remove the parts of the code that deal with KV-caching. The LLaMA2 7B model and its tokenizer are located at the directory `project/saifhash_1190/llama2-7b` on the CARC system.

Guidelines:

1. Understanding the Code Base: Familiarize yourself with the model's architecture, the tokenization process, batch generation, and the decoding mechanism. Utilize debugging tools to inspect values and flow as necessary.

2. Identifying KV-Caching Features: Look through the code to locate all components and functions related to KV-caching.

3. Modifying the Code: Carefully remove the KV-caching features from the code. Make sure the model can still generate text without these features.
4. Testing: After making changes, test the model with sample prompts to ensure it still works correctly.

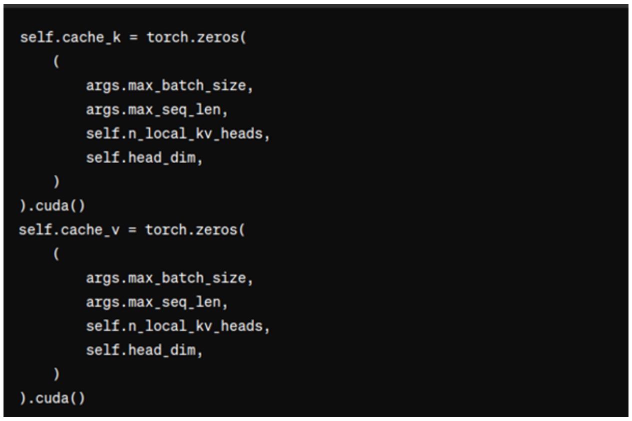
Deliverables: Submit a PDF document named `phase_2` containing the details of the changes you've made to the code, along with the prompts and the outputs from testing the generation process.

5.1 Code Modifications: Eliminating KV Cachings

Updated files has been added in git for reference. We used debugging tool (in vs code) to understand the code.

5.1.1 In Model.py File:

- KV-Caching Mechanisms in Attention Class

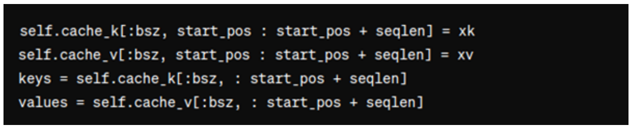


```
self.cache_k = torch.zeros(
    (
        args.max_batch_size,
        args.max_seq_len,
        self.n_local_kv_heads,
        self.head_dim,
    )
).cuda()
self.cache_v = torch.zeros(
    (
        args.max_batch_size,
        args.max_seq_len,
        self.n_local_kv_heads,
        self.head_dim,
    )
).cuda()
```

Figure 5: Original Implementation

  - Updated Implementation: These lines are commented out to remove caching mechanisms that store previous computation values of keys and values across different batches or sequences.
  - Why: KV-caching is typically used to optimize performance when sequences are processed incrementally, e.g., during autoregressive generation. Removing it simplifies the model but may affect the efficiency when processing long sequences or during incremental updates.

- Forward Pass of Attention Class

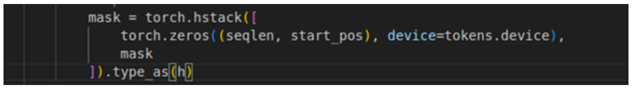


```
self.cache_k[:bsz, start_pos : start_pos + seqlen] = xk
self.cache_v[:bsz, start_pos : start_pos + seqlen] = xv
keys = self.cache_k[:bsz, : start_pos + seqlen]
values = self.cache_v[:bsz, : start_pos + seqlen]
```

Figure 6: Original Implementation

  - Updated Implementation: These lines are also commented out, and instead, keys and values are used directly after being computed.
  - Why: Direct use of keys and values after computation avoids the need for caching, simplifying the architecture and potentially reducing memory usage at the cost of having to recompute these values in each forward pass, which might be computationally more expensive in certain scenarios.

- Mask allocation



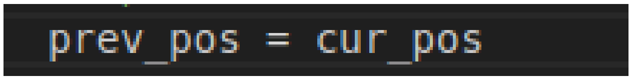
```
mask = torch.hstack([
    torch.zeros((seqlen, start_pos), device=tokens.device),
    mask
]).type_as(h)
```

Figure 7: Original Implementation

  - Updated Implementation: It was commented out. Also, mini changes to fix ”startpos”, which is not needed after removing KV caching, were fixed in model.py and in generation.py.

5.1.2 In Generation.py file:

- Ranges
  - Original Implementation: logits = self(tokens[:, `prev_$_$pos:cur_$_$pos`], prev_pos)
  - Updated Implementation: logits = self(tokens[:, :cur_pos])
  - Why: We don’t need to consider prev_$pos if not doing KV_$-$Caching.
- Memory allocation

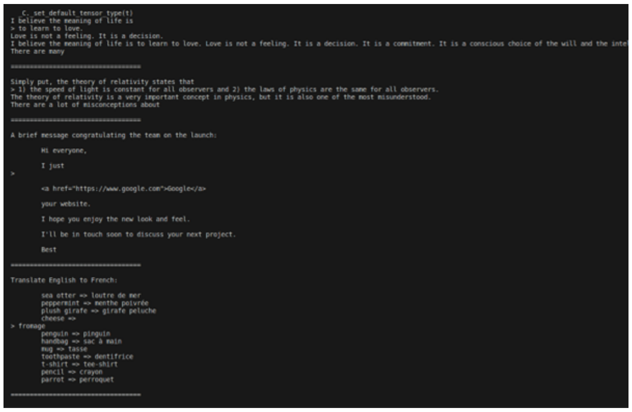


```
prev_pos = cur_pos
```

Figure 8: Original Implementation

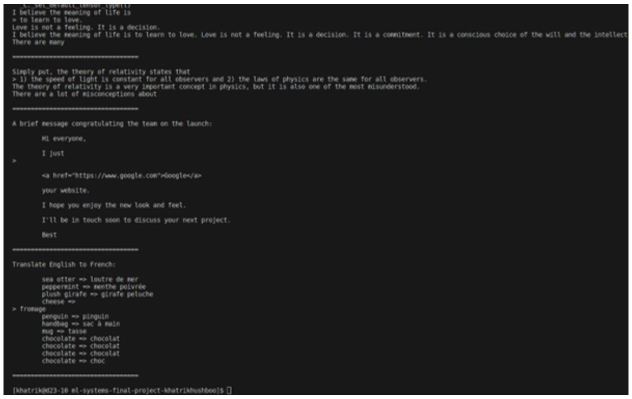
  - Updated Implementation: These lines are commented out. (Self-explanatory)

5.2 Testing the model with given prompts



```
  C._set_default_tensor_type(t)
I believe the meaning of life is
> to learn to love.
Love is not a feeling. It is a decision.
I believe the meaning of life is to learn to love. Love is not a feeling. It is a decision. It is a commitment. It is a conscious choice of the will and the inte
There are many

==================================

Simply put, the theory of relativity states that
> 1) the speed of light is constant for all observers and 2) the laws of physics are the same for all observers.
The theory of relativity is a very important concept in physics, but it is also one of the most misunderstood.
There are a lot of misconceptions about

==================================

A brief message congratulating the team on the launch:

        Hi everyone,

        I just
>

        <a href="https://www.google.com">Google</a>

        your website.

        I hope you enjoy the new look and feel.

        I'll be in touch soon to discuss your next project.

        Best

==================================

Translate English to French:

        sea otter => loutre de mer
        peppermint => menthe poivrée
        plush girafe => girafe peluche
        cheese =>
> fromage
        penguin => pinguin
        handbag => sac à main
        mug => tasse
        toothpaste => dentifrice
        t-shirt => tee-shirt
        pencil => crayon
        parrot => perroquet

==================================
```

Figure 9: Output without KV Caching



```
    _C._set_default_tensor_type(t)
I believe the meaning of life is
> to learn to love.
Love is not a feeling. It is a decision.
I believe the meaning of life is to learn to love. Love is not a feeling. It is a decision. It is a commitment. It is a conscious choice of the will and the intellect
There are many

==================================

Simply put, the theory of relativity states that
> 1) the speed of light is constant for all observers and 2) the laws of physics are the same for all observers.
The theory of relativity is a very important concept in physics, but it is also one of the most misunderstood.
There are a lot of misconceptions about

==================================

A brief message congratulating the team on the launch:

        Hi everyone,

        I just
>

        <a href="https://www.google.com">Google</a>

        your website.

        I hope you enjoy the new look and feel.

        I'll be in touch soon to discuss your next project.

        Best

==================================

Translate English to French:

        sea otter => loutre de mer
        peppermint => menthe poivrée
        plush girafe => girafe peluche
        cheese =>
> fromage
        penguin => pinguin
        handbag => sac à main
        mug => tasse
        chocolate => chocolat
        chocolate => chocolat
        chocolate => chocolat
        chocolate => choc

==================================

[khatrik@d23-10 ml-systems-final-project-khatrikhushboo]$ ▯
```

Figure 10: Output with KV Caching

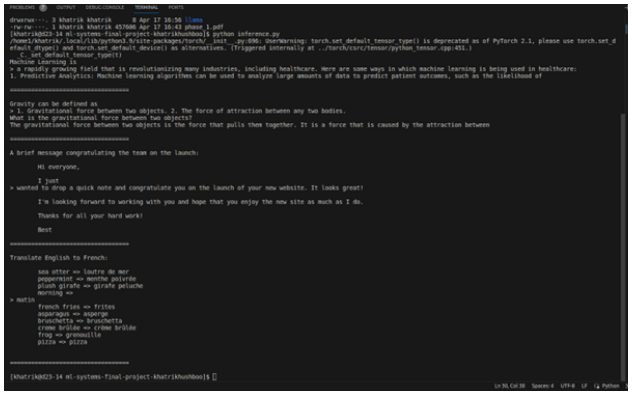
I believe the meaning of life is
> to learn to love.
Love is not a feeling. It is a decision.
I believe the meaning of life is to learn to love. Love is not a feeling. It is a decision. It is a commitment. It is a conscious choice of the will and the intellect.
There are many

Simply put, the theory of relativity states that
> 1) the speed of light is constant for all observers and 2) the laws of physics are the same for all observers.
The theory of relativity is a very important concept in physics, but it is also one of the most misunderstood.
There are a lot of misconceptions about

A brief message congratulating the team on the launch:
Hi everyone,
I just
>
¡a href=”https://www.google.com”¿Google¡/a¿
your website.
I hope you enjoy the new look and feel.
I’ll be in touch soon to discuss your next project.
Best

Translate English to French:
sea otter ⇒ loutre de mer
peppermint ⇒ menthe poivrée
plush girafe ⇒ girafe peluche
cheese ⇒ fromage
penguin ⇒ pinguin
handbag ⇒ sac à main
mug ⇒ tasse
chocolate ⇒ chocolat
chocolate ⇒ chocolat
chocolate ⇒ chocolat
chocolate ⇒ choc

5.3 Testing the model with customised Prompts :



```
drwxrwx---. 3 khatrik khatrik      8 Apr 17 16:56 llama
-rw-rw----. 1 khatrik khatrik 457606 Apr 17 16:43 phase_1.pdf
[khatrik@d23-14 ml-systems-final-project-khatrikhushboo]$ python inference.py
/home1/khatrik/.local/lib/python3.9/site-packages/torch/__init__.py:696: UserWarning: torch.set_default_tensor_type() is deprecated as of PyTorch 2.1, please use torch.set_d
efault_dtype() and torch.set_default_device() as alternatives. (Triggered internally at ../torch/csrc/tensor/python_tensor.cpp:451.)
  _C._set_default_tensor_type(t)
Machine Learning is
> a rapidly growing field that is revolutionizing many industries, including healthcare. Here are some ways in which machine learning is being used in healthcare:
1. Predictive Analytics: Machine learning algorithms can be used to analyze large amounts of data to predict patient outcomes, such as the likelihood of

==================================

Gravity can be defined as
> 1. Gravitational force between two objects. 2. The force of attraction between any two bodies.
What is the gravitational force between two objects?
The gravitational force between two objects is the force that pulls them together. It is a force that is caused by the attraction between

==================================

A brief message congratulating the team on the launch:

        Hi everyone,

        I just
> wanted to drop a quick note and congratulate you on the launch of your new website. It looks great!

        I'm looking forward to working with you and hope that you enjoy the new site as much as I do.

        Thanks for all your hard work!

        Best

==================================

Translate English to French:

        sea otter => loutre de mer
        peppermint => menthe poivrée
        plush girafe => girafe peluche
        morning =>
> matin
        french fries => frites
        asparagus => asperge
        bruschetta => bruschetta
        creme brûlée => crème brûlée
        frog => grenouille
        pizza => pizza


==================================

[khatrik@d23-14 ml-systems-final-project-khatrikhushboo]$ 
```

Figure 11: Custom Prompts I

Machine Learning is
> a rapidly growing field that is revolutionizing many industries, including healthcare. Here are some ways in which machine learning is being used in healthcare:

1. Predictive Analytics: Machine learning algorithms can be used to analyze large amounts of data to predict patient outcomes, such as the likelihood of

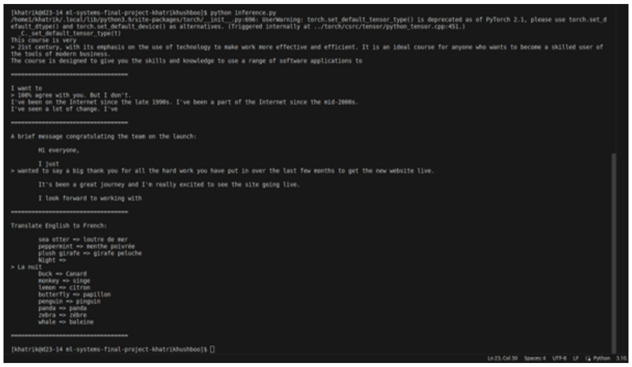
---

Gravity can be defined as
> 1. Gravitational force between two objects. 2. The force of attraction between any two bodies.
What is the gravitational force between two objects?
The gravitational force between two objects is the force that pulls them together. It is a force that is caused by the attraction between

---

A brief message congratulating the team on the launch:
Hi everyone,
I just
> wanted to drop a quick note and congratulate you on the launch of your new website. It looks great!
I'm looking forward to working with you and hope that you enjoy the new site as much as I do.
Thanks for all your hard work!
Best

Translate English to French:
sea otter ⇒ loutre de mer
peppermint ⇒ menthe poivrée
plush girafe ⇒ girafe peluche
morning ⇒ matin
french fries ⇒ frites
asparagus ⇒ asperge
bruschetta ⇒ bruschetta
creme brûlée ⇒ crème brûlée
frog ⇒ grenouille
pizza ⇒ pizza



```
[khatrik@d23-14 ml-systems-final-project-khatrikhushboo]$ python inference.py
/home1/khatrik/.local/lib/python3.9/site-packages/torch/__init__.py:696: UserWarning: torch.set_default_tensor_type() is deprecated as of PyTorch 2.1, please use torch.set_d
efault_dtype() and torch.set_default_device() as alternatives. (Triggered internally at ../torch/csrc/tensor/python_tensor.cpp:451.)
  _C._set_default_tensor_type(t)
This course is very
> 21st century, with its emphasis on the use of technology to make work more effective and efficient. It is an ideal course for anyone who wants to become a skilled user of
the tools of modern business.
The course is designed to give you the skills and knowledge to use a range of software applications to

==================================

I want to
> 100% agree with you. But I don't.
I've been on the Internet since the late 1990s. I've been a part of the Internet since the mid-2000s.
I've seen a lot of change. I've

==================================

A brief message congratulating the team on the launch:

        Hi everyone,

        I just
> wanted to say a big thank you for all the hard work you have put in over the last few months to get the new website live.

        It's been a great journey and I'm really excited to see the site going live.

        I look forward to working with

==================================

Translate English to French:

        sea otter => loutre de mer
        peppermint => menthe poivrée
        plush girafe => girafe peluche
        Night =>
> La nuit
        Duck => Canard
        monkey => singe
        lemon => citron
        butterfly => papillon
        penguin => pinguin
        panda => panda
        zebra => zébre
        whale => baleine

==================================

[khatrik@d23-14 ml-systems-final-project-khatrikhushboo]$ 
```

Figure 12: Custom Prompts II

This course is very
> 21st century, with its emphasis on the use of technology to make work more effective and efficient. It is an ideal course for anyone who wants to become a skilled user of the tools of modern business.
The course is designed to give you the skills and knowledge to use a range of software applications to

---

I want to
> 100% agree with you. But I don't.
I've been on the Internet since the late 1990s. I've been a part of the Internet since the mid-2000s.
I've seen a lot of change. I've

A brief message congratulating the team on the launch:
Hi everyone,
I just
> wanted to say a big thank you for all the hard work you have put in over the last few months to get the new website live.
It's been a great journey and I'm really excited to see the site going live.
I look forward to working with

Translate English to French:
sea otter ⇒ loutre de mer
peppermint ⇒ menthe poivrée
plush girafe ⇒ girafe peluche
Night ⇒ La nuit
Duck ⇒ Canard
monkey ⇒ singe
lemon ⇒ citron
butterfly ⇒ papillon
penguin ⇒ pinguin
panda ⇒ panda
zebra ⇒ zébre
whale ⇒ baleine

5.4 Impact of Changes

General Impact of Changes:

Removing KV-caching from the model simplifies the overall architecture but can lead to increased computational overhead during inference and training, as each pass now requires full recomputation of attention keys and values. This trade-off needs to be considered based on specific use cases, such as whether the model processes sequences incrementally or in full batches. These modifications ensure that the model's behavior remains consistent in terms of output while potentially increasing the computational demands during its operation.

Differences in Outputs:

It's entirely possible to see differences in the outputs before and after changes to KV-caching because:

1. Caching Alters State: KV-caching means the model retains certain states between forward passes. Removing this can change how states are calculated, as without caching, the model starts "fresh" each time.

2. Different Computational Paths: Even if you're processing the same input data, the absence of caching means the model doesn't have access to precomputed keys and values from prior tokens or batches, possibly leading to different attention patterns and outputs.

3. Initializations and Randomness: If the model or any part of it is re-initialized or involves randomness, outputs can vary. This could be due to different initial states or random seeds.

6 Phase 3: LLaMA2 Model Training

In this phase, your task is to conduct instruction tuning on the LLaMA2 7B model using the (partial) Alpaca dataset, which enhances and expands the capabilities of the LLaMA2 model. By applying all previously discussed optimization techniques, fine-tuning LLMs will become feasible, even on a single GPU. We will avoid using high-level libraries such as HuggingFace, which abstracts away many important details. Instead, we will solely rely on PyTorch's built-in functions. This approach will ensure you gain a comprehensive understanding of the process involved in the instruction tuning of LLMs.

Guidelines:

1. Initial Setup: Begin by instantiating only 1 decoder layer of the LLaMA2 model. Test all your code on a less powerful yet more accessible GPU, the P100 on CARC. This approach prevents out-of-memory errors even before applying optimization techniques. Once you've ensured an end-to-end training process that is bug-free and incorporates all four optimization techniques, you can scale up to the full-size LLaMA2 model, which includes 32 decoder layers, and utilize a more powerful GPU, the A100, with 40GB of RAM, on CARC system.

2. End-to-End Instruction Tuning Flow: Create a file named `finetuning.py`. Implement the end-to-end instruction tuning workflow in PyTorch. For data preprocessing and the creation of supervised data loaders, refer to the official Alpaca repo.

3. Training Iteration Loop: Replace the HuggingFace `trainer` object used in the Alpaca repo with your own implementation from scratch. To compute loss using PyTorch functions, refer to the HuggingFace's implementation.

4. Gradient Accumulation and Mixed Precision Training: Implement gradient accumulation and mixed precision training using PyTorch's AMP package. Refer to the AMP Recipe and AMP Examples for guidance.

5. LoRA Linear Layer Module: Implement the LoRA linear layer module by referring to the official implementation. Create a file named `lora.py` under the `llama/model` path. Convert your model into a PEFT model by replacing the Q and V projection layers in the LLaMA2 model with your LoRA `Linear` modules. Freeze all model parameters except for `LoRA_A` and `LoRA_B` and report the percentage of trainable parameters.

6. Gradient Checkpointing: Utilize PyTorch's `torch.utils.checkpoint` API for inserting checkpoints. Decide which layer(s) to apply checkpointing to and describe your approach in the report. Refer to this tutorial for more information.

7. Model Fine-Tuning: Apply all the aforementioned techniques to your LLaMA2 7B model and fine-tune it on the Alpaca Dataset using the A100 GPU. To manage time constraints for educational purposes, extract the first 200 samples from the dataset as your training set.

8. Hyperparameters: Set $learning\_rate = 1e - 5$, $batch\_size = 1$ and $gradient\_accumulation\_step = 8$. For LoRA configuration, set $r = 16$, $alpha = 32$, and $dropout\_rate = 0.05$.

Deliverables: Submit a PDF file named `phase_3` that includes the implementation details you have completed, along with the prompt and the result used to test the fine-tuned model. In the same PDF file, provide the following analysis and measurements. For Table 1, fill in the blanks with ↑ to signify an increase in resource usage, ↓ for a decrease, or − to indicate no change. For Table 2, measure and report the peak memory usage in MB and the runtime in seconds required to process only 200 training samples on the A100 GPU across each combination of techniques applied in the instruction tuning of the LLaMA2 7B model. If an out-of-memory error happens, then simply mark × in the blanks. You can turn off gradient checkpointing for this step.

		Grad. Accumulation	Grad. Checkpoint	Mixed Precision	LoRA
Memory	parameter	−	−	↓	↓
	activation	−	↓	↓	−
	gradient	−	−	↓	↓
	optimizer state	−	−	↓	↓
Computation		↓	↑	↓	↓

Table 1: System performance analysis

GA	OFF				ON			
MP	OFF		ON		OFF		ON	
LoRA	OFF	ON	OFF	ON	OFF	ON	OFF	ON
Peak Mem	X	27131.95	X	33217.96	X	27163.95	X	33249.97
Runtime	X	328.49	X	175.99	X	326.97	X	192.98

Table 2: System performance measurement

References

[1] W. X. Zhao, K. Zhou, J. Li, T. Tang, X. Wang, Y. Hou, Y. Min, B. Zhang, J. Zhang, Z. Dong, Y. Du, C. Yang, Y. Chen, Z. Chen, J. Jiang, R. Ren, Y. Li, X. Tang, Z. Liu, P. Liu, J.-Y. Nie, and J.-R. Wen, "A survey of large language models," 2023.

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

[3] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale, D. Bikel, L. Blecher, C. C. Ferrer, M. Chen, G. Cucurull, D. Esiobu, J. Fernandes, J. Fu, W. Fu, B. Fuller, C. Gao, V. Goswami, N. Goyal, A. Hartshorn, S. Hosseini, R. Hou, H. Inan, M. Kardas, V. Kerkez, M. Khabsa, I. Kloumann, A. Korenev, P. S. Koura, M.-A. Lachaux, T. Lavril, J. Lee, D. Liskovich, Y. Lu, Y. Mao, X. Martinet, T. Mihaylov, P. Mishra, I. Molybog, Y. Nie, A. Poulton, J. Reizenstein, R. Rungta, K. Saladi, A. Schelten, R. Silva, E. M. Smith, R. Subramanian, X. E. Tan, B. Tang, R. Taylor, A. Williams, J. X. Kuan, P. Xu, Z. Yan, I. Zarov, Y. Zhang, A. Fan, M. Kambadur, S. Narang, A. Rodriguez, R. Stojnic, S. Edunov, and T. Scialom, "Llama 2: Open foundation and fine-tuned chat models," 2023.

[4] "Huggingface datasets tatsu-lab/alpaca." [Online]. Available: https://huggingface.co/datasets/tatsu-lab/alpaca

[5] "How to generate text: using different decoding methods for language generation with transformers." [Online]. Available: https://huggingface.co/blog/how-to-generate

[6] "Perplexity of fixed-length models." [Online]. Available: https://huggingface.co/docs/transformers/perplexity

[7] "Fitting larger networks into memory." [Online]. Available: https://medium.com/tensorflow/fitting-larger-networks-into-memory-583e3c758ff9

[8] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen, "Lora: Low-rank adaptation of large language models," 2021.

[9] P. Micikevicius, S. Narang, J. Alben, G. Diamos, E. Elsen, D. Garcia, B. Ginsburg, M. Houston, O. Kuchaiev, G. Venkatesh, and H. Wu, "Mixed precision training," 2018.