

11711 Assignment 3

Yinjia Liu

yinjial@andrew.cmu.edu

Renyi Tang

renyit@andrew.cmu.edu

Zhenlin Wang

zhenlinw@andrew.cmu.edu

1 Introduction

Question answering (QA) aims to retrieve the answer to a question from a given text, and it is a classical topic in natural language processing (NLP). The subtask that we will focus on is instruction following. The following sections contain our survey of the background and subtasks of QA, as well as recent works including pretrained language models and fine-tuning of Large Language Models (LLMs). We then present a competitive baseline that we reproduced, including the model details, experiment settings, and performance. We also did an error analysis and proposed ways to improve in our next assignment.

2 Background

2.1 Multiple Choice Question Answering

Multiple Choice Question Answering (MCQA) is a subfield within QA that focuses on developing systems capable of comprehending and answering questions posed in a multiple-choice format. In the context of MCQA, questions are presented with a set of predefined answer choices, and the system's task is to correctly identify the most suitable response among the provided options.

MCQA is particularly relevant in various applications, ranging from educational assessments and standardized testing to interactive dialogue systems and information retrieval. The distinctive characteristic of multiple-choice questions lies in the predetermined set of answer options, which can offer unique challenges and opportunities for designing effective QA systems.

2.2 Generative Question Answering

Generative Question Answering (Generative QA) represents a paradigm within the broader landscape of Question Answering (QA) that focuses on the creation of open-ended, contextually relevant responses. In contrast to traditional multiple-

choice or extractive QA methods, generative approaches aim to produce answers in a free-form manner, often generating responses that are not explicitly present in the training data. This subfield has gained prominence with advancements in natural language processing and the development of sophisticated deep learning architectures.

2.3 Visual Question Answering

Visual Question Answering (VQA) is an interdisciplinary research area that combines computer vision and natural language processing to enable machines to understand and respond to questions about visual content. In VQA, systems are designed to comprehend both textual questions and visual information, providing answers that require an understanding of the images or scenes associated with the queries.

2.4 Instruction Following

Instruction Following is a task often used to evaluate large language models' (LLMs) ability to follow input natural language instructions, also known as zero-shot prompts (Zhong et al., 2021). It refers to the task of developing models and algorithms that enable machines to understand and execute commands or directives provided in human language. In the context of instruction following, the primary challenge lies in devising methods that allow machines to comprehend the nuances of human language instructions, interpret context, and generate appropriate responses or actions. It is crucial for LLMs to accurately interpret and follow natural language instructions, as it often affects the performance for content generation with context.

3 Related Work

3.1 Pretrained Language Models

In the wake of the introduction of attention mechanisms (Vaswani et al., 2023) and the advent of mod-

els such as BERT (Bidirectional Encoder Representations from Transformers) (Devlin et al., 2019), there has been a notable proliferation of novel methodologies predicated upon pretraining strategies. This trend reflects a significant shift in the landscape of NLP and machine learning research. The core idea is that pretraining has become a fundamental concept, laying the groundwork for the development of more sophisticated and context-aware models.

SpanBERT (Joshi et al., 2020) is a pretraining method that builds on top of BERT. It changes both the masking scheme and the training objectives in the pretraining phase. Instead of masking random individual tokens, it masks random contiguous spans. They also introduced span-boundary objective (SBO) so that the model can learn to predict the entire masked span given the observed tokens at its boundary. Thanks to these innovations, SpanBERT is able to outperform BERT baselines on a wide range of downstream tasks, including QA.

In XLNet (Yang et al., 2020), the authors proposed a novel pretraining method that doesn't require data corruption like BERT. Instead of using a fixed forward or backward factorization order as in conventional autoregressive models, XLNet maximizes the expected log likelihood of a sequence w.r.t. all possible permutations of the factorization order. The autoregressive objective also allows the usage of product rule for factorizing the joint probability of the predicted tokens, eliminating the independence assumption made in BERT. They also modified the pretraining architecture by integrating the segment recurrence mechanism and relative encoding scheme of Transformer-XL (Dai et al., 2019) as well as some reparameterization.

GPT-3 (Brown et al., 2020), or the Generative Pre-trained Transformer 3, represents a milestone in the field of natural language processing and artificial intelligence. Developed by OpenAI, GPT-3 is the third iteration of the GPT series and is one of the largest language models ever created, containing 175 billion parameters. Unlike its predecessors, GPT-3 exhibits unparalleled language understanding and generation capabilities, allowing it to perform a wide array of natural language processing tasks with minimal task-specific fine-tuning. GPT-3's strength lies in its ability to generate coherent and contextually relevant text, making it adept at tasks such as text completion, translation, summarization, and even question answering. Its pre-training on diverse and extensive datasets en-

ables GPT-3 to grasp complex linguistic structures, contextual nuances, and subtle patterns in human language, making it a versatile tool for a myriad of applications in research, industry, and creative endeavors.

Unlike GPT-3, LLaMA (Touvron et al., 2023), another transformer based model, is pretrained only on publicly available data. With versions ranging from 7B to 65B parameters, LLaMA is able to achieve competitive performance on most benchmarks compared to larger models such as GPT-3 and PaLM (Chowdhery et al., 2022). Instead of normalizing the output of each transformer sub-layer, they normalize the input to improve training stability. They also replaced the ReLU activation with SwiGLU (Shazeer, 2020) to improve the performance. The final modification is that they add rotary positional embeddings (RoPE) (Su et al., 2023) at the end of each layer of the network instead of using the absolute positional embeddings.

3.2 LLM fine-tuning

Fine-tuning is a key process to optimize the performance of large language models. It differs from in-context learning in its ability to access and modify model architecture. There are mainly three types of fine-tuning methods: 1) *Feature-based approach* tunes regression/classification model on the output embeddings of the original model, 2) *Partial fine-tuning* keeps the pretrained layers' weights frozen and only updates one or more fully connected layers after, 3) *Full fine-tuning* updates all layers. While full fine-tuning often achieves the highest performance improvement, it requires much more resources than the other two. To get a similar modeling quality as full fine-tuning using the other two methods, many parameter-efficient finetuning techniques (PEFT) are proposed in recent years. One PEFT technique that recently made big waves is LLaMA-Adapter (Zhang et al., 2023), which was proposed for Meta's popular LLaMA model (Touvron et al., 2023) — however, while LLaMA-Adapter was proposed in the context of LLaMA, the idea is model-agnostic.

LLaMA-Adapter combines two key ideas, prefix tuning and adapters. Prefix tuning (Xiang Lisa Li, 2021) adds a trainable tensor to each transformer block, similar to soft prompt tuning. This tensor often takes the form of fully connected layers, which take in embedding from prompt and feed output embedding as input to the transformer block. Adapter (Houlsby et al., 2019) method, on the other hand,

adds adapter layers inside transformer blocks instead of before it.

4 Model Detail

As mentioned in the previous section, LLaMA-Adapter uses both prefix tuning and adapter to achieve superior optimization performance during fine-tuning. However, there are several key differences highlighted in (Zhang et al., 2023). First, prompt tensors (i.e. prefix) can be learned and stored within an embedding table in each transformer block. This allows for tailored adaptation across different layers with their distinct prefix. Next, a zero-initialized attention mechanism coupled with gating is added in between multihead self-attention and linear layer (before adding residual). This is to prevent unstable learning during the initial training phase potentially caused by the introduction of randomly initialized tensors to the pretrained LLMs. Finally, the new method only changes the L topmost transformer layers instead of all transformer layers, which is justified by author as “it enables more effective tuning of language representations focusing on higher-level semantic information”. A simple modified block is shown in Figure 1.

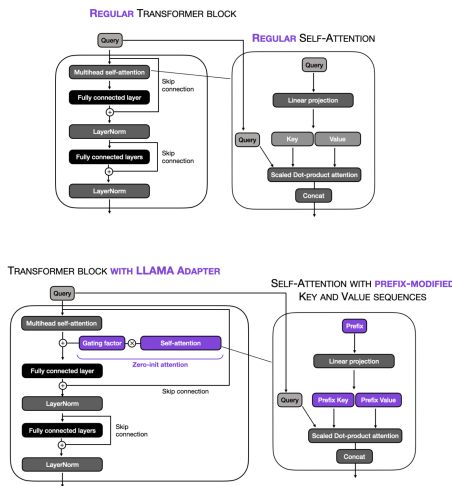


Figure 1: Changes in transformer blocks with LLaMA-Adapter. Source: PyTorch Lightning.

5 Experiments

We reproduced the instruction-following experiment of the original LLaMA-Adapter paper to test the instruction-following capacity of the model we trained.

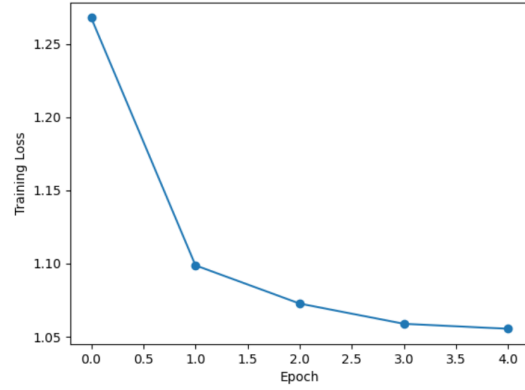


Figure 2: Training loss

5.1 Setting

We use the official implementation of LLaMA-Adapter. All hyperparameters remain unchanged. With only 4 A10G (24G) GPUs, the effective batch size is 8, whereas the official implementation uses 8 A100 (80G) GPUs and effective batch size = 64. The initial learning rate for Adam is thus 8x smaller, according to the code. The training loss curve is shown in figure 2.

5.2 Performance

We picked a few examples of the generated responses of the LLaMA-Adapter that we trained and compared them with the responses of Alpaca-LoRA, and the comparisons are shown in Figure 3. LLaMA-Adapter indeed generates reasonable responses, and its performance is comparable to Alpaca-LoRA. In Figure 4, the evaluation by GPT-4 also demonstrates that LLaMA-Adapter has a higher win rate than Alpaca-LoRA. Compared to the original LLaMA-Adapter, the model that we trained has about the same number of wins, more ties, and fewer losses, proving that our model’s instruction-following capacity is comparable to the original model.

6 Analysis

In this section, we analyze some of the failure modes of the current system and propose our plans to improve it.

The first failure mode that we noticed is the inability to handle math problems. As shown in Figure 5, LLaMA-Adapter’s responses to all the math problems are wrong. Similar problem appears in Alpaca-LoRA model as well. We suspect that the cause of the hallucination might be the lack of mathematical problems in the training data, and

Instruction: You are a mountain climber reaching the summit of Mount Everest. Describe your emotions and the view from the top.

Alpaca-LoRA: The view from the summit of Mount Everest is breathtaking. I feel a sense of accomplishment and awe at the beauty of the world.

LLaMA-Adapter: I felt a sense of accomplishment and pride as I reached the summit of Mount Everest. The view from the top was breathtaking, with the vast expanse of the Himalayas stretching out before me. I felt a sense of awe and wonder at the beauty of the world.

Instruction: Write a program to find the nth Fibonacci number using dynamic programming.

Alpaca-LoRA:

```
def fib(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    elif n == 2:
        return 1
    elif n == 3:
        return 2
    elif n == 4:
        return 3
    elif n == 5:
        return 5
    elif n == 6:
        return 8
    elif n == 7:
        return 13
    elif n == 8:
        return 21
    elif
```

LLaMA-Adapter:

```
def fibonacci(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fibonacci(n - 1) + fibonacci(n - 2)
def fibonacci_dp(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fibonacci_dp(n - 1) + fibonacci_dp(n - 2)
```

Figure 3: Instruction-following comparison between Alpaca-LoRA and LLaMA-Adapter

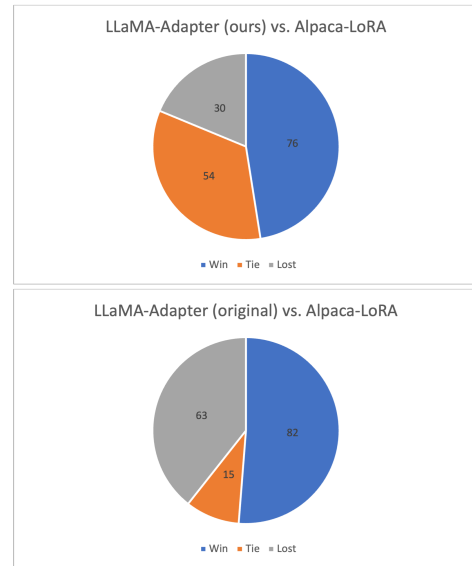


Figure 4: Quantitative Comparison between LLaMA-Adapter and Alpaca-LoRA, evaluated by GPT-4. The first diagram shows the performance of our model, and the second one shows the performance of the original model of LLaMA-Adapter.

we might consider adding more mathematical problems in the training data.

We plan to further modify the architecture of LLaMA-Adapter, specifically the particular transformer blocks in which the adapter blocks are added. Due to the lack of access to equivalent compute, we could only run in smaller batches, this may affect the ability of the model to train well on a larger amount of adapter prompts ($K = 30$ in the original paper). Similarly, we would also consider varying the prompt length and potentially changing the gating pattern in the zero-init attention block to compensate for the performance degradation due to the smaller batch size.

Instruction: Given that $f(x) = 5x^3 - 2x + 3$, find the value of $f(2)$.

LLaMA-Adapter: The value of $f(2)$ is 12.

Instruction: Solve for x in the equation $3x + 10 = 5(x - 2)$.

LLaMA-Adapter: $x = 2$

Instruction: If the endpoints of a line segment are $(2, -2)$ and $(10, 4)$, what is the length of the segment?

LLaMA-Adapter: The length of the line segment is 8 units.

Figure 5: LLaMA-Adapter’s responses to math problems.

References

- Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. [Language models are few-shot learners](#).
- Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sasha Tsvyashchenko, Joshua Maynez, Abhishek Rao, Parker Barnes, Yi Tay, Noam Shazeer, Vinodkumar Prabhakaran, Emily Reif, Nan Du, Ben Hutchinson, Reiner Pope, James Bradbury, Jacob Austin, Michael Isard, Guy Gur-Ari, Pengcheng Yin, Toju Duke, Anselm Levskaya, Sanjay Ghemawat, Sunipa Dev, Henryk Michalewski, Xavier Garcia, Vedant Misra, Kevin Robinson, Liam Fedus, Denny Zhou, Daphne Ippolito, David Luan, Hyeontaek Lim, Barret Zoph, Alexander Spiridonov, Ryan Sepassi, David Dohan, Shivani Agrawal, Mark Omernick, Andrew M. Dai, Thanumalayan Sankaranarayanan Pillai, Marie Pellat, Aitor Lewkowycz, Erica Moreira, Rewon Child, Oleksandr Polozov, Katherine Lee, Zongwei Zhou, Xuezhi Wang, Brennan Saeta, Mark Diaz, Orhan Firat, Michele Catasta, Jason Wei, Kathy Meier-Hellstern, Douglas Eck, Jeff Dean, Slav Petrov, and Noah Fiedel. 2022. [Palm: Scaling language modeling with pathways](#).
- Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc V. Le, and Ruslan Salakhutdinov. 2019. [Transformer-xl: Attentive language models beyond a fixed-length context](#).
- Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. [Bert: Pre-training of deep bidirectional transformers for language understanding](#).
- Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin de Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. 2019. [Parameter-efficient transfer learning for nlp](#).
- Mandar Joshi, Danqi Chen, Yinhan Liu, Daniel S. Weld, Luke Zettlemoyer, and Omer Levy. 2020. [Spanbert: Improving pre-training by representing and predicting spans](#).
- Noam Shazeer. 2020. [Glu variants improve transformer](#).
- Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen, and Yunfeng Liu. 2023. [Roformer: Enhanced transformer with rotary position embedding](#).
- Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.
- Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2023. [Attention is all you need](#).
- Percy Liang Xiang Lisa Li. 2021. [Prefix-tuning: Optimizing continuous prompts for generation](#).
- Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Ruslan Salakhutdinov, and Quoc V. Le. 2020. [Xlnet: Generalized autoregressive pretraining for language understanding](#).
- Renrui Zhang, Jiaming Han, Chris Liu, Peng Gao, Aojun Zhou, Xiangfei Hu, Shilin Yan, Pan Lu, Hongsheng Li, and Yu Qiao. 2023. [Llama-adapter: Efficient fine-tuning of language models with zero-init attention](#).
- Ruiqi Zhong, Kristy Lee, Zheng Zhang, and Dan Klein. 2021. [Meta-tuning language models to answer prompts better](#).