Fine-tuning an LLM On Recommendation Task

Halil İbrahim Ergül, Ahmet Yasin Aytar

Abstract— **In this study, we explored the utility of Stanford's Alpaca and Alpaca-Lora, instruction-following large language models, in a recommendation system setting. Our approach leveraged a fine-tuned dataset consisting of 'input', 'instruction', and 'output' components, inspired by the Stanford Alpaca fine-tuning framework. The application of LORA (Low-Rank Adaptation), a technique that allows efficient fine-tuning of large pre-trained language models, was also utilized to enhance the performance of our system. We tested multiple configurations of hyperparameters and evaluated the performance based on training and validation loss. Encouragingly, all configurations demonstrated decreasing loss values over time, signaling effective learning during the fine-tuning process. However, despite these positive indications, the performance during the inference phase was not as good as expected. The models demonstrated a tendency to repeat the input data rather than generating diverse recommendations, limiting their applicability in a real-world setting. The challenges faced during the inference phase emphasize the complexity of creating effective instruction-following models and highlight the need for more substantial computational resources. Future work will seek to address these limitations and further explore the potential of large language models in recommendation tasks.**

Keywords — Large Language Model; LoRA; Alpaca; Instrucion-Following; Recommendation Systems.

I. INTRODUCTION

The rise of Large Language Models in the computational realm shows a revolutionary development in the world of artificial intelligence. These tools help computers grasp and use human language to generate different kinds of content. By extending their utility across a broad range of applications, such as content generation, language transition, and answering questions, these tools have definitely become a crucial part of many modern technological solutions. In recent months, it has been seen an extensive effort to take advantage of these LLMs in new ways. These models are tweaked and fine-tuned to utilize in even more specific domains. This adjustment and trend in the models' functionality opens up a wide range of possibilities. Not only this fine-tuning introduce these models to new contexts, but also boosts their performance within specific areas.

Recommendation systems are one of the domains that have benefitted from this trend. These systems aim at enhancing digital experiences across sectors by analyzing user preferences and recommending products as a result of this analysis accordingly. They typically employ one of two strategies, which are collaborative filtering or content-based filtering. Collaborative filtering suggests items to a user that similar users have liked in the past. This method assumes that similar users behave similarly, and it is broadly used in many online platforms [5]. On the other hand, content-based filtering takes advantage of item features to make recommendations. For example, if a user often purchases mystery novels on a book site, the system will recommend other mystery novels [4].

Both techniques generally make use of tabular data and have some limitations. However, the potential of LLMs in shaping the performance of recommendation systems is very crucial and these limitations can be overcome with the help of LLMs' extensive capabilities. Given their abilities to understand and generate the text in a human-like manner, fine-tuned LLMs for recommendation systems could leverage the potential of these systems to create more effective and nuanced recommendations. Not only do the fine-tuned recommendation systems utilize the tabular data and mentioned two techniques above, but they also make use of text-like data such as product reviews and understand key features for specific customers. Besides, they can generate recommendations in a human-like manner by explaining why it recommends a specific product to a certain customer.

II. LITERATURE AND RESEARCH QUESTION

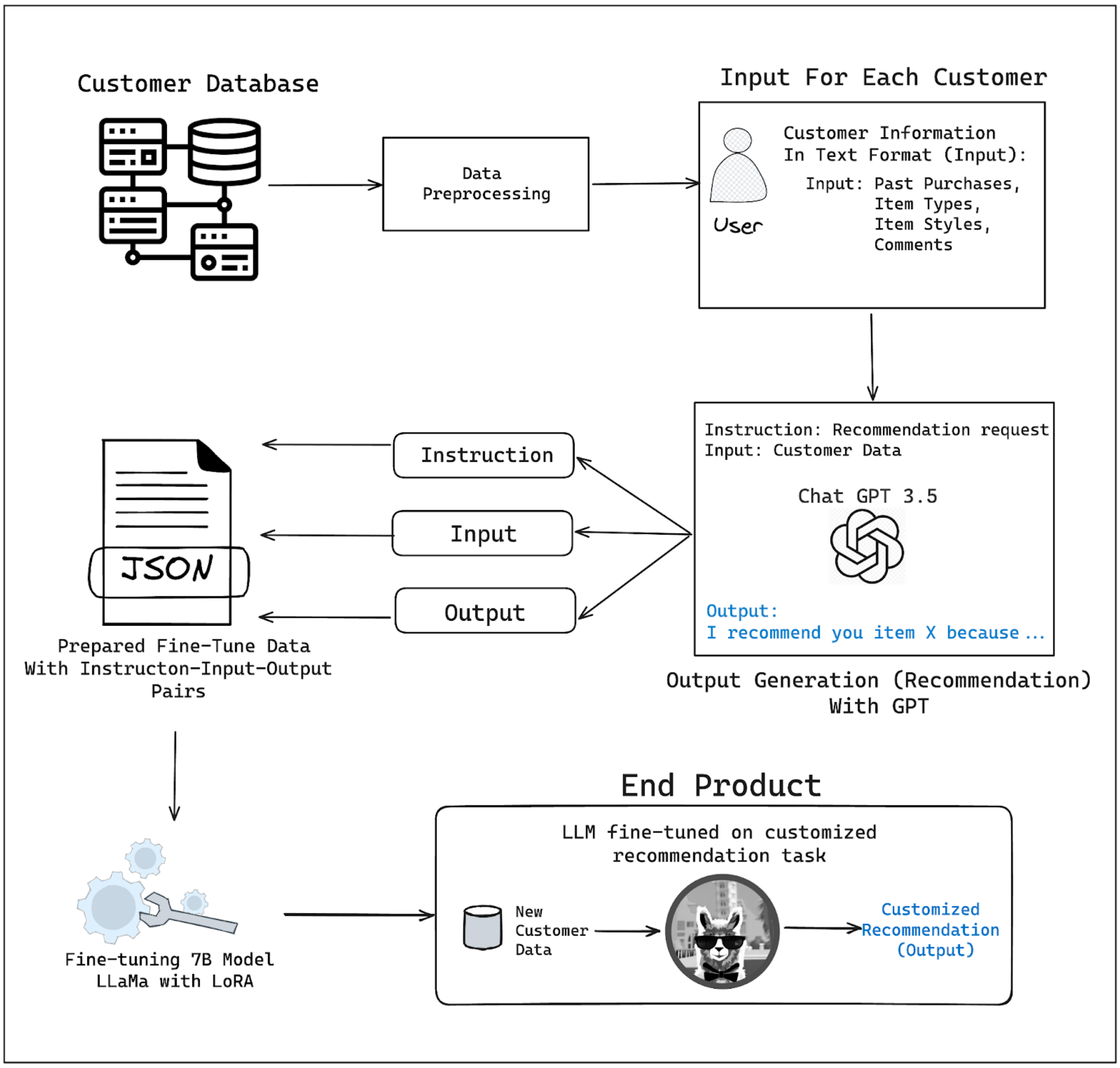
The development of LLMs has opened new opportunities for the development of recommendation systems. Some different approaches and studies have explored this potential to finetune LLMs for recommendation tasks.

Cui et al. developed M6-Rec, a foundation model for recommendation systems based on Alibaba's pre-trained LLM, M6 [1]. They used this model to turn various tasks into language problems, such as suggesting items to users or predicting user behavior. This approach did not depend on user or item IDs, which allows it to be used in encountering new items. Also in another study, Zhang et al. transformed recommendation tasks into a language fill-in-the-blank game [10]. They used users' past interactions to predict what the user might interact with next. Although their model achieved better results than random suggestions in unfamiliar scenarios, it did not perform well when some data about the task was available.

Wang et al. (2023) introduced a three-step strategy for suggesting the next item to a user in situations where the model has no prior knowledge [9]. First, they create a list of potential items, then the model narrows down this list based on the user's preferences and finally recommends items from this refined list. They use a straightforward method to extract these final recommendations from the model's response.

Gao et al. (2023) developed a system called Chat-REC that combines a standard recommender system with the Chat-GPT model [2]. The conventional recommendation system provides a large list of potential items, and then Chat-GPT reduces this list to generate the final recommendations.

These studies emphasize the potential of LLMs to enhance recommendation systems, specifically their ability to understand



Customer Database
Data Preprocessing
Input For Each Customer
Customer Information In Text Format (Input):
Input: Past Purchases, Item Types, Item Styles, Comments
User
Instruction: Recommendation request
Input: Customer Data
Chat GPT 3.5
Output:
I recommend you item X because ...
Output Generation (Recommendation) With GPT
Instruction
Input
Output
JSON
Prepared Fine-Tune Data With Instructon-Input-Output Pairs
End Product
LLM fine-tuned on customized recommendation task
New Customer Data
Customized Recommendation (Output)
Fine-tuning 7B Model LLaMa with LoRA

Figure 1: Project Pipeline Flowchart

text-like data and generate human-like recommendations. However, further research is needed to understand how to best finetune LLMs for specific recommendation tasks, which is our focus in this study.

In this project, we aim to fine-tune Stanford Alpaca Large Language Model on a recommendation task with the open-source Amazon dataset. We processed the public Amazon dataset and use Chat-GPT to create fine-tuning data including three parts as “Instruction”, “Input”, and “Output” with a certain structure. After preparing the fine-tuning data, we aimed at adjusting the Alpaca Model with Alpaca-LoRA framework and generating inferences if we could. Besides, we intend to generate a set of recommendations via a traditional benchmark algorithm to compare the results of the benchmark algorithm and inferences of our fine-tuned LLMs for the same customers. For this comparison, we have chosen the *Surprise* algorithm as traditional benchmark, which is a popular algorithm to generate product recommendations with collaborative filtering.

III. Dataset

A. *Customer Dataset and Input-Instruction Generation*

Our research made use of two distinct datasets: the original Amazon customer dataset collected randomly between 1995-2015 and a derived fine-tuned dataset tailored for Alpaca LLM. The original dataset (former one) is in a tabular format amalgamation of customer and product metadata obtained from Amazon [11]. This dataset comprises features like 'unique id of reviewer, 'name of reviewers', unique product ids, information about features of products, comment text, categories of products such as kitchen and electronics, name of the product, brand, and purchased date. Each row in this data schema shows a unique comment text written by customers.

Figure 1 demonstrates briefly and in an organized manner the end-to-end pipeline of this study. As shown in this figure, to create an instructive and salient dataset ready for fine-tuning from Amazon dataset, a series of comprehensive data cleaning processes were performed. This comprised of various techniques such as trimming lengthy text sequences, removing unnecessary symbols, and normalizing textual attributes like the category or brand of product. The goal of these pre-processing steps was to ensure a cleaner, more streamlined dataset that could efficiently be used to fine-tune Alpaca, fine-tuned version of LLaMA. Fine-tuning such a model requires a json data in which every item contains three components: instruction, input, and output [8]. Thus, after data preprocessing, our resultant fine-tune dataset consists of two main components: 'instruction' and 'input'. The 'instruction' component was created using a set of distinct recommendation instructions (more than ten), such as "Given the products I bought and my comments below, recommend to me what I should buy next and provide a reason for your recommendation." Each of these instructions was randomly assigned to different inputs to diversify the instructive framework of the dataset.

The 'input' component of the dataset was generated on a per-customer basis, employing a prompt-style text format. This format took into account factors like the customer's prior purchases, item categories, item types, and their respective comments. We incorporated this different information for each customer data in the 'input' component. Hence, we formalized 'input' into prompt-like text that put words into customer's mouth. To illustrate, a single 'input' for a customer may appear as follows:

"*I am 'Gordon Boone'. I bought the following products: Wilton Easy Pour Funnel, Frieling Cilio C107005 Egg Cup Set, Blue Q, Sylvania SCR1053 1.2, AmazonBasics 6. Here are some features of these products respectively: Produced with the highest grade materials Manufactured to the highest quality available Manufactured in Hong Kong, Made of fire-proof hard porcelain Formed holder holds salt shaker and spoon Dishwasher safe, Color is Weed Money, Size is 1.2-Inch, Size is 6 Feet Color is White. For one of the products I bought, I wrote this comment: The bigger than normal BLUE LED numerals are easy to see and recognize. I keep one in the bedroom, one in the kitchen, and travel with one. Has am/fm capability. These were wise purchases for me!*"

Our carefully refined fine-tune dataset, which was created from original Amazon dataset, will be used in fine-tuning the Alpaca LLM on recommendation task. Hence, these were the 'instruction' (recommendation instructions) and 'input' (customer data in text format) components of our fine-tune dataset.

B. Output Generation

Following the creation of 'input' and 'instruction' data for each customer in our Amazon dataset, a process of data transformation and recommendation generation was carried out utilizing the ChatGPT-3.5 model as shown in **Figure 1**. Our newly generated dataset (instruction and input) was provided to GPT-3.5 to make intelligent and user-specific inferences. The underlying aim of this process is the generation of personalized recommendations, which are informed by the customer's preferences, prior purchases, and accompanying comments that provide insights about customers' lifestyles, habits and even demographic information.

To generate the 'output' pair, we fed the 'instruction' and 'input' pairs for each customer to ChatGPT -3.5. The model was thereby engaged to infer customer-specific patterns and generate recommendations corresponding to the user profile. This process was conducted for each of the 200 unique customers in our dataset who bought at least five products from different categories. The generated output represents a recommendation from GPT-3.5, formulated after parsing and processing the 'instruction' and 'input' pairs. As an example, one output we get from ChatGPT for a specific user is the following: "*Taking into consideration your praise for the durable Hydro Flask and your apparent love for outdoor activities, the 'Patagonia Women's Better Sweater Fleece Jacket' could be a superb addition to your gear. This product is known for its durability and warmth, making it a great companion for your hikes, similar to your trusted Hydro Flask*"

As a result of this procedure, our fine-tuned dataset was completed with a third necessary component, 'output' that represents the generated recommendation from GPT-3.5. Consequently, our fine-tuned dataset was completed with these three components: 'input', 'instruction', and 'output'.

Finally, following the framework of the Stanford Alpaca fine-tune dataset that was used for training Alpaca, we successfully transformed the instruction-input-output pairs for 200 unique customers into a JSON data format. This format takes each data item as a singular instance embodying the instruction, input, and output for an individual customer.

IV. MODEL

A. Stanford Alpaca

Stanford's Alpaca, an instruction following large language model developed by Facebook's Meta, is a model with 7B and 13B parameters that was fine-tuned from the LLaMA 7B model on 52,000 instruction-following demonstrations [6]. Alpaca was designed as an instruction-following language model, displaying behaviours akin to OpenAI's text-davinci-003, while remaining significantly smaller and cost-effective to reproduce. However, developers of the model note that Alpaca is intended solely for academic research due to its underlying licensing restrictions, and the fact that it lacks adequate safety measures for general deployment. The LLaMA models were then fine-tuned using the Hugging Face training framework, incorporating techniques like Fully Sharded Data Parallel and mixed precision training.

B. Alpaca-LoRA

To understand the principle of our approach that we follow inspired from the community of open LLMs, it's necessary to first delve into the foundational concept of LoRA: Low-Rank Adaptation of Large Language Models.

LoRA is a transformative technique that allows for efficient fine-tuning of large pre-trained language models. Largely based on the principle of low-rank matrix factorization, LoRA separates the parameters of these models into a large pre-trained "base" and a small, task-specific "adapter" [3]. This separation substantially reduces the number of parameters required for the fine-tuning process. The base, or backbone, parameters are frozen during adaptation and contributes the general language understanding ability of the model. The adapter, on the other hand, is a low-rank matrix that is trained for specific tasks. Since the adapter is considerably smaller than the base, it can be rapidly trained on a limited amount of task-specific data. This renders the process much more efficient and economical. Following this, our study extends the above concepts by utilizing the Alpaca-Lora framework [8], a derivative of the original LORA technique.

Table 1: Hyperparameter Configurations

Hyperparameters							
Training Hyperparameters					**LoRA Hyperparameters**		
Data Size	**Batch Size**	**Epoch**	**Learning Rate**	**Val Set Size**	**Dropout**	**Lora_r**	**Alpha**
200	16	16	3,00E-04	30	0.05	8	16
200	8	12	1,00E-04	30	0.05	8	16
2200	64	5	3,00E-04	60	0.05	8	16
200	32	5	3,00E-04	40	0.05	8	16
200	32	10	2,00E-04	32	0.05	8	16

The Alpaca-Lora framework is a GitHub project that reproduces the Stanford Alpaca via LLaMa tokenizer. In Alpaca-Lora, the base model is trained to follow instructions in the input, which it uses to generate responses. The architecture and training procedure of Alpaca-Lora is designed to inherit and build upon the general language understanding ability of large pre-trained language models, and to adapt rapidly to task-specific data.

Our fine-tuning process employs the Alpaca-Lora framework, leveraging its strength in instruction following and rapid adaptation to the nuances of the recommendation task. We'll explore the specifics of this process in the following sections.

V. Results

Table 1 shows different configurations (from 1 to 5) of model hyperparameters with ADAM optimizer as far as our computational resources allowed. Each configuration adjusts data size, batch size, epochs, learning rate, validation set size, dropout, LoRA rate, and alpha to fine-tune the LLM for a recommendation task.

Configuration 1 starts with a data size of 200 and batch size of 16, iterating 16 times over the dataset, and sets the learning rate to a common value of 3e-4. **Configuration 2** has the same data size but halves the batch size and reduces the epochs to 12, with a more conservative learning rate of 1e-4. **Configuration 3** significantly upscales the data size to 2200, with a larger batch size of 64 and fewer epochs (5), maintaining the learning rate of 3e-4. All configurations share dropout of 0.05, and LoRA and alpha rates of 8 and 16, respectively. These adjustments offer different trade-offs between learning stability, speed, and capacity, with various performance outcomes based on the task's specifics and data characteristics.

Due to computational scarcity, we could not tweak a lot with LoRA hyper parameters of lora_r, alpha and dropout. First one (lora_r) is an optional hyperparameter that define the rank of the lora parameters. This implies that the smaller lora_r is , the fewer parameters lora has. The second is there to control the initial scale of the linear transformations in LoRA.

Table 2 shows the results of these different configurations. The results show that **Configuration 3**, which utilized a larger data size, produced the lowest training and validation loss (1.361 and 1.383 respectively), indicating it performed best in modeling the data and generalizing to unseen data. This suggests that the larger dataset allowed the model to capture more complex patterns. In this **Configuration 3**, even though we created a data to be fine-tuned with 200 customers, we also added instruction-following data pairs taken from Alpaca's training data to our dataset to increase. It took several hours on single GPU by Nvidia RTX 3090.

Table 2: Cross-Entropy Loss Results

Fine-Tune Losses (Average)		
	Train Loss	**Val Loss**
Configuration 1	1.344	1.484
Configuration 2	1.505	1.529
Configuration 3	**1.361**	**1.383**
Configuration 4	2.492	2.444
Configuration 5	2.270	2.149

Configuration 1 also exhibited relatively low loss values, demonstrating its effectiveness despite a smaller dataset. Conversely, **Configurations 4 and 5**, despite the same data size as **Configuration 1**, resulted in significantly higher losses, potentially due to the combination of higher batch size and epochs, indicating a possible overfitting issue. **Configuration 2**, with the smallest batch size and reduced epochs, led to a slightly higher loss than **Configuration 1**, which could reflect the less stable gradients generated by smaller batch sizes.

Figure 2, Figure 3, and Figure 4 shows only graphs for **Configuration 1, 2, and 3** with respect to epoch sizes and train-validation losses. All models improve over time as expected. However, there are differences worth noting. For **Configuration 1**, both training and validation losses decrease steadily with each epoch. This suggests that the model is learning effectively from the data and generalizing well to the validation set, with no clear signs of overfitting given the close correlation between the training and validation losses.

Configuration 2 also shows a generally decreasing trend for both losses, but the path is more jagged, with instances where the loss increases from one epoch to the next. This might be due to the smaller batch size leading to less stable gradients during optimization. There's also a noticeable move in validation loss towards the end. This may suggest that the model may start overfitting after epoch 10. Lastly, **Configuration 3** exhibits a steady drop in training loss over the epochs. However, after reaching a minimum, the training loss increases slightly. This might suggest that the model might start overfitting. This is less evident in the validation loss, which continues to decrease, possibly due to the larger data size offering a broader range of examples to learn from, mitigating overfitting to some degree.

Overall, all configurations show improvement over time, but their trajectories and the balance between training and validation losses suggest different degrees of model optimization and generalization.

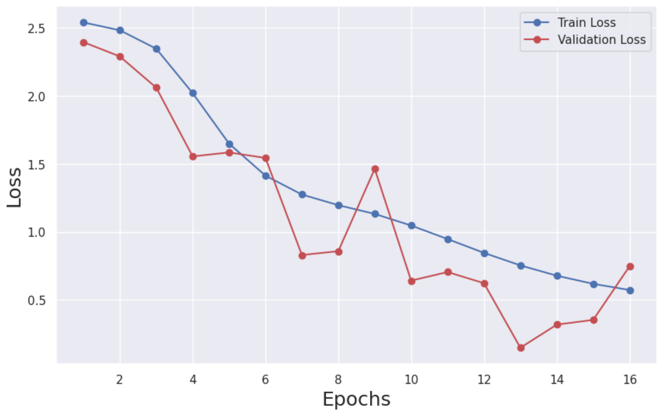


Figure 2: Configuration 1

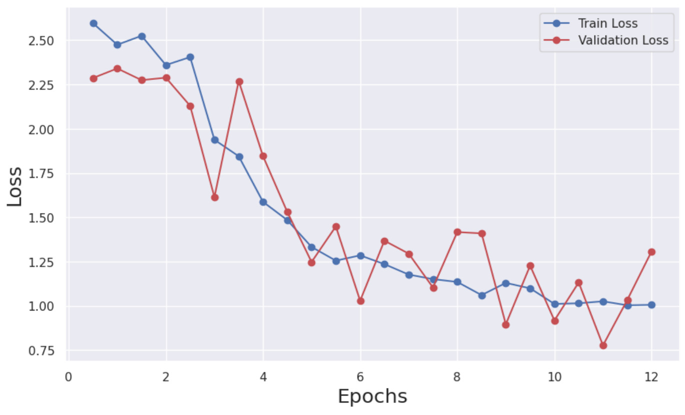


Figure 3: Configuration 2

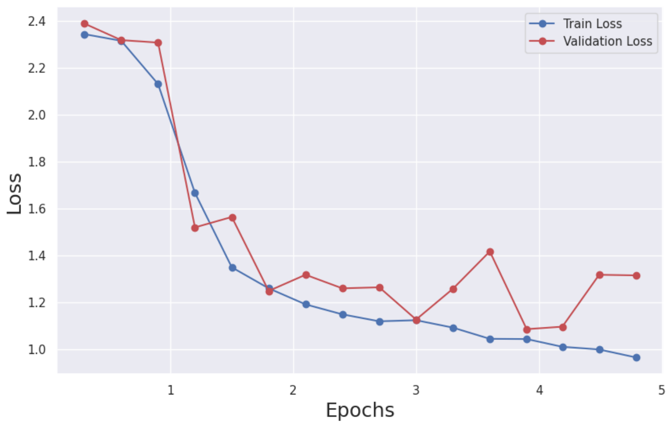


Figure 4: Configuration 3

Testing the performance of a fine-tuned LLM is by convention composed of two parts: fine-tuning process and inference part in which one would expect a model to be an instruction-following model that behaves roughly expected when given input. In the present study, we faced challenges with generating satisfactory results for inference part. As shown above, the fine-tuning process was healthy and the model with different configurations converged successfully. The observed fine-tuning process and reported results initially appeared promising. The model demonstrated decreasing loss values over epochs, which traditionally signals successful learning. However, despite our best efforts to configure and optimize the model during the fine-tuning process, the final output (generation part) did not meet our expectations for meaningful, varied recommendations.

When deployed for the recommendation task, the model demonstrated a tendency to repeat the input (which is customer data) provided as if it got stuck in a loop, rather than generating diverse and novel suggestions. This issue of 'parroting' significantly reduced the utility and quality of the model's recommendations, rendering it useless for now. This behavior also raises questions about the Alpaca's ability to extract and apply complex patterns from the training data, a necessary feature for effective recommendation systems. We suspect that this issue stems from the fact that we had to train the model with a small fine-tune dataset (instruction-input-output) because of availability of GPU to train. Given scarcity of computational resources, we could not manage to make our newly fine-tuned Alpaca yet, an instruction-following model, so it end up being a raw language model.

VI. Conclusion and Future Work

This project demonstrated that fine-tuning Alpaca Large Language Model for recommendation tasks offers promising results but still faces substantial challenges. Experimentation with different configurations showed that larger data sets and careful tuning of batch sizes and learning rates can lead to lower training and validation losses. Despite successful convergence during the fine-tuning process, the model's recommendation performance was suboptimal, characterized by a propensity for input repetition instead of generating diverse, meaningful suggestions. This shortfall might be associated with our limited fine-tuning dataset. Further research is needed to fully realize the potential of the Alpaca LLM for recommendation tasks, particularly focusing on addressing the "parroting" issue and developing strategies to use larger training sets and computational resources.

VII. Acknowledgement

We would like to thank *Upily* for allowing us to use their GPU for fine-tuning our model.

References

[1] Cui, Zeyu, et al. "M6-Rec: Generative Pretrained Language Models are Open-Ended Recommender Systems." arXiv preprint arXiv:2205.08084 (2022).

[2] Gao, Yunfan, et al. "Chat-rec: Towards interactive and explainable llms-augmented recommender system." arXiv preprint arXiv:2303.14524 (2023).

[3] HU, Edward J., et al. Lora: Low-rank adaptation of large language models. arXiv preprint arXiv:2106.09685, 2021.

[4] M. J. Pazzani and D. Billsus, “Content-based recommendation systems,” The Adaptive Web, pp. 325–341. doi:10.1007/978-3-540-72079-9_10

[5] P. Parhi, A. Pal, and M. Aggarwal, “A survey of methods of collaborative filtering techniques,” 2017 International Conference on Inventive Systems and Control (ICISC), 2017. doi:10.1109/icisc.2017.8068603

[6] R. Taori, “Alpaca: A Strong, Replicable Instruction-Following Model,” Stanford CRFM, https://crfm.stanford.edu/2023/03/13/alpaca.html (accessed Jun. 11, 2022).

[7] Rudnicky, A. I., Polifroni, Thayer, E H., and Brennan, R. A. "Interactive problem solving with speech", J. Acoust. Soc. Amer., Vol. 84, 1988, p S213(A)

[8] Tloen, “Tloen/Alpaca-Lora: Instruct-tune llama on Consumer Hardware,” GitHub, https://github.com/tloen/alpaca-lora (accessed Jun. 11, 2023).

[9] Wang, Lei, and Ee-Peng Lim. "Zero-Shot Next-Item Recommendation using Large Pretrained Language Models." arXiv preprint arXiv:2304.03153 (2023).

[10] Zhang, Yuhui, et al. "Language models as recommender systems: Evaluations and limitations." (2021).

[11] He, Ruining, and Julian McAuley. "Ups and downs: Modeling the visual evolution of fashion trends with one-class collaborative filtering." proceedings of the 25th international conference on world wide web. 2016.