Abstract

Our capstone team was tasked with creating a large language model (LLM) based chatbot assistant capable of answering highly technical questions. The focus of this project was Amazon Virtual Private Cloud (VPC). The capstone team collected data throughout the project, including publicly accessible documentation on VPC-related topics, Q&A posted to Amazon Web Services (AWS) forums, and text generated from web links by ChatGPT. We developed a proof of concept for three approaches: retrieval augmented generation (RAG), low-rank adaptation fine tuning (LoRA), and a combination of RAG with the model produced by LoRA.

Introduction

Large language models (LLMs) have demonstrated notable versatility in base form, such as ChatGPT by OpenAI when prompted directly through their web interface. However, in particular niche use cases, LLMs can fail to provide relevant and accurate information. Currently, there are limitations to how much information an LLM could reasonably be expected to "memorize" and difficulties in keeping an LLM updated with newer information. One such niche use case that faces this challenge yet has significant business value would be through the form of support assistants for highly technical products. An LLM-based chatbot that adequately answers highly technical questions could significantly reduce the load on live support agents and simultaneously shorten or effectively eliminate the need for a customer to wait in queue for a response. Our capstone team was tasked by an Amazon Web Services (AWS) team to explore approaches to tackling this use case.

Data: Retrieval, Training, Evaluation

The team did data collection, and the collected data could be grouped to serve three purposes.

The first was for retrieval, where information was collected using the retrieval augmented generation (RAG) approach. This approach requires text data relevant to potential prompts given by a user and will be described in more detail later. Our approach to creating this data was to manually collect over 800 links to AWS web pages containing documentation on VPC, from which the text was extracted and split into short chunks. These chunks were then moved into Pinecone, a vector database, using OpenAI embeddings (as the LLM we selected for this project was ChatGPT). Using a vector database was to make the chunks easily accessible for searching by the RAG approach.

The second purpose was for training, specifically with low-rank adaptation (LoRA) fine-tuning. This fine-tuning approach requires high-quality Q&A pairs, ideally of similar token

lengths. It should make an LLM more suitable for answering questions similar to the training data. We used ChatGPT and another set of links to VPC documentation to obtain this data. We prompted ChatGPT to create Q&A pairs of relatively similar lengths using only information from the links and obtained 331 usable pairs. Before this approach, we also attempted to collect questions and answers from AWS forums manually, but the content's quality and nature were incompatible with our needs.

The third purpose was for evaluation. We used an approach similar to that used when collecting data for training. We produced about 400 new Q&A pairs similar in content to what was included in the retrieval and training data but different enough to be considered “unseen” by the LLM. This ensured that the model was being evaluated on content it reasonably expected to address.

Approach I: Retrieval Augmented Generation (RAG)

Retrieval augmented generation (RAG) is a popular approach to adapting an LLM to a particular topic, industry or use case. It takes a prompt from the user and then uses similarity search to select relevant information (often referred to as context) from a vector database. This context is then passed alongside the original prompt to the LLM, allowing the LLM to answer the question using the information offered by the passed context. This reduces the reliance on the “knowledge” of the LLM, as context from the vector database can be conveniently and continuously updated and then provided to the LLM to generate a response to the prompt.

Our implementation was built using LangChain and used data from over 800 links of VPC documentation. We use cosine similarity to the user prompt when searching the vector database for context and then select the top four chunks of context to pass to the LLM. We also experimented with different prompt templates, the words surrounding the context and question, which collect everything into a single piece of text for the LLM as a prompt. The results of our experimentation will be discussed in a later section.

Approach II: Low Rank Adaptation Fine Tuning (LoRA)

Ensuring efficient resource utilization and cost-effectiveness is crucial when selecting a strategy for fine-tuning. In our exploration, we focused on one of the most popular and effective variants of parameter-efficient methods—Low-Rank Adaptation (LoRA), specifically emphasizing QLoRA, an even more efficient iteration of LoRA. QLoRA, as a fine-tuning method, combines quantization and LoRA techniques.

To implement QLoRA on the VPC dataset, we utilized the PEFT library from Hugging Face. The initial step involved converting the dataset from a CSV format to a .jsonl file. Data tokenization with padding and truncation was employed to facilitate efficient training and reduce memory usage. The LLaMA 2 7B model, utilizing 4-bit quantization from Hugging Face, served as our base model for this process.

QLoRA was applied to all linear layers of the model, including 'q_proj', 'k_proj', 'v_proj', 'o_proj', 'gate_proj', 'up_proj', 'down_proj', and 'lm_head'. Regarding the QLoRA configuration, while the values recommended in the QLoRA paper are r=64 and lora_alpha=16, we opted for r=32 and lora_alpha=64. This decision emphasizes the newly fine-tuned data while concurrently reducing computational complexity.

Post-training, only the QLoRA adapters were saved into the local folder. For inference, we merged the base LLaMA 2 7B model from the Hugging Face Hub with the trained QLoRA adapters. This fine-tuning approach addresses the need for resource utilization and cost-effectiveness efficiency, making it a pragmatic choice for enhancing the performance of large language models in niche use cases such as technical support chatbots.

Approach III: RAG + LoRA

This approach utilizes the same logic and pipeline as the first RAG approach. However, instead of using the OpenAI model, it uses the model produced from the LoRA fine-tuning approach. The model we produced with LoRA can be run and hosted on local servers and was modified to be callable like the OpenAI API. A potential benefit of this approach is that it captures the unique advantages offered by each of RAG and LoRA while being locally deployable without relying on an externally hosted LLM. This is advantageous for users who are concerned about privacy and want to avoid complications associated with reliance on third parties, such as updates to APIs and changes in pricing.

Evaluation Methodology

Our evaluation is conducted on the 48 Q&A pairs, and our metrics revolve around two concepts: response coherence and relevance. For response coherence, we picked **BLEU** (Bilingual Evaluation Understudy), which measures the similarity between references and predictions based on n-grams (contiguous sequences of n words), initially intended for machine translation, and **ROGUE** (Recall oriented understudy for gisting evaluation), which compare the automatically produced text

against human-produced references, initially intended for evaluation on summarization systems. For relevance, we picked **BERTScore,** which helps leverage the pre-trained contextual embeddings from BERT and matches words in candidate and reference sentences by cosine similarity. It also gives us the precision, recall, and F1-Score based on the embeddings. We evaluate the three approaches as mentioned earlier, both at the answer and model levels - by taking a simple average of all the answer-level metrics. Note that all these metrics merely give us an idea of how "close" the answer is to that of a human, so that the next step would be seeking human expert feedback on sensibleness/specificity score or maybe even applying **RLHF** (reinforcement learning from human feedback).

Results + Next Steps

Based on the results of the final table, our RAG approach is the best across all metrics, especially for BLEU and ROGUE. We hypothesize that it's not an apple-to-apple comparison to start with. For RAG, we used the entire set of 800 links on VPC documentation with GPT-3.5 Turbo, the once most capable of the GPT family before GPT-4 came out. For LoRA, due to computational constraints, we generated one Q&A pair of each documentation link (even though our codebase can generate multiple) and used LLama 2 with 7B parameters (smallest) to train on 230 Q&A pairs and validate on 100. Our literature review suggests we need a couple of thousand Q&A pairs. The under-trained model also carried over to our RAG + LoRA.

Our proof of concept shows promise in applying LLMs to answering highly technical questions. Successful implementations could enhance user experience, reducing headaches with troubleshooting issues and waiting for customer support availability. It will also reduce the burden on those deploying these solutions regarding resource allocation among customer support agents and call centers. We expect further experimentation to be fruitful and recommend our client continue pursuing an LLM-based solution to chat support.