Query Tool for 5G Protocol Understanding

- Fine-Tuning on OpenAI's ChatGPT Language Model-

Second Project Checkpoint

Tommy Taing, Ivy Huang, Krisha Adhikari

Group 6

UMass Amherst

Electrical and Computer Engineering

0.1 Project Introduction

0.1.1 Project Background

Every year, organizations from around the world come together to discuss standards and protocols for telecommunication. The foundation, 3GPP, creates numerous amounts of documents that are constantly updated with new terms, acronyms, and meanings for different aspects of mobile telecommunication. However, these changes are almost happening 'behind the scenes', making it hard for a student, professor, or the general public to understand what is happening. Our project attempts to bridge this gap by using large learning models, such an OpenAI's ChatGPT.

0.1.2 Project Goals

Our main goal is to create a specialized ChatGPT API that can answer 5G Protocols (made by 3GPP) in a way that is understandable to any inquirer. We attempt to do this using three methods, each method having its own flaws.

0.2 Project Methods

0.2.1 AssistantAPI

This method of creating a 5G Protocol-specialized ChatGPT API was the easiest, but one of the more limiting solutions. AssistantAPI works using the base ChatGPT model, with the option to input a document for the model to refer to. The AssistantAPI is currently available on OpenAI's site, with GUI to easily help build an AssistantAPI. The results can be seen below.

Query	ChatGPT Base	AssistantAPI
What does DS-TT mean?	DS-TT stands for "Data Science Techniques and Tools". This refers to the methods and software used in the field of data science for analyzing and interpreting complex data.	DS-TT stands for "Device-Side Time-Sensitive Networking Translator." It is a protocol aspect defined in 3GPP TS 24.535 for the translation between Device-Side TSN Translator (DS-TT) and Network-Side TSN Translator (NW-TT) in the context of Time-Sensitive Networking (TSN) in 5G systems[1].

Figure 1: Base ChatGPT model response vs AssistantAPI

While the answer from AssistantAPI is far more concise and correct compared to the base ChatGPT, the limitation with AssistantAPI is that it can only handle up to 20 documents. For smaller projects, this would be okay, but our project is centered around teaching ChatGPT the many different protocols made by the 3GPP, which requires far more than 20 documents.

0.2.2 Embeddings

Our second method to solving this problem was using embeddings. For this method, we took two different documents and turned them into embeddings using OpenAI's embeddings model. With embeddings, there is no limitation to the number of documents that are used, but there are limitations regarding how much text the model can handle. Therefore, we cannot give the model the whole document and expect a quick response. Instead, we have to break it down into sections. The way we did this was utilizing 3GPP's consistent protocol document formatting. While going through the documents, we realized that each protocol document had a very specific sectioning method. Anything that was a title/subtitle could be size 16, 14, or 12 pt. font, depending on the heirarchy of the title. Meanwhile, the actual content text was always size 10 pt. font. We used this formatting to break down the protocol document into sections. This main code can be seen below.

```python
def section_doc(doc):
    section_titles = []
    section_texts = []
    current_section_text = ""
    title_font_sizes = {16, 14, 12}
    section_font_size = 10

    for paragraph in doc.paragraphs:
        paragraph_font_size = get_paragraph_font_size(paragraph)

        if paragraph_font_size in title_font_sizes:
            if current_section_text:
                section_texts.append(current_section_text.strip())
                current_section_text = ""
            section_titles.append(paragraph.text.strip())
        else:
            current_section_text += " " + paragraph.text.strip()

    if current_section_text:
        section_texts.append(current_section_text.strip())

    # Return list of tuples with titles and their corresponding section text
    return list(zip(section_titles, section_texts))
```

Once the sectioning was complete, we turned each section into embeddings using the embeddings model. From there, we used a guide provided by OpenAI to create the ChatGPT API [1]. The way embeddings works with ChatGPT is that when a question is asked, a search algorithm returns the sections with the most relatedness to the query. So, if the query had something with 'homebase station', then the sections containing this term would appear in the top 100 results. This code was provided by OpenAI and can be seen below.

```python
def strings_ranked_by_relatedness(
    query: str,
    df: pd.DataFrame,
    relatedness_fn=lambda x, y: 1 - spatial.distance.cosine(x, y),
    top_n: int = 100
) -> tuple[list[str], list[float]]:
    """Returns a list of strings and relatednesses, sorted from most related to
       least."""
    query_embedding_response = client.embeddings.create(
        model=EMBEDDING_MODEL,
        input=query,
    )
    query_embedding = query_embedding_response.data[0].embedding
    strings_and_relatednesses = [
        (row["text"], relatedness_fn(query_embedding, row["embedding"]))
        for i, row in df.iterrows()
    ]
    strings_and_relatednesses.sort(key=lambda x: x[1], reverse=True)
    strings, relatednesses = zip(*strings_and_relatednesses)
    return strings[:top_n], relatednesses[:top_n]
```

The results from using base ChatGPT vs Embeddings can be seen below.

Query	ChatGPT Base	Embeddings
What does DS-TT mean?	"DS-TT" could potentially refer to a variety of things depending on the context. Without more information, it is difficult to pinpoint the exact meaning. Could you provide more context or details to help me understand the context in which you encountered "DS-TT"?	DS-TT stands for Device-Side TSN Translator. It is used to interface with a TSN network while achieving transparency in a 5G network.

Figure 2: Base ChatGPT model response vs Embeddings

While the results from embeddings are better than the base model, we want the users to be creative with their queries. This means creating a model that is able to cross reference multiple documents for a 'bigger picture' answers, rather than one that answers solely based off references. Additionally, search algorithms are slow and this is even more so when working with multiple documents. To avoid this setback from embeddings and to enable the 'creativeness' of the users, we instead turn to fine-tuning.

0.2.3 Fine-Tuning

OpenAI provides a GUI with a built-in fine-tuning models which makes it very easy to implement fine-tuning on our project. Embeddings is used to provide the model with new information that can be used to answer questions. Fine-tuning teaches the model how to answer a question like the structure or format of it. It's also better for long-term memory, so it can handle the creative and complicated questions. This makes fine-tuning more effective than embeddings in this scenario. In our project, we used fine-tuning on the JSONL files. There are many advantages to fine-tuning like leveraged learning, faster training, and domain adaptation.

OpenAI offers a GUI to create fine-tuned models, making the process straightforward and accessible. This GUI allows us to upload data, configure a model, and initiate the fine-tuning process through an intuitive and user-friendly platform. First we will discuss the advantages and disadvantages of fine-tuning.

Fine-Tuning Advantages

1. Data Preparation: Creating a fine-tuning dataset requires writing multiple conversations, which can be time-consuming and labor-intensive.

2. Enhanced Memory: The model gains a better understanding of context, making it capable of handling more complex and creative questions.

3. Improved Performance: Models that are fine-tuned on domain-specific data often perform better in that domain compared to general-purpose models.

Fine-Tuning Disadvantages

1. Data Preparation: Creating a fine-tuning dataset requires writing multiple conversations, which can be time-consuming and labor-intensive.

2. Consistency: Both sides of the conversation need to be crafted carefully to ensure consistency and relevance, which can be challenging.

3. Overfitting Risk: There is a risk of the model overfitting to the fine-tuning data, making it less effective at general tasks outside the fine-tuned domain.

QUESTION	ChatGPT Base model	ChatGPT Fine-tuned
What does DS-TT mean?	"DS-TT" could potentially refer to a variety of things depending on the context. Without more information, it is difficult to pinpoint the exact meaning. Could you provide more context or details to help me understand the context in which you encountered "DS-TT"?	DS-TT stands for "Data Science Techniques and Tools". This refers to the methods and software used in the field of data science for analyzing and interpreting complex data.

Figure 3: Base ChatGPT model response vs Fine-Tuning

In summary, OpenAI's fine-tuning GUI simplifies the process of customizing models for specific tasks. While fine-tuning enhances the model's long-term memory and ability to generate creative responses, it requires significant effort in data preparation and careful crafting of training examples. Despite these challenges, the benefits of having a highly specialized and responsive model often outweigh the drawbacks.

In this comparison, the fine-tuned model was not at all accurate in its response to the query about "DS-TT". It provided an incorrect and overly specific answer without considering the need for additional context or the potential for multiple meanings.

On the other hand, the base model showcased a better approach by acknowledging the ambiguity of the term and requesting more information before attempting to provide an answer. This approach is beneficial in scenarios where terms may have multiple interpretations or where the initial query lacks sufficient context.

Reproducing Steps for Fine-Tuning

To be able to have similar results, there needs to be a path and the same steps for fine-tuning, there needs to be clear information feeding into the fine-tuning GUI. Here are our methodologies and steps we've taken to achieve fine-tuning.

1. First we have to setup the prompt layout that OpenAI API's uses for their fine-tuning model, the format is a JSONL format. In the following is an example of the required and specific way of input:

```
{"messages": [{"role": "system", "content": "Your assistant provides
    accurate responses to technical queries about 5G protocols."},
    {"role": "user", "content": "What's the purpose of Ethernet port
    management in a 5G system?"}, {"role": "assistant", "content":
    "Ethernet port management in a 5G system helps manage the
    communication between Time-Sensitive Networking (TSN) Application
    Function (TSN AF) and the Device-Side TSN Translator (DS-TT) or
    Network-Side TSN Translator (NW-TT). It supports procedures for
    Ethernet port configuration, monitoring, and updates."}]}
```

   Using this specific format for fine-tuning ensures that the data fed into the model is well-structured, easy to process, and scalable. This structured approach is crucial for training effective conversational models, as it enables the model to learn from clear, consistent, and contextually rich data.

   The specific structure with "messages", "role", "system", "user", and "content" fields in the JSON Lines (JSONL) format is designed to create a clear, organized, and contextually rich dataset for fine-tuning conversational models

2. Structure: It groups together all the exchanges that constitute a single dialogue or conversation.

3. Messages Purpose: This field contains an array of message objects, each representing a part of the conversation.

4. Role Purpose: This field indicates the role of the entity involved in the conversation. It helps the model differentiate between different participants in the dialogue.

5. Common Values: system: Sets up initial instructions or context for the conversation. This role is usually used to define the behavior or constraints of the assistant.

   user: Represents the input or query from the end-user. This helps the model understand what kind of question or prompt is being posed.

   assistant: Denotes the model's response. This helps the model learn what kind of answers to provide based on the user's input.

6. System Purpose: Provides initial setup or contextual instructions to the model. This helps in guiding the model's behavior for the conversation. Example: "Your assistant provides accurate responses to technical queries about 5G protocols." This sets the context for the assistant's responses.

7. Content Purpose: Contains the actual text of the message from each role. This is the primary data the model uses to learn from and generate responses. Example: The user's question or the assistant's reply.

We create 28 of these conversations and add them into the JSONL file. After feeding the JSONL file into OpenAI's finetuning interface, the process continues through check and verification stages listed in the following:

1. Data Upload and Validation: Ensure the file is correctly formatted and error-free.

2. Preprocessing: Tokenize and clean the data for training

3. Model Configuration: Select the base model and configure hyperparameters

4. Fine-Tuning Process: Train the model using the provided data, iterating over multiple epochs.

5. Monitoring and Logging: Track progress and maintain logs for analysis

6. Evaluation and Validation: Validate the model's performance with metrics and a separate dataset

7. Model Deployment: Save and prepare the final model for deployment and integration.

8. Testing and Feedback: Test the model extensively and refine it based on feedback.

Steps 1-7 listed above are not performed by us, but rather OpenAI's platform. Thus, with finetuning, the primarily goal is to provide conversations until the model can perform accurately.

0.3 Project Improvements

To enhance the performance and accuracy of our fine-tuned model, we explored additional strategies beyond standard fine-tuning. These strategies involve leveraging embeddings and generating query-answer pairs using the base model. By combining these approaches, we aim to create a more robust training dataset and improve the overall quality of the model's responses.

For example, here are some ways we can enhance Embeddings and Fine-Tuning if combined

1. Embeddings for Query Responses: We utilize embeddings to create queries and generate corresponding responses. These embeddings, which have referenced responses, serve as a valuable resource for constructing high-quality conversation pairs.

2. Integration into Fine-Tuning: The responses generated through embeddings are incorporated into the JSONL file used for fine-tuning. This ensures that the model is trained on well-referenced and contextually relevant data.

0.3.1 The Process for Embeddings with Fine-Tuning

1. Generate Queries: Create a set of relevant queries based on the domain or task.

2. Generate Responses Using Embeddings: Use embeddings to find the most relevant responses for these queries. Embeddings capture the semantic meaning of text, ensuring that the responses are contextually accurate.

3. Incorporate into JSONL: Structure the queries and responses in the JSONL format, maintaining the "messages", "role", and "content" fields.

By combining embeddings and fine-tuning or generating query-answer pairs using the base model, we create a comprehensive and high-quality training dataset. This hybrid approach leverages the strengths of both embeddings and the base model to enhance the fine-tuning process, resulting in a more robust and reliable model.