Summer 2024 Olympics Question Answering (QA) Chatbot

1st Carrie Aponte
Kansas State University

I. Introduction

This project focuses on creating a reliable, domain-specific question-answering (QA) chatbot trained on data from the Summer 2024 Olympics. Given the fact that large language models (LLMs) typically do not have access to recent data such as results from sporting events, this fine-tuning task could show to be beneficial and helpful. I seek to maximize precision in response generation and ensure that the answers are grounded in the data.

My approach involves fine-tuning multiple LLMs to identify the most effective model for QA tasks, specifically, Summer 2024 Olympics related QA tasks. I trained and evaluated each model using QA metrics, then concentrated on optimizing the best-performing model. The data I obtained for the Summer 2024 Olympics was not in QA format, and instead was in regular spreadsheet format. It also was not cleaned or pre-processed, and had many missing values in fields. I developed three main scripts for converting the data to a suitable format for training a QA chatbot, which included cleaning, choosing relevant columns, and generating multiple relevant questions for each field of interest.

My null hypothesis is that the fine-tuned models will not perform better than baseline, non-fine-tuned models. My alternative hypothesis is that fine-tuning my models will yield improved performances over the baseline model.

A. Background and Related Work

The development of QA systems has significantly advanced with the advent of language models like BERT, GPT, and their successors. These models have been fine-tuned for various domain-specific applications. Fine-tuning LLMs is an effective method to refine a model and improve its ability to answer topic-specific information. Fine-tuning is especially helpful when a model is trained on recent, topical material, such as the Summer 2024 Olympics. This is because many language models are not trained on the most recent events. In fact, a query to ChatGPT on its most recent knowledge will show that it is currently trained on information up to October 2023. It is currently December 2024, and this discrepancy shows that relying on language models that have not been fine tuned to handle recent events or information will not have accurate responses.

Lomshakov et al. (2023) [1] explored fine-tuning LLMs for answering programming-related questions, emphasizing the importance of domain-specific training data to enhance model performance. Their findings underscore the necessity of adapting LLMs to specific knowledge areas to improve accuracy and relevance in responses.

Zhang et al. (2024) [2] investigated methods to enhance LLM performance in QA tasks by fine-tuning models with domain-specific data. They demonstrated that incorporating targeted datasets during training significantly improves the accuracy and reliability of the generated answers.

Ye et al. (2024) [3] provided empirical insights into fine-tuning strategies for LLMs in QA applications. Their research offers valuable guidance on optimizing training processes to achieve better performance in domain-specific QA systems.

My project builds upon these studies by focusing on fine-tuning LLMs specifically for 2024 Olympics QA tasks. I utilized domain-specific Olympics datasets, the Stanford Question Answering Dataset (SQuAD), and employed fine-tuning techniques. I aimed to develop a chatbot capable of delivering accurate and reliable information in response to user queries about the 2024 Olympics. I looked into using multiple models, including Llama, BERT variants, gpt variants, and other singular models like Gorilla. My model selection was mostly due to interest in the gpt models, desire to gain experience with transformer models like BERT, and selected FLAN-T5 because it's a model that has been optimized for instruction-based tasks.

B. Problem Statement

My primary challenge is to develop a QA chatbot capable of generating domain-specific, fact-based answers to Summer 2024 Olympics-related queries, based on my dataset. I focused on fine-tuning pre-existing language models on my domain-specific dataset, to supplement their existing knowledge with targeted information related to the Summer 2024 Olympics. This fine-tuning is a critical step in adapting language models, specifically chatbots, as they need to be able to generate accurate, up-to-date, contextually relevant information in response to user queries. My project addresses this issue by fine-tuning the models on a comprehensive dataset with information on the Summer 2024 Olympics, ensuring that model responses are fact-based and grounded in relevant data.

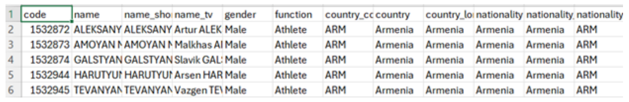
II. Methodology

My methodology involves evaluating and fine-tuning multiple language models, including transformers and LLMs to identify the model best suited for this QA task. I chose this thorough approach to allow me to systematically compare models and techniques and optimize the chatbot's reliability and accuracy in answering Olympics-related queries.

A. Dataset

The main dataset used in this project was obtained from Kaggle: https://www.kaggle.com/datasets/piterfm/paris-2024-olympic-summer-games [4]. The Kaggle author obtained the information from both the Summer 2024 Olympics official website [5], and the Wikipedia page for the 2024 Olympics [6]. The data was originally a zip file, which consisted of a large number of csv files containing information regarding the 2024 Summer Olympics. A snapshot of the original data set-up can be seen in Figure 2, which shows the beginning of the Athletes.csv file. There were 58 csv files in all. 13 of these files contained general Olympics information such as athletes, coaches, venues, medalists, and schedules. These 13 files were deemed to be the most informative. The other 45 csv files consisted of results for all the various sports, such as archery, badminton, and boxing. I visually inspected the files to become familiar with the format and the information contained in them. I then loaded all the data into Google Drive so that I could mount it to Google Colab and write preprocessing scripts. It is important to note that this dataset was not in the format required for training a QA file originally, and I transformed it into an appropriate QA dataset, as discussed below.

I also used the SQuAD dataset, which contains over 100,000 general QA pairs [7]. I used this dataset to train and evaluate my gpt2-large model, to gain an understanding of the model's effectiveness on the QA task. I chose to fine-tune gpt2-large on the entire SQuAD dataset to determine whether pre-training the model with a large, general QA dataset like SQuAD would improve performance in my domain-specific Summer 2024 Olympics QA task.



1	code	name	name_sho	name_tv	gender	function	country_cc	country	country_lo	nationality	nationality	nationality
2	1532872	ALEKSANY	ALEKSANY	Artur ALEK	Male	Athlete	ARM	Armenia	Armenia	Armenia	Armenia	ARM
3	1532873	AMOYAN N	AMOYAN N	Malkhas Al	Male	Athlete	ARM	Armenia	Armenia	Armenia	Armenia	ARM
4	1532874	GALSTYAN	GALSTYAN	Slavik GAL:	Male	Athlete	ARM	Armenia	Armenia	Armenia	Armenia	ARM
5	1532944	HARUTYUI	HARUTYUI	Arsen HAR	Male	Athlete	ARM	Armenia	Armenia	Armenia	Armenia	ARM
6	1532945	TEVANYAN	TEVANYAN	Vazgen TE\	Male	Athlete	ARM	Armenia	Armenia	Armenia	Armenia	ARM

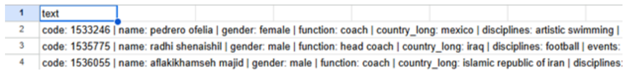
Fig. 1. The original dataset format - Athletes.csv

B. Data Processing

I chose to remove two of the original 13 general files: preliminary schedules, and torch route. I removed preliminary schedules as it was redundant and less accurate and informative than the schedules.csv file. I chose to remove the torch route as I deemed the information unimportant for the purposes of training this QA model. I then inspected the columns in each of the remaining 11 general files, and left out irrelevant or redundant columns in my preprocessing scripts. The 45 result datasets were all similar to each other, and I chose to retain all the fields, deeming them relevant. While some result columns had only minimal information or were blank for many of the entries, I considered the information important when it was included, and therefore chose to handle the missing fields rather than remove those columns.

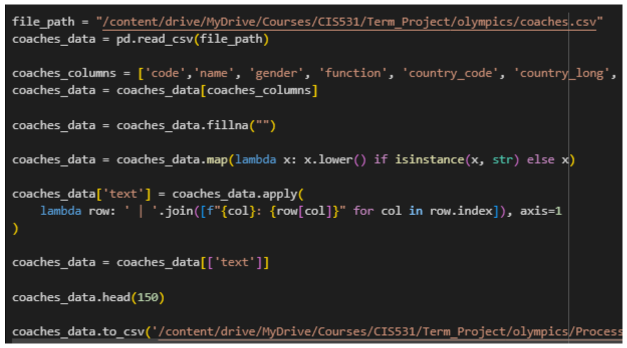
I first converted all the text to lowercase for data standardization and to ensure consistent text handling. I replaced each missing data point with an empty string ”” to avoid errors such as NaN. I chose which columns to retain, and which to omit in my processing, as they were repetitive or unnecessary.

To make the data format suitable for training a language model, I concatenated the columns for each dataset into single text fields, in a string format. The resulting structure had each field separated by column name and answer with a colon, and each column separated by a bar, resulting in the following structure: ”name: aleksanyan artur — gender: male”. The script for this pre-processing can be seen in Figure 3, and the pre-processed data in this format can be seen in **??**. I initially thought this to be the best formatting for training a language model, but later realized I had not formatted the data in an appropriate manner to train a model on a QA task, and therefore needed to apply additional preprocessing to each file.



1	text
2	code: 1533246 | name: pedrero ofelia | gender: female | function: coach | country_long: mexico | disciplines: artistic swimming |
3	code: 1535775 | name: radhi shenaishil | gender: male | function: head coach | country_long: iraq | disciplines: football | events:
4	code: 1536055 | name: aflakikhamseh majid | gender: male | function: coach | country_long: islamic republic of iran | disciplines:

Fig. 2. Dataset format after first processing - Coaches.csv



```
file_path = "/content/drive/MyDrive/Courses/CIS531/Term_Project/olympics/coaches.csv"
coaches_data = pd.read_csv(file_path)

coaches_columns = ['code','name', 'gender', 'function', 'country_code', 'country_long',
coaches_data = coaches_data[coaches_columns]

coaches_data = coaches_data.fillna("")

coaches_data = coaches_data.map(lambda x: x.lower() if isinstance(x, str) else x)

coaches_data['text'] = coaches_data.apply(
    lambda row: ' | '.join([f"{col}: {row[col]}" for col in row.index]), axis=1
)

coaches_data = coaches_data[['text']]

coaches_data.head(150)

coaches_data.to_csv('/content/drive/MyDrive/Courses/CIS531/Term_Project/olympics/Process
```

Fig. 3. First Data Pre-processing Script - Coaches.csv

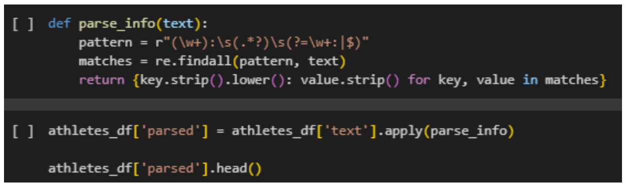
The additional preprocessing was more involved, as it required developing questions for each field, and noting their answers. I chose to format the data in the standard set by the SQuAD: as context, question, and answer. The strings I had previously developed became the context, and I used ChatGPT to help generate questions based on each individual dataset. Here is a sample of the queries I entered to ChatGPT to obtain the questions I used for developing my QA script. The following example is for my athletes dataset:

> ”I am training a question-answering chatbot on Olympics data. My data is currently not in the correct format for a QA task, so I need to transform it to the format: context question answer. Please help me generate questions for my data about olympic athletes. Here are examples of the contents of the data: code: 1532872 — name: aleksanyan artur — nametv: artur aleksanyan — gender: male — function: athlete — country: armenia — nationality: armenia — height: 0 — weight: 0.0 — disciplines:

['wrestling'] — events: ["men's greco-roman 97kg"] — birthdate: 1991-10-21 — birthplace: gyumri — birthcountry: armenia — residenceplace: gyumri — residencecountry: armenia — nickname: white bear — hobbies: playing and watching football — education: graduated from shirak state university (gyumri, arm) — family: father, gevorg aleksanyan — lang: armenian, english, russian — coach: gevorg aleksanyan (arm), father — reason: he followed his father and his uncle into the sport — hero: footballer zinedine zidane (fra), world cup winner (1998) and european champion (2000) with france, won the champions league as a player and three times as a manager with real madrid, three-time fifa world player of the year — influence: his father, gevorg aleksanyan — philosophy: "wrestling is my life." (mediamax.am. 18 may 2016) — sportingrelatives: — ritual: — othersports: "

In response to these queries, ChatGPT would provide a variety of questions concerning the specific dataset. For example, "What events does this athlete compete in?" or "What is the nickname of this athlete?". I manually investigated the generated questions, which required modifications for clarity and flow. I had to apply additional data modification for the answers, as some of the columns I would've used for answers were not clear and informative. For example, in team events, where the 'name' field may consist of a country name rather than the name of an individual athlete. Additionally, some fields such as "event" would contain the answer "mens", rather than detailing which men's event it was. In these cases, I visually inspected the data and chose to concatenate various columns together to develop fewer, but more informative fields.

This script required parsing the previously created strings containing the information for each field to access the individual columns again. This script can be seen in Figure 4. I then applied the data transformation by creating a dictionary containing QA entries with context, question, and answer. The questions came from modified versions of the responses from ChatGPT to my query. A sample of this question generation can be seen in Figure 5.



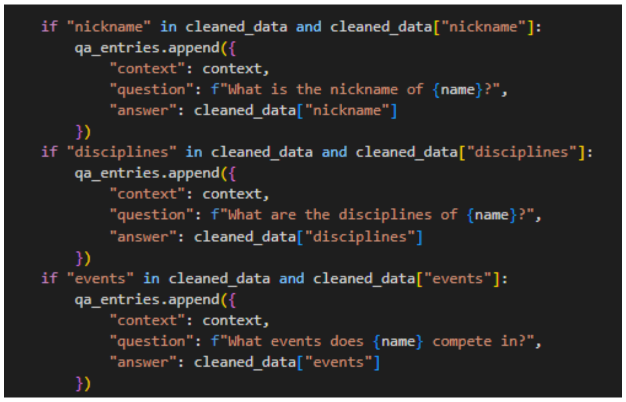
```
[ ]  def parse_info(text):
         pattern = r"(\w+):\s(.*?)\s(?=\w+:|$)"
         matches = re.findall(pattern, text)
         return {key.strip().lower(): value.strip() for key, value in matches}

[ ]  athletes_df['parsed'] = athletes_df['text'].apply(parse_info)

     athletes_df['parsed'].head()
```

Fig. 4. Data Parsing for QA Script

I also chose to modify the date/time fields, which were originally formatted like this: 2024-07-24T16:30:00+02:00. I chose to split the format, ending with a more readable: 2024-07-24 at 16:30:00+02:00. While the 11 general dataset files required manually generating and modifying questions and answers for the model to train on, the 45 result datasets were



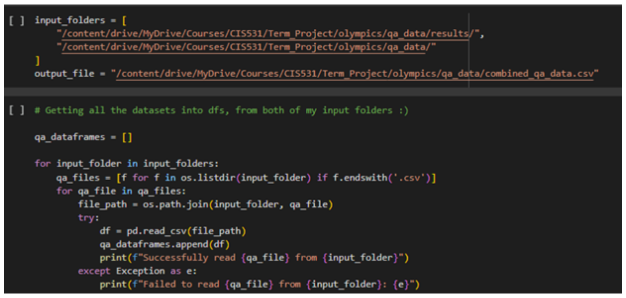
```
if "nickname" in cleaned_data and cleaned_data["nickname"]:
    qa_entries.append({
        "context": context,
        "question": f"What is the nickname of {name}?",
        "answer": cleaned_data["nickname"]
    })
if "disciplines" in cleaned_data and cleaned_data["disciplines"]:
    qa_entries.append({
        "context": context,
        "question": f"What are the disciplines of {name}?",
        "answer": cleaned_data["disciplines"]
    })
if "events" in cleaned_data and cleaned_data["events"]:
    qa_entries.append({
        "context": context,
        "question": f"What events does {name} compete in?",
        "answer": cleaned_data["events"]
    })
```

Fig. 5. Question Generation in QA Script - Athletes.csv

much more similar to each other. I processed around 5 of the result files individually, then once I had a good handle on the differences in the datasets between various sports and the data set-up for team vs. individual sports, I was able to develop a robust loop to handle the variances in formatting (mainly between individual and team sports), and convert the result files into the correct QA format and create good questions. I still manually inspected each resulting file, as it was essential that the results made sense and were in a consistent format. I needed to add more robust handling for potential missing columns into my loop, but this was not a difficult task, and I found the development of the loop to be well worth the time saved in manually creating each script for processing.

The last step in my data preprocessing was to combine all the individual QA datasets into one large QA dataset. For this, I created a list of all the separate QA csv files I'd saved, then looped through the files and combined them with Panda's concat() [8]. The script for this can be seen in Figure 6, and the resulting finalized dataset can be seen in Figure 7. I decided this was possible and necessary because it was essential that my model had enough data to train on, and if I attempted training with one smaller dataset at a time, this would not be sufficient data and would likely lead to overfitting. Additionally, as I had developed and designed the QA format of the datasets, I knew that there would be no issue in combining all the sets into one. The resulting QA dataset had nearly 300,000 QA pairs, which was an excellent, very large size for fine-tuning language models. The dataset was even larger than the generic SQuAD dataset!

For training, I chose to split the dataset into training and validation sets, with 80% used for training and 20% for validation. This split ensured the model had sufficient data for training while preserving a meaningful validation subset for evaluation. Tokenization played a critical role in preparing the data. Contexts, questions, and answers were combined into a single structured input prompt to make the task clear for the model. For example, prompts were formatted as: "context ###Questions: question ###Response: answer ". The tokenizer was customized to handle padding efficiently by using the end-of-sequence token (eos) as a padding token.



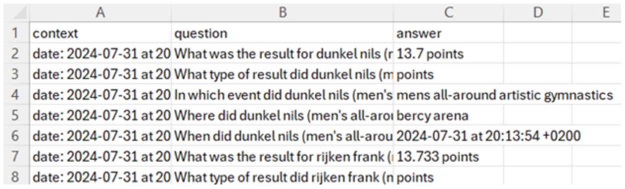
```
[ ] input_folders = [
        "/content/drive/MyDrive/Courses/CIS531/Term_Project/olympics/qa_data/results/",
        "/content/drive/MyDrive/Courses/CIS531/Term_Project/olympics/qa_data/"
    ]
    output_file = "/content/drive/MyDrive/Courses/CIS531/Term_Project/olympics/qa_data/combined_qa_data.csv"
```

```
[ ] # Getting all the datasets into dfs, from both of my input folders :)

    qa_dataframes = []

    for input_folder in input_folders:
        qa_files = [f for f in os.listdir(input_folder) if f.endswith('.csv')]
        for qa_file in qa_files:
            file_path = os.path.join(input_folder, qa_file)
            try:
                df = pd.read_csv(file_path)
                qa_dataframes.append(df)
                print(f"Successfully read {qa_file} from {input_folder}")
            except Exception as e:
                print(f"Failed to read {qa_file} from {input_folder}: {e}")
```

Fig. 6. Script for combining QA Datasets



	A	B	C	D	E
1	context	question	answer		
2	date: 2024-07-31 at 20	What was the result for dunkel nils (r	13.7 points		
3	date: 2024-07-31 at 20	What type of result did dunkel nils (m	points		
4	date: 2024-07-31 at 20	In which event did dunkel nils (men's	mens all-around artistic gymnastics		
5	date: 2024-07-31 at 20	Where did dunkel nils (men's all-arou	bercy arena		
6	date: 2024-07-31 at 20	When did dunkel nils (men's all-arou	2024-07-31 at 20:13:54 +0200		
7	date: 2024-07-31 at 20	What was the result for rijken frank (i	13.733 points		
8	date: 2024-07-31 at 20	What type of result did rijken frank (n	points		

Fig. 7. The Final QA Dataset

C. Model

I experimented with various models: GPT models, FLAN-T5, and BERT variants. I used two different GPT-2 model sizes, gpt2-large and gpt2-xl. I trained gpt2-large on the SQuAD and Olympics datasets, and gpt2-xl only on my Olympics dataset. Initially, I used the SQuAD dataset to test and evaluate the models' ability to handle the QA task, using Beocat to train gpt2-large on the entire SQuAD dataset. This was not a small task, and took a large amount of time and resources. However, I unfortunately found that pre-training the model on the SQuAD dataset did not significantly improve performance over solely training on the Olympics data, therefore I did not train any other models on the SQuAD dataset.

I then chose to train FLAN-T5 on my Olympics dataset, to determine whether alternate LLMs would perform better than the gpt2 models. Lastly, I trained the BERT variants: BERT, RoBERTa, and ALBERT on my Olympics dataset. My plan was to fine-tune and evaluate all of these models to determine which performs the best. I also decided to evaluate a non-fine-tuned gpt2-large model on my Olympics dataset as a baseline. The code for fine-tuning the BERT variants was adapted from that of a fellow student in one of my courses. While my final code was heavily modified compared to the original code, in order to fit my data and needs, it's important to note that the original work was completed by a different student.

I incorporated the Parameter-Efficient Fine-Tuning (PEFT) technique [9] called Low-Rank Adaptation (LoRA) [10] to my models, in order to optimize the fine-tuning process and allow for less memory consumption during the fine-tuning tasks. I also chose to apply 4-bit weight quantization to reduce memory and computational costs. These configurations significantly reduced the memory footprint of the training process while maintaining the model's ability to generate accurate answers. I applied PEFT using the Hugging Face Libraries [11]. In general, I used the Hugging Face Libraries to access many models, to train them, and to evaluate them.

I used the TRL (Transformers Reinforcement Learning) library [12] to streamline the supervised fine-tuning. TRL provided tools such as the SFTTrainer, a specialized trainer for supervised fine-tuning tasks. This trainer was essential for adapting GPT-2 Large to the QA task while maintaining compatibility with LoRA and quantization configurations. The SFTTrainer seamlessly handled tokenized inputs, LoRA-specific adjustments, and the overall optimization process. Its integration ensured that fine-tuning adhered to the constraints of low memory usage and high computational efficiency, leveraging mixed-precision training and other advanced techniques.

D. Experiment Design

I experimented with various language models on my Olympics dataset, to determine which model performs best with fine-tuning. I also used the SQuAD dataset to train gpt2-large, to determine whether pre-training the model on a large QA dataset would improve performance on my specific dataset. I experimented using models without Lora, but ultimately applied Lora and 8-bit quantization to all my models to improve the memory and time requirements.

I evaluated a generic gpt2-large model (not fine tuned on any datasets) on my Olympics dataset to obtain a baseline model performance to compare to the performance of my fine-tuned models.

Initially I focused on observing loss, then I implemented additional validation measures, specifically ROUGE, BLEU, and BERT scores. I decided to not implement EM due to its rigidness and the fact that it will return a 0 unless every word is exactly the same in the generated answer as in the original answer. This is also why I chose to use ROUGE, BLEU, and BERT - because of their leniency and ability to understand answers that are close matches but not exact. I also am using human evaluation to evaluate answer quality, fluency, and relevance.

It is important to note that due to the extensive training time required for model training and evaluation, I wasn't able to complete all the tasks I planned. Many of my model training and evaluation scripts timed out on Beocat, despite setting them to run for many hours (90 hours in some cases). Due to the time limitations, I was only able to obtain evaluation results for two models: my baseline gpt2-large (not fine-tuned at all), and my FLAN-T5 model, which was fine tuned on my Olympics dataset.

III. Evaluation and Results

My main baseline for performance comparison is evaluation of pre-trained LLMs on generic QA tasks before applying fine-tuning. This allows for assessment of initial model performance. I also chose to evaluate pre-trained LLMs on the domain-specific 2024 Olympics dataset without fine tuning, to determine baseline model performance on the specific task at hand.

I initially investigated the loss in model training, and this is how I determined the SQuAD fine-tuning to not make a significant improvement to model performance. I then chose to apply Bleu and Rouge scores to determine model performance for each of my trained models. I initially planned to apply Exact Match (EM) and F1 scores, but instead opted for Bleu and Rouge due to their ability to recognize synonyms and therefore be more forgiving and accurate in determining model performance than a harsh metric such as EM.

A. GPT2 Model Results

I first evaluated gpt2-large without any fine-tuning at all, on my Summer 2024 Olympics dataset. The results I achieved are in Table I. We can see that the BLEU and ROUGE scores are very low, indicating that the un-trained gpt2-large did not perform well on the QA task.

TABLE I
BASELINE GPT2-LARGE EVALUATION METRICS

Metric	Score
BLEU Score	1.4527
ROUGE1 Score	0.0505
ROUGE2 Score	0.0342
ROUGEL Score	0.0505

Next, I fine-tuned my gpt2-large model on the full SQuAD dataset. The training and validation loss can be seen in Table II, and visualized in Figure 8. We can see that the loss is much higher than we'd want and expect from a model that performed well in fine-tuning.

Epoch	Training Loss	Validation Loss
1	2.64	2.61
2	2.63	2.62
3	2.61	2.62

TABLE II
TRAINING AND VALIDATION LOSS FOR GPT2-LARGE TRAINED ON FULL SQUAD DATASET

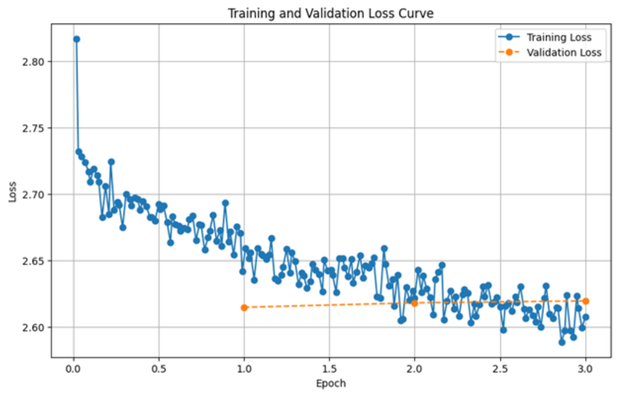


Fig. 8. Training and Validation Loss Plot for gpt2-large with Full SQuAD Dataset

After this, I took the gpt2-large model that I'd fine-tuned on the SQuAD dataset, and fine-tuned it on my Olympics dataset. The training loss for this can be seen in 9. We can see that the loss is much better than the gpt2-large only trained on the SQuAD dataset.

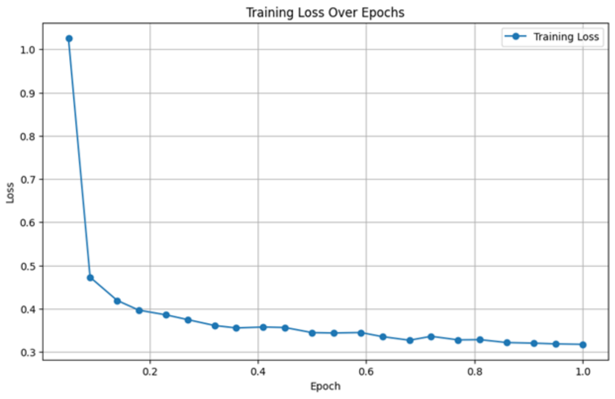


Fig. 9. Loss Plot for gpt2-large, SQuAD and Olympics fine-tuned

I also fine-tuned gpt2-large on only my Summer 2024 Olympics QA dataset. The training and validation loss plot can be seen in 10. Here we can see that gpt2-large performs significantly better when only trained on my Olympics dataset vs. being trained on only SQuAD or both SQuAD and Olympics. After these findings, I chose to stop using the SQuAD dataset, as I determined it was not adding any value to my project.

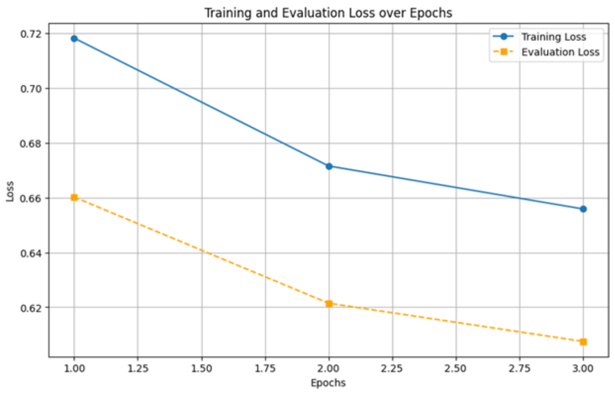


Fig. 10. Loss Plot for gpt2-large fine-tuned on Olympics

Next, I decided to train the more robust gpt2 model, gpt2-xl. The training and validation loss plot for gpt2-xl trained on my Olympics dataset can be seen in 11. We can see that gpt2-xl had even better loss than gpt2-large, indicating that the larger gpt2 model was even better at handling my QA task.

These were all the scripts I ran for the gpt2 models. Please note that the only evaluation results I was able to obtain for the gpt2 models was for the baseline, non-fine-tuned gpt2-large. I wrote scripts for evaluating all the other gpt2 models which were fine-tuned on the various combinations of the SQuAD and Olympics dataset and tried to run them. I ran into timing issues here, with many of the evaluation runs taking over 80 hours, and many Beocat jobs timing out before evaluation was

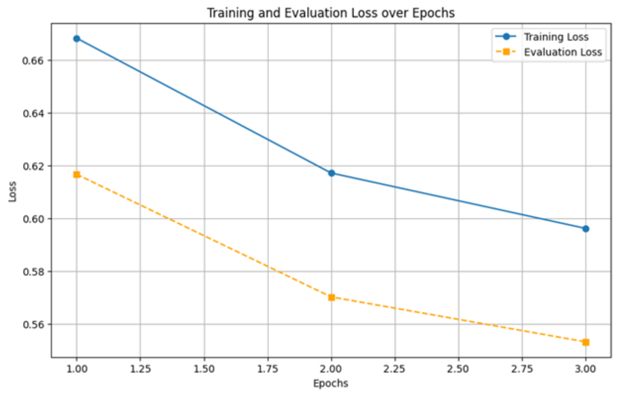


Fig. 11. Loss Plot for gpt2-xl fine-tuned on Olympics

complete. I continued attempting to run these evaluations until there was no longer time.

B. BERT Variant Results

I fine-tuned three BERT variants on my Olympics dataset: ALBERT, BERT, and RoBERTa. I will first discuss ALBERT.

My fine-tuned ALBERT model had a promising loss plot - while missing the data for the second epoch, we can still see that the loss starts lower than any of the gpt2 model losses, and significantly decreases by the third epoch, indicating the training was effective. The loss plot can be seen in Figure 12.

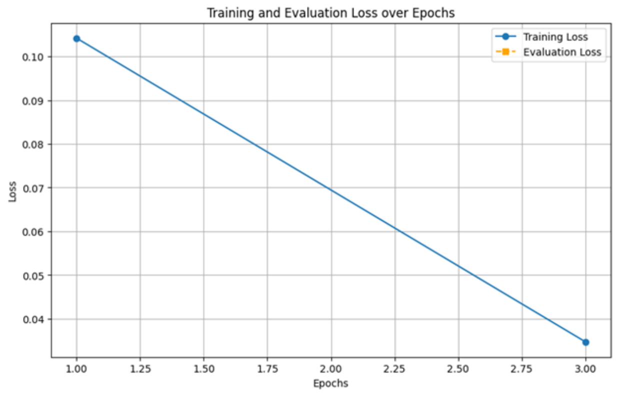


Fig. 12. Loss Plot for ALBERT fine-tuned on Olympics

I also fine-tuned BERT on my Olympics dataset. The loss plot can be seen in Figure 13. We can see that the loss starts low, and significantly drops for the second and third epochs. However, it is suspicious that the loss drops all the way to zero on the second and third epochs, and this does indicate potential issues during training.

Lastly, I fine-tuned the robust version of BERT, RoBERTa on my Olympics dataset. The loss plot can be seen in 14. We can see that this loss plot is very similar to BERT's loss plot, though the initial loss is lower than it was for BERT. We still see the issue of the loss dropping to zero on the second epoch and remaining there for the third epoch. This is not an expected result and indicates issues.

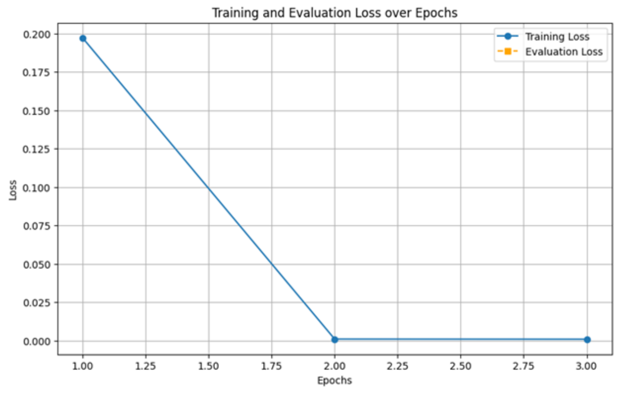


Fig. 13. Loss Plot for BERT fine-tuned on Olympics

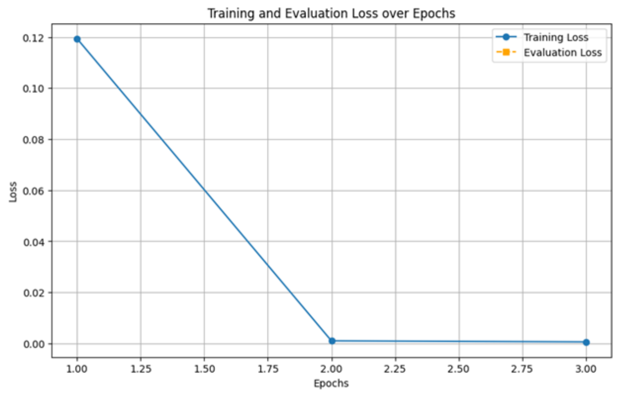


Fig. 14. Loss Plot for RoBERTa fine-tuned on Olympics

Similarly to the gpt2 models, I wrote scripts for evaluating each of my fine-tuned BERT variants, but I ran out of time. This was a common theme throughout my project, due to the nature of language models being computationally expensive, as well as my Olympics dataset being very large, with nearly 300,000 QA pairs.

C. FLAN-T5 Results

The last model I chose to fine-tune on my Olympics dataset was FLAN-T5. The loss plot for the training can be seen in 15. We can see that the training loss dropped significantly at the beginning of training, then still steadily decreased over the last epochs. While the last epochs don't vary too much, they are above zero and still indicate a model that is training well on the data. The validation loss remains the same throughout the training, barely above zero.

FLAN-T5 was the only model besides the baseline gpt2-large that I was able to obtain evaluation results for. I ran my evaluation script for FLAN-T5 two times, as I did not remove the padding tokens for my generated answers in the initial script. A sample of my generated vs. reference answers from this first run is in 16, and the evaluation metrics are in Table III. I modified my script to remove the padding tokens and re-ran it, and my scores were significantly improved. The answers for this run can be seen in 17, and the evaluation scores can be seen in Table IV.

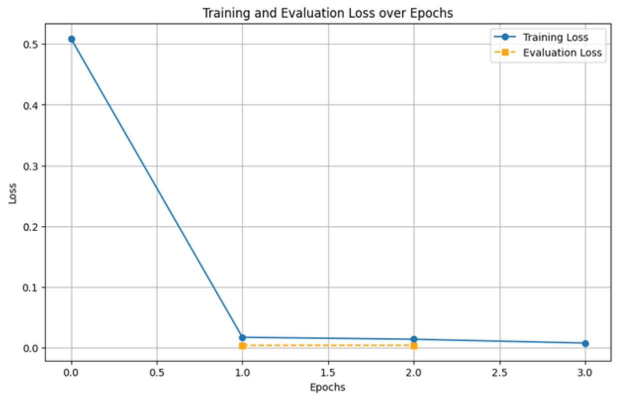
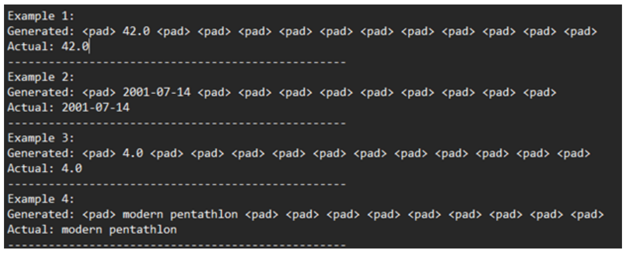
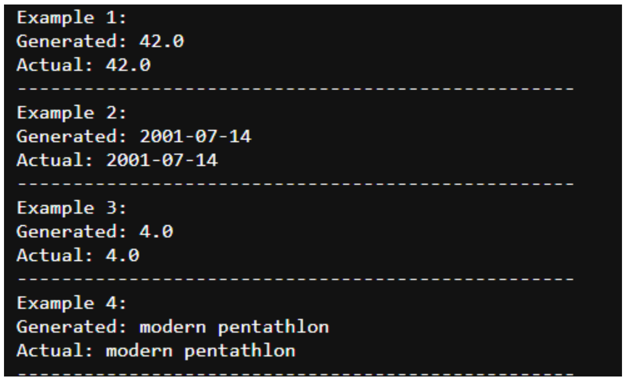


Fig. 15. Loss Plot for FLAN-T5 fine-tuned on Olympics



```
Example 1:
Generated: <pad> 42.0 <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad>
Actual: 42.0
-------------------------------------------------
Example 2:
Generated: <pad> 2001-07-14 <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad>
Actual: 2001-07-14
-------------------------------------------------
Example 3:
Generated: <pad> 4.0 <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad>
Actual: 4.0
-------------------------------------------------
Example 4:
Generated: <pad> modern pentathlon <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad> <pad>
Actual: modern pentathlon
-------------------------------------------------
```

Fig. 16. Generated vs. Reference Answers FLAN-T5, First run



```
Example 1:
Generated: 42.0
Actual: 42.0
-------------------------------------------------
Example 2:
Generated: 2001-07-14
Actual: 2001-07-14
-------------------------------------------------
Example 3:
Generated: 4.0
Actual: 4.0
-------------------------------------------------
Example 4:
Generated: modern pentathlon
Actual: modern pentathlon
-------------------------------------------------
```

Fig. 17. Generated vs. Reference Answers FLAN-T5, Final run

TABLE III
FLAN-T5 EVALUATION METRICS FOR FIRST RUN

Metric	Score
BLEU Score	5.4010
ROUGE1 Score	0.3556
ROUGE2 Score	0.2613
ROUGEL Score	0.3556
BERTScore	0.8514

TABLE IV
FLAN-T5 EVALUATION METRICS FOR FINAL RUN

Metric	Score
BLEU Score	80.1234
ROUGE1 Score	0.9480
ROUGE2 Score	0.6381
ROUGEL Score	0.9479
BERTScore	0.9831

D. Hypothesis Testing Results

I performed hypothesis testing based upon the results I got during my runs. Since the only evaluation metrics I was able to obtain were for my baseline gpt2-large and my tuned FLAN-T5, those are the models I performed hypothesis testing on. During my evaluations, I only saved the ending results for the metrics (e.g., ROUGE score means), so I back-propagated synthetic values based on these average values to perform hypothesis testing.

My hypothesis testing metrics resulted in mean scores, variance, mean squared error (MSE), t-statistic, and p-values. The full set of results can be seen in Tables V and Table VI.

TABLE V
COMPARISON OF METRICS FOR BASELINE GPT2-LARGE AND TUNED FLAN-T5 (PART 1)

Metric	Mean Baseline	Mean FLAN-T5	Variance Baseline
ROUGE1	0.035814	0.963109	0.016330
ROUGE2	0.037354	0.630180	0.018008
ROUGEL	0.059678	0.931593	0.023278

TABLE VI
COMPARISON OF METRICS FOR BASELINE GPT2-LARGE AND TUNED FLAN-T5 (PART 2)

Metric	Variance FLAN-T5	MSE	t-statistic	p-value
ROUGE1	0.015476	0.897096	51.733951	2.76×10^{-117}
ROUGE2	0.022404	0.384190	29.341993	2.53×10^{-74}
ROUGEL	0.016888	0.806432	43.287470	3.35×10^{-103}

Evaluation of these results shows that FLAN-T5 outperforms the baseline gpt2-large model significantly. For the mean scores, ROUGE1, ROUGE2, and ROUGEL all clearly outperform the baseline, with the ROUGE2 score having a less pronounced improvement. This is an expected result, as ROUGE2 is stricter than ROUGE1 and ROUGEL in how it determines correct answers, requiring exact bigram matches. We can see that the variance for the baseline model is much higher than the tuned model, showing that the tuned model is more consistent. The MSE shows a distinct performance gap between the two models, again with ROUGE2 having a less distinct improvement than ROUGE1 and ROUGEL. The t-statistic is high for all the metrics, which shows that the difference in means between the baseline gpt2-large model and the tuned FLAN-T5 model is very significant. Lastly, and most importantly, the p-values are far below the significance threshold (0.05), and in fact are so low they are effectively 0. This allows us to reject the null hypothesis with extremely high confidence, and confirm our alternative hypothesis. The conclusion from these results is that fine-tuning the FLAN-T5 model greatly improves performance in the domain-specific QA task.

IV. Summary and Future Work

My project involved cleaning a dataset and transforming it into a QA dataset, then training multiple language models on the dataset as well as a generic QA set. My results clearly show that the tuned model (FLAN-T5) significantly outperforms the baseline model (gpt2-large). This shows that tuning the LLM on my Summer 2024 Olympics dataset is very effective and highly improves model performance. The loss plots indicate that gpt2 models had the worst performance out of all the models I tried, with the BERT models having better loss and FLAN-T5 also performing well.

Future work would involve continuing to run the evaluation scripts for all the gpt2 and BERT models in order to obtain evaluation metrics for more models. This would allow me to compare the models against each other and determine which model performs better on the QA task overall, instead of solely comparing a tuned model to the baseline. This would require quite a lot of time, therefore future work on LLMs should start early and allocate many hours to the training and evaluation process. Additionally, I'd like to further clean and organize the data in the QA dataset. Some of the answers specifically were choppy and non-eloquent, which is due to them being concatenations of multiple columns in the dataset. I did this in order to make the answers descriptive and accurate, but I did not improve fluency. I'd also like to develop more QA pairs, as there are more questions I could've generated concerning the dataset. While the dataset size was good and it did not require more pairs, there were more areas that could've used questions. Lastly, I'd like to make a chatbot UI to allow for interactive prompting. I wanted to create one for this project, but I didn't have time, especially considering the extremely long training and evaluation times. This would likely require re-formatting the context, question, and answers and implementing pretty printing in order to have a chatbot capable of accepting user queries and responding in a more fluent manner. For future work, I'd also like to look into RAG (retrieval-augmented generation) to make my models more robust in retrieving up to date information. I'd also like to look into vector databases, as they store and index data for efficient retrieval. Integrating these techniques in future iterations will improve result accuracy and enable the model to retrieve answers more efficiently.

References

[1] V. Lomshakov, S. Kovalchuk, M. Omelchenko, S. Nikolenko, and A. Aliev, "Fine-tuning large language models for answering programming questions with code snippets," in *Proceedings of the International Conference on Computational Science (ICCS)*, (Prague, Czech Republic), pp. 172–185, 2023.

[2] S. Zhang, P. Zhu, M. Ma, J. Wang, Y. Sun, D. Li, J. Wang, Q. Guo, X. Hua, L. Zhu, and D. Pei, "Enhanced fine-tuning of lightweight domain-specific q&a model based on large language models," in *Proceedings of the 35th International Symposium on Software Reliability Engineering (ISSRE)*, (Tsukuba, Japan), 2024.

[3] J. Ye, Y. Yang, Q. Zhang, T. Gui, X. Huang, P. Wang, Z. Shi, and J. Fan, "Empirical insights on fine-tuning large language models for question-answering," *arXiv preprint*, vol. arXiv:2409.15825, 2024.

[4] P. FM, "Paris 2024 olympic summer games dataset," 2024. Accessed: 2024-11-07.

[5] Paris 2024 Organizing Committee, "Paris 2024 - olympic and paralympic games official website," 2024. Accessed: 2024-11-07.

[6] Wikipedia contributors, "2024 summer olympics," 2024. Accessed: 2024-11-07.

[7] P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang, "Squad: 100,000+ questions for machine comprehension of text," *arXiv preprint arXiv:1606.05250*, 2016.

[8] T. pandas development team, "pandas-dev/pandas: Pandas," *Zenodo*, 2020.

[9] N. Houlsby, A. Giurgiu, S. Jastrzebski, B. Morrone, *et al.*, "Parameter-efficient transfer learning for nlp," *arXiv preprint arXiv:1902.00751*, 2022.

[10] E. J. Hu, Y. Shen, P. Wallis, *et al.*, "Lora: Low-rank adaptation of large language models," *arXiv preprint arXiv:2106.09685*, 2021.

[11] T. Wolf, L. Debut, *et al.*, "Transformers: State-of-the-art natural language processing." https://huggingface.co/, 2020.

[12] H. Face, "Transformers reinforcement learning library (trl)," 2023.