Vinny Pilone
4/25/25
CMPSC 497

CMPSC 497 Final Project Report

Introduction

Creating an artificial intelligence approach to a research project is a challenging task. Not only does it require in-depth knowledge of artificial intelligence architecture, but it also requires a deep understanding and interpretation of the research question presented. In order to make these artificial intelligence approaches more accessible, we were tasked to fine-tune an LLM to give researchers an artificial intelligence approach to a given research problem. This was an extremely involved project, with some major facets including constructing the database, selecting an LLM, training (and reselecting) the LLM, evaluating performance and navigating various problems throughout. While I will discuss the various successes and failures throughout the rest of this report, I do think that this was a successful learning experience and taught me a lot about training LLM. However, without further ado, let's begin by discussing the dataset.

Dataset Construction

To begin, I started with searching for an appropriate data set to train the model on. I first tried a simple internet search (including google and huggingface as two of the major data sources), but as expected there wasn't a premade dataset available that appropriately fit what I needed. One promising dataset that was found was the "Younger" dataset, a collection of AI architectures designed to be used for autonomous AI generation. However, this dataset lacked a critical component for my intended solution, which was a link between the question and the architecture. Additionally, as the data was in the format of a directed acyclic graph, it wasn't practical to use for an LLM database.

Fortunately, I also found a much more useful dataset on HuggingFace. The other dataset I found was by user khushwant04, and was a collection of almost 1300 research articles, all covering various advanced aspects of Artificial Intelligence. Although not incredibly useful on their own, my plan at this point was to convert these papers into sets of [research question, approach] pairs using a pre-trained LLM. This would serve two purposes: information gathering and context training. My first goal with this method was to gather in depth information on various advancements in artificial intelligence. This would hopefully help to fix one of the core issues at the heart of this challenge, which is a lack of domain knowledge. By encoding these research papers (or at least summaries of their approaches) I would be able to offset this limitation and give the AI more information. My second goal was that this method of encoding would give more consistent results when prompting the AI. Rather than using papers of varying format, the consistent [research question, approach] pairs would be regular, and thus give more regular answers. However, the challenge still remained that I needed an AI to create these pairs, so I continued onwards to choose my LLM.

Vinny Pilone
4/25/25
CMPSC 497

LLM Selection

Selecting the LLM I would use for this project had three goals in mind. First, it had to be open source, so that I could track both how it was made and ensure that I could train it without issue. Second, I wanted the AI to be good at summarization, as it was my goal to use the same AI to generate the dataset and to solve the research task (furthering consistency within training). Finally, I wanted the AI to already have a decent baseline with complex reasoning challenges, since even with training it would be difficult to get a LLM from no AI knowledge to an expert capable of being helpful in a research setting.

I figured the best place to search would be Ollama, an open-source AI platform with a large number of LLM models. My search led me to two major frontrunners: OpenThinker and QwQ. OpenThinker is a fully open source LLM model based on Deepseek-r1. Designed to specialize in complex reasoning tasks, this was a major frontrunner for my experiment. The other major model was QwQ, a branch of the popular open source Qwen models. Not only is Qwen a frontrunner in open source AI data, but QwQ in particular was designed to excel in complex reasoning tasks. While both of these seemed like plausible contenders, unfortunately during testing OpenThinker produced a lot of garbled responses. I am unsure if this was an issue with the particular version I got or simply some error in the model, but QwQ seemed like the clear winner. However, this was a decision I would ultimately regret later in the training process.

Training

The first step in my training process was to generate the database. As I mentioned previously, to do this my plan was to take the Research Papers database and use my chosen AI to create [research question, approach] pairs. To do this, I used a method I found on a LinkedIn tutorial (submitted by user Robyn LeSueur) to tokenize the pdf reports and feed them through QwQ. However, this unfortunately ended up being my first major mistake. While QwQ did a good job of getting me the data I wanted, the process of summarizing these long documents took a really long time. Ultimately, I ended up summarizing two documents. one for training and one for testing, from each of the 72 research categories present in the database. However, this process literally took days to complete, as I had to run the AI locally on my machine due to distribution rights. This put me on a tight schedule to complete the rest of the training process, and it meant I wasn't able to get as much data as I would have liked. Thankfully, once the AI process was done it was a relatively easy task to clean and assemble the data for training.

Once I had all of my data gathered, the next step was to actually train QwQ. To accomplish this, I eventually settled on using HuggingFace's Trainer class, as it was very user friendly and let me adjust parameters without too much internal coding. Unfortunately for me, as soon as I attempted to train the LLM I quickly ran into two major issues: memory and tokenization.

The first major issue I encountered was memory. As soon as I attempted to actually train QwQ, I encountered a crash in my python code because of how much memory the trainer was asking for. No matter what I tried, I simply could not get the code to run past the initial training point. Eventually I sought out a different model, and hoping to keep the task in the same "family" of LLM I went onto HuggingFace and looked at their available Qwen models. I eventually ended up using the smallest model (a mere 1.5B parameters), a significant downgrade from my 32B QwQ model. However, despite later experimentation with 14B, 7B, and 3B Qwen models, as well as different batch sizes, this was the only model my personal computer was capable of running, and the memory problems didn't stop there. Once my model was trainable, I next ran into an issue where the requested buffer size for my database was 64GB (for centext, I have 32GB RAM). My first approach to solve this problem was to truncate my data to fit within the buffer. A binary search of possible sizes revealed that 350 tokens was the maximum possible size. However, this approach had a major flaw in that it would remove significant portions of the data. So, I put this issue on hold in favor of solving my second problem first.

The second major issue I encountered was figuring out tokenization. HuggingFace has a nice tokenizer class built in, but I was having trouble getting it to function properly. Learning how to manage the resulting columns from the tokenizer, and learning how to use those columns to get training metrics, was a task that I couldn't quite seem to crack. Fortunately, this issue ended up leading to a solution for not only tokenization, but also for my memory issue. I found a method on the HuggingFace documentation to train a LLM based on consistent prompts. This method had the entire conversation (in my case a simple question to answer) concatenated and then split into segments based on the max model length. This allowed me to get all of my data into the model, while also solving my memory problem. While this method did sacrifice some internal connections between questions and answers, I was hopeful it would still produce good results.

At this point the last thing to do was to figure out my overall training strategy going into the experimentation phase. I tried a variety of different epochs and model saving techniques, and eventually ended on the following technique: training for two epochs (making it very efficient to train the model) and evaluating the model at regular intervals. In my case, I evaluated the model every 10% of the total steps, which roughly equated to every 15 [research question, approach] pairs. Once I had determined this strategy, it was time to figure out how to evaluate the success of each model and what experiments I would attempt.

Evaluation Metric and Experiments

The final step in training was picking an evaluation metric. Fortunately, HuggingFace's Trainer API had a lot of easy to implement metrics and training methods. Personally, I choose to have my training metric be the evaluation loss, and also told HuggingFace to save the model weights each time a new lowest loss was found. This allowed me to both get the optimal solution based

on the testing dataset, but also allowed me to relatively easily avoid overfitting my data. Because HuggingFace saved the model weights for me, I could simply get the lowest weight and the rest would be deleted. Importantly, I did not choose the training set loss because of the overfitting problem.

After training, the next step would be deciding what the final evaluation would be. There were several options for this evaluation, but I planned to use a combination of two methods that I thought would be good choices. First, I would compare the final loss of each model compared to its test data. Because the test data was slightly different for each, this wasn't a perfect comparison, but it did give me a good baseline for training effectiveness. Unfortunately, they were all pretty close (between 1.8 and 2), so this didn't end up getting used much. However, I mainly choose to compare the responses for each model to a series of predetermined questions. Each model got the same exact questions, and I would simply look at the response for each to see how logical it is. This turned out to be a great strategy, as when I was setting up the questions I found that my initial trainings had generated a LLM that mostly generated decorative "*" characters, due to the frequency of them in my test data. I removed these characters from final training and testing datasets.

The final step before I could look at results was to determine my experiments. Due to my save strategies with HuggingFace, things like data amount and epochs were largely handled, and due to time constraints I was nervous about attempting to find an optimal learning rate. So instead, I opted to see how changes to the dataset setup affected the results. The table below shows each model I tested, as well as how the data was set up:

Model	Data	Reasoning
Base Model	N/A	Base model for comparison
"Fully Trained" Model	Prompt + Question + Approach, allowed to overfit	See effects of overfitting test data
"PromptQA" Model	Prompt + Question + Approach	Full dataset and instruction prompt
"QA" Model	Question + Approach	By lowering repeated phrases, hopefully remove redundant training
"Approach" Model	Approach	By only having the approach, hopefully just embed more information that can be reasoned with generally

“QandA” Model	Question, Approach	Separate question and approach, honestly mostly just curiosity

Once the models were all trained, I fed them all through an identical testing loop. This loop consisted of giving each model an input made of a consistent prompt + each of four questions. Each question was designed to test a different area of the model’s performance. The first question was a question from the training dataset, which tested the model’s recall ability. The second question was from the testing dataset, so it should be representative of eval. The third was a question in the exact style and format as both test and eval that wasn’t in either dataset. My hope with this question was to see the models performance on a truly unknown question. The final question was my research question from phase 1 of the final project, with the hope that this question would allow me to check the specific details of the response more closely. I compiled the results below.

1. Base Model
   The results of the base model weren’t unexpected, but are still worth discussing as a baseline. The baseline model (Qwen 1.5B) was overall decently well performing. For each question it gave reasonable answers that weren’t too far off topic. However, the answers it gave were somewhat vague, and sometimes did miss the mark. Additionally, it is worth noting that at the end of most responses the model would begin to repeat itself, a trend that continued to each of the fine-tuned models.
2. Fully Trained Model
   This model was originally a testing model, and was allowed to overfit the data for 5 epochs. Although not trained in the same way as the others, this model was used as a way to see how overfitting affected the results. In essence, this model just became trained to repeat the question over and over. Apparently there were enough buzzwords in the question asked to train the model to repeat this way. Although disappointing, this was expected.
3. Prompt QA Model
   This model was interesting, as it was trained on theoretically the most data. However, this model produced somewhat consistent results in that it would rephrase the question, or give one small portion of an answer, and then repeat that over and over. This was the most disappointing model as I really thought it would be most successful. However, it seems that the amount of data and repetition of information was too much, at least for this particular base model.
4. QA Model
   Another fascinating result, this model got much closer to a useful answer. While still suffering from the now standard issue of repeating information at the end, this model seemed to get much closer to a reasonable response. However, it also had the most

variation in its answer quality, with some answers being useful or almost useful and some answers being almost totally incorrect.

5. Approach Model
   This model was the best performing of the experiments. Although it also had the same issues of repeating information at the end of its responses, it usually got out a complete or almost complete response before that began to occur. Additionally, the responses it gave, while not perfect, were often possible solutions to the given research problem. However, it did still seem to struggle with context for its solutions, and often the given solution would require some deciphering before it was useful.
6. QandA Model
   This model was the runner up to the Approach model for the best results. As with every other model, this model would repeat itself at the end of its responses. However, unlike most of the other models, this model would give a decent response until it began to repeat itself. The reason why it is runner up is because the spot it would begin to repeat itself at was very variable, and sometimes interrupted what would otherwise be a decent answer.

Overall, I felt that training the model went pretty well. My training strategy seemed to give the best model options, and clear improvement was seen compared to the base model. It was surprising to me that the approach only method was the best version though. I suspect that the combination of less data, combined with giving it only useful information (as opposed to wasting some training on questions and prompting) did help considerably. However, I do feel that with a larger model and more training time (using a lower rate) the same methods could be used to create a much better LLM.

Learning and Problems

Reflecting back on this project, I am very pleased to say that it seemed to go much smoother than the midterm project. Not only was I able to do a lot more, but I felt more confident about what I needed to do throughout the process. After all was said and done, there was a lot that I learned, but also several problems that I ran into throughout the process.

Overall, I learned two major things. First, I learned a lot about the importance of and the creation of useful datasets. All of my previous experiences have had a dataset handed to me, and trying to solve a problem that required a lot of very specific data made me realize just how important it is to have that data. Additionally, I learned a lot about using HuggingFace and fine-tuning LLM. Honestly, it wasn't quite as difficult as I thought it would be, and in particular if I had a better dataset I think it would be easier. Knowing the process of fine tuning a model though is something I plan to use a lot more in the future. However, to use it more, I will have to overcome the issues I found as well.

Vinny Pilone
4/25/25
CMPSC 497

Within this project I had three major challenges. My first (and largest) challenge was memory and the limits of my machine. It should be obvious, but training an AI is really difficult, particularly on a smaller scale. In particular, even with 32GB of memory my ability to train was significantly limited by my machine, forcing me to a smaller model. While I was still able to complete the project, I do definitely feel that with a better machine I could have done a lot more. Also, when I was training, my computer was extremely slow to complete other tasks. This made it hard to manage training time and my other classes. My second challenge was the data. While I did learn a lot about getting data, the fact is that even the method I used had a lot of issues with formatting and quality. Getting good data takes a lot of time and energy that I simply didn't have, and that made the project a lot more difficult. The final challenge I had was evaluating the LLM to produce the output I wanted. I had a lot of success in terms of training parameters, but when it came to actually ensuring I got the output I wanted it was extremely difficult. Some of the issues I was able to overcome (like seeing repeated symbols in my response made me clean those symbols from the data), but others were much more difficult. In particular, trying to create an LLM with incredibly specific domain knowledge was tricky as I personally didn't understand all the information it would receive and output. This meant that evaluating responses was harder due to my own personal limitations.

Final Thoughts

While this project had its strengths and weaknesses, I overall felt that this was an incredibly successful project. I learned a lot about the entire LLM tuning process from start to finish, and I feel confident that I am able to use this knowledge well in the future if given the chance. I did feel a bit of a limitation due to my own knowledge of AI, as I often had to look up different metrics or methods being used to ensure I could get the result I wanted. However, each problem was eventually solved, and the final results, while less than perfect to be sure, were still incredibly satisfying to see. Getting to see a lot of complex work pay off was super cool, and in particular getting to experiment with advanced technologies is always exciting. Finally, I did want to again mention that this project was so much more successful than the midterm. With this project I felt much more confident, in control, and while there were definitely bumps in the road I think the end result speaks for itself in terms of quality. I'm really appreciative of the chance to get to learn about all this, and I'm excited to continue my studies of AI in the future.