Building an AI-Powered Robofest Chatbot: A Fine-Tuning Approach with OpenAI and Hugging Face Exploration

Igri Fishta, BS Computer Science Candidate, CoAS; Annalia Schoenherr, BS Computer Science Candidate, CoAS; CJ Chung, PhD, Professor, CoAS

College of Arts and Sciences, Lawrence Technological University

Abstract

This project designs and implements an AI-powered chatbot to answer questions about the Lawrence Technological University Robofest [6] event. Currently, no chatbot exists for this purpose, and creating one will enhance user-friendliness, provide valuable event information, and boost engagement and participation. The chatbot uses a fine-tuned OpenAI language model to manage both simple and complex interactions with quick and accurate responses. Key project steps include collecting and preparing JSON data on Robofest, fine-tuning the model with OpenAI and Hugging Face, and testing its performance with 20 event-related questions. The project also involves designing a user-friendly front-end using JavaScript and HTML. The final outcome was an operational chatbot that answers user questions effectively, despite limitations like the cost of running the models and challenges in dataset conversion. Iterative testing and data refinement enhanced the model's robustness, and future efforts may focus on deploying the chatbot on the live Robofest website. However, using the fine-tuning technique did not turn out very optimal because of the large amount of data updating it required, so using a different approach instead can give better results.

Introduction

In today's digital landscape, user experience is crucial for engaging participants effectively, especially on websites that host events like Robofest at Lawrence Technological University. The current Robofest website lacks an interactive component to assist users, which presents an opportunity to enhance user engagement. Integrating a chatbot could significantly improve user-friendliness by providing quick access to important information about the event.

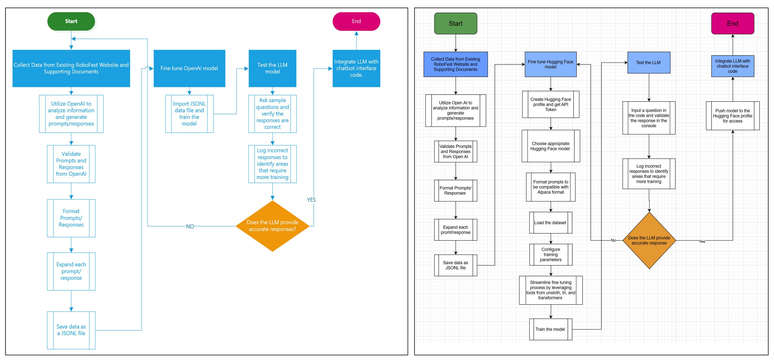
Recent studies indicate that chatbots are increasingly popular among consumers, with **68%** of users having utilized them for customer service interactions [5]. This statistic underscores the effectiveness of chatbots in delivering assistance and support to users. By leveraging OpenAI and Hugging Face technologies, this project aims to create a robust AI-powered chatbot capable of handling both simple and complex inquiries.

The implementation of this project followed a structured approach: initially, data was gathered and formatted in a question-and-answer JSONL format, which served as the foundation for training the model. Once the training data was prepared, the model was fine-tuned and rigorously tested to ensure its accuracy and functionality. We then connected the trained model to a test website to verify its connectivity and performance. After confirming that everything operates smoothly, we updated the training data with additional information to further enhance the model's capabilities.

In conclusion, this project aimed to develop an AI-powered chatbot that enhances user interaction and engagement through efficient and accurate information delivery. By systematically collecting and refining training data, testing model performance, we created a prototype that not only meets user needs but could also contribute to the overall success of the Robofest event. This initiative demonstrates the potential of chatbots to transform user experiences in event management and foster greater participation in academic and community activities.

Design

Utilization of Large Language Models: OpenAI vs Huggingface



After conducting research on fine-tuning models, OpenAI emerged as one of the most popular platforms for similar projects. To utilize OpenAI, we followed a video tutorial available on the RIIS website [4]. The initial step involved formatting our data into the appropriate JSONL format. Once formatted, the file was uploaded to OpenAI for validation. After successful validation, the data was used to fine-tune a model. This newly trained model was then accessible for testing via the OpenAI playground.

Following this, a simple chatbot web application was developed to integrate the model, allowing us to test its functionality directly within the chatbot application. The integration process was relatively straightforward, aided by comprehensive documentation provided by OpenAI. The primary drawback of OpenAI, however, is its cost. Depending on the chatbot usage, expenses can accumulate significantly.

Hugging Face is another widely used platform for creating and fine-tuning models, with the key advantage of being free. However, the fine-tuning process on Hugging Face proved to be more complex compared to OpenAI [2]. To proceed, we followed a YouTube tutorial on a similar project [3]. We used the same JSONL format for our data but had to write Python code to properly train the model.

Once trained, the model was uploaded to our Hugging Face account for deployment. However, integrating the model with our website presented challenges. Despite following all the recommended steps, we encountered errors when attempting to use the model. Further investigation revealed that the model lacked sufficient activity for deployment. Addressing this required either deploying it to Inference Endpoints, which incurred costs, or increasing its social visibility—a less feasible solution [2].

Given these complications, we opted to continue with OpenAI. Although it involves a cost, it is more manageable than the alternatives posed by Hugging Face. Additionally, OpenAI offers more extensive documentation, which is beneficial for future development [1].

Training Process

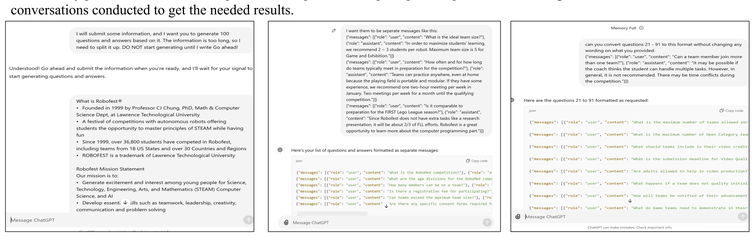
Generating Training Data using ChatGPT

Using Chat GPT to Process Information

The information housed on the Robofest website contains a significant amount of information and extensive documentation. Manually reviewing all this material to create a training dataset of questions and answers would be both challenging and inefficient. To simplify this process, we utilized Chat GPT, a tool capable of efficiently generating questions and answers based on provided content.

We input the necessary documents, whether in PDF or Word format, into Chat GPT with a prompt instructing it to generate questions and answers. ChatGPT performed well, typically generating a substantial number of questions, though not always reaching the target of 100 in a single attempt. When fewer than 100 questions were generated, additional prompts were issued to incrementally build the dataset until the desired count was achieved.

It was found that asking Chat GPT to handle smaller amounts of data made the output more accurate. In addition, ChatGPT occasionally provided undesirable output, which required a new prompt to be provided. The images below show the Chat GPT conversations conducted to get the needed results.



Expanding The Data

Fine-tuning requires a large amount of data for training. Having one prompt and response per question to train on was not effective in training the model. To increase the amount of data we had to work with, ChatGPT was used to reword each prompt five times to provide more examples to train with. An example of the prompt used to do this was:

"you are helping me expand my data to fine-tune an LLM for a chatbot. Consider the prompt provided and reword it in five different tones representing the students, parents, or mentors who may ask about a Robofest competition. Use first, second, and third-person points of view. Keep the same response and JSONL format"

This provided new prompts like the following:

"Where do I register as a new coach for Robofest?"
"What steps should I take to become a registered coach in the Robofest system?"
"How can I sign up as a new coach on the Robofest platform?"
"Can you guide me on how to register as a new coach for Robofest?"
"What is the process to register as a new coach in the Robofest system?"
"How do I register as a new coach in the Robofest system?"

JSON Formatting

The best format for Open AI data to process data is JSONL file, where L stands for line. Each line is considered a single JSON object, and it makes it easier for the Open AI to read it. The JSON file could look something like this:

```
{
  "Message" : [
            { "role" : "user", "question" : "When is Robofest 2025?" },
            { "role" : "chatbot", "answer" : "Robofest 2025 takes place on May 15th." }
                  ]
}
```

Once we have this format, we can convert it to a JSONL file where it would look something like this:

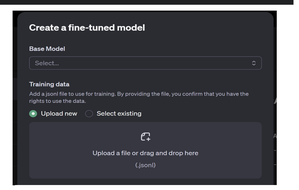
{ "message" : [{ "role" : "user", "question" : "When is Robofest 2025?" },{ "role" : "chatbot", "answer" : "Robofest 2025 takes place on May 15th." }] }

The next step involved formatting the generated questions and answers into the required JSONL format. To do this, we provided Chat GPT with a sample of the desired format and prompted it to convert the generated questions and answers accordingly. While this task occasionally provided inaccurate responses containing incorrect formatting, the process was improved by providing a more specific prompt for OpenAI to use which included examples for it to follow. An example of a prompt used can be seen as follows:

User
Keeping the same prompts and responses, format your response using the following format:
{"messages":{"role": "user", "content": "**Prompt here**"}, {"role": "assistant", "content": "**Response here**"}}
{"messages":{"role": "user", "content": "**Next prompt here**"}, {"role": "assistant", "content": "**Next response here**"}}

Fine-Tuning Process

The base model used to fine-tune was the **gpt-4.0-mini-2024-07-18** due to its cost effectiveness and the robustness of its existing language model. Each model trained created a new model, which was used iteratively to train the existing model on new data.
Training was done with the OpenAI fine-tuning interface. The interface allows you to provide training data in a JSONL format, and customize the training settings which includes base model to train on, suffix, seed, batch size, learning rate multiplier, and number of epochs. The log of fine-tuning sessions can be seen in the following table:



An example of the fine-tuning interface used for training the model:

Date	Model Name	Tokens Trained	Time Taken to Train
9/24/2024	ft:gpt-4o-mini-2024-07-18:personal::AB7XyYmL	5,775	11 minutes, 14 seconds
10/22/2024	ft:gpt-4o-mini-2024-07-18:personal::ALF9l9LS	15,807	8 minutes, 38 seconds
11/4/2024	ft:gpt-4o-mini-2024-07-18:personal::AQ09wEci	48,933	28 minutes, 9 seconds
11/14/2024	ft:gpt-4o-mini-2024-07-18:personal::ATZQuleK	265,251	40 minutes, 26 seconds
11/21/2024	ft:gpt-4o-mini-2024-07-18:personal::AW3mpIpl	56,148	29 minutes
12/5/2024	ft:gpt-4o-mini-2024-07-18:personal::AbBCLEE3	13,428	11 minutes, 5 seconds
12/12/2024	ft:gpt-4o-mini-2024-07-18:personal::AdfyJTor	15,195	9 minutes, 51 seconds

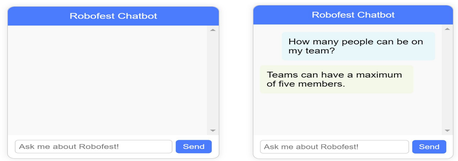
JSON File Validation

Once the data was in JSON format, it was validated using a simple Python script to verify and correct the formatting of each line. The script reads the original file, containing questions and answers formatted by ChatGPT, and checks for any discrepancies. If errors are found, such as missing brackets, parentheses, or commas, the script corrects them. Once the entire file is processed, the script generates a new output file with the corrected format, ready for training the model in OpenAI. This step is crucial because even minor formatting errors can cause the file to fail validation, despite Chat GPT being prompted to follow a specific format. Below is the code used to perform this validation:

```python
def correct_jsonl_format(input_file, output_file):
    corrected_entries = []
    with open(input_file, 'r') as f:
        for line in f:
            line = line.strip()  # Remove leading/trailing whitespace
            if not line:  # Skip empty lines
                print("Skipping empty line")
                continue
            try:
                # Load the JSON entry
                entry = json.loads(line)

                # Validate the structure
                if 'messages' in entry and isinstance(entry['messages'], list):
                    for msg in entry['messages']:
                        if not (set(msg.keys()) == {'role', 'content'}):
                            raise ValueError("Invalid message format")

                    # Append corrected entry to the list
                    corrected_entries.append(entry)
                else:
                    print(f"Skipping invalid entry: {line}")
            except json.JSONDecodeError as e:
                print(f"Error decoding JSON: {e} for line: {line}")
            except ValueError as e:
                print(f"Error: {e} for line: {line}")

    # Write the corrected entries to a new file
    with open(output_file, 'w') as f:
        for entry in corrected_entries:
            json.dump(entry, f)
            f.write('\n')
        print(f"Wrote entry: {entry}")  # Add this line to verify writing
```

User Interface

The front end development for the chatbot was created using html, css, and JavaScript. It consists of a simple icon that rests in the bottom-right corner of a webpage, which expands into a dialogue box when clicked on. From there, users can input their question and receive a response from the fine-tuned language model. An example of the code used for creation and the user interface can be seen as follows:




```javascript
// Call the OpenAI API
try {
  const response = await fetch('https://api.openai.com/v1/chat/completions', {
    method: 'POST',
    headers: {
      'Authorization': 'Bearer sk-[redacted]',
      'Content-Type': 'application/json'
    },
    body: JSON.stringify({
      model: 'ft:gpt-4o-mini-2024-07-18:personal::AQ09wEci', //  MODEL VERSION: 11.04
      messages: [
        { role: 'user', content: userInput }
      ]
    })
  });

  if (!response.ok) {
    throw new Error('Network response was not ok: ${response.statusText}');
  }

  const data = await response.json();

  const botResponse = document.createElement('div');
  botResponse.classList.add('bot-response');
  botResponse.textContent = data.choices[0].message.content; // Adjust to match OpenAI's response s
  chatHistory.appendChild(botResponse);
} catch (error) {
  console.error('Error fetching the LLM response:', error);
  const errorResponse = document.createElement('div');
  errorResponse.classList.add('bot-response');
  errorResponse.textContent = 'Sorry, there was an error retrieving the response.';
  chatHistory.appendChild(errorResponse);
}
```

Testing and Results

To evaluate the accuracy of our model's responses, a series of tests with 20 questions each was conducted. Initially, the model's performance was weak, often providing inaccurate answers. The average score from these responses was 1.2 out of 5. After incorporating additional data, the average score improved to 1.8, though the model still required significant enhancement before being suitable for real-world interaction. An example of the prompt and response from early testing can be seen as follows:

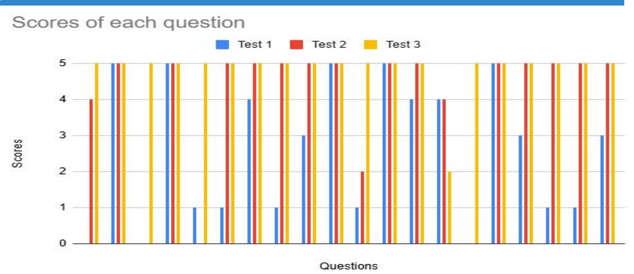
Prompt: "How do I register as a coach in the Robofest system?"
Fine-tuned response: "To register as a coach, you must be a full-time employee of the participating school. Coach registration occurs during the team registration process."

The response from the model in the shown case is inaccurate and vague. This was believed to be due to a lack of training data. To remedy this, a third test was conducted by rephrasing and including the same question five times in different forms. This approach raised the average score to 2.6, offering valuable insights into strategies for further improvement. Next, we analyzed the questions where the model's answers were less accurate, and the model was re-trained specifically on those problematic areas. With this targeted adjustment, the average score increased to 4.0 and later to 4.85, demonstrating a strong overall performance. An example of the new responses can be seen in the following:

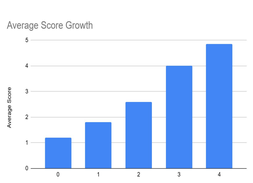
Prompt: "How do I register as a NEW coach in the Robofest system?"
Fine-tuned response: "Submit New Coach Registration online under Registration at robofest home page. See section 4 of the Robofest General Rules on the 2025 Main Page."

This response was more accurate and helpful, and proved that training the model on more data was beneficial to its overall accuracy. While a significant amount of data was generated for the model, resources were limited to generate the quantity and quality adequate for highly accurate responses. More efficient methods for data generation is a topic to be further explored to enhance the efficacy of the model.

To better visualize the testing process, two graphs were created. The first graph in the top illustrates how the scores for 20 specific questions for each test, while the second graph beneath it highlights the progression of the model's average score over time.



Scores for all 20 questions across 3 tests



Progression of fine-tuned models response scores over time.

References

[1] OpenAI API reference, https://platform.openai.com/docs/api-reference/introduction (accessed Apr. 18, 2025).
[2] "Hugging face - documentation," Hugging Face - Documentation, https://huggingface.co/docs (accessed Apr. 18, 2025).
[3] YouTube, https://www.youtube.com/watch?v=tjahjuROCWk (accessed Apr. 18, 2025).
[4] "Create your own customized website chatbot using chatgpt," RIIS, https://www.riis.com/blog/create-your-own-customized-website-chatbot-using-chatgpt (accessed Apr. 18, 2025).
[5] "24 amazing chatbot statistics for 2024," Backlinko, https://backlinko.com/chatbot-stats (accessed Apr. 18, 2025).
[6] Robofest 2025, www.robofest.net

Takeaways

What was Limited:

Quantity of data - Large language models require extremely large amounts of data to train, more data can be generated to further enhance the accuracy of the model.

Quality of responses- Responses from the model may not provide the most detailed information. Further research is required to determine how responses can be improved and include hyperlinks to specific pages and documents.

Implementation- The chatbot application runs locally and has not been tested with unknown users. Further testing for robustness is required.

What we Learned:

Enhancing user interaction - developing an AI chatbot will improve the user experience with Robofest by providing quick, accurate responses

Importance of data preparation - properly formatting training data into JSONL format was critical for successful model fine-tuning

Integration challenges - while Open AI offered smoother integration with the chatbot interface, Hugging Face posed deployment difficulties

Testing - testing the chatbot in the Open AI Playground provided insights into the performance of the fine-tuned model compared to generic models

Iterative development - the iterative nature of the project - refining data, validating formats, and addressing platform-specific challenges - highlighted the importance of adaptability in AI development

Prototype success - creating a functional chatbot prototype lays the groundwork for future development and deployment on the Robofest website

Future Potential - this project serves as a foundation for further research and development, including expanding the chatbot capabilities and improving its deployment on live platforms