Project_P2: Finetuning models for Domain-Specific Tasks

Start Assignment

- Due Wednesday by 11:59pm
- Points 100
- Submitting a website url or a file upload

Project Overview

Warning: This part of the project is hard. Please start early.

This is part 2 of our semester-long project: Intelligent Human Assistant.

In part 1, we aimed to make the command line interface more usable by allowing users to enter natural language queries. In this part, we will add a new capability—making computer use easier with Voice Control (on iOS) or Voice Access (on Android). Let's call it Voice User Interface (VUI) to be agnostic to iOS or Android. Like command lines, VUIs are very rigid—users must memorize commands, and any minor errors in command syntax are discarded by the VUI system.

Our goal is to clean up users' utterances in real-time using a fine-tuned small language model (e.g., Gemma 3 270M). It takes the raw user command and returns an output that the VUI expects at that time.

To make this system practical, we use two devices: a phone where the VUI is running, and a laptop where the LLM chatbot is running. For now, the chatbot takes user commands via text input, cleans them up, and reads them out loud so that the phone (nearby) listens to the command and acts upon it.

Background

Please watch this short tutorial on how to use VUI for basic interaction.

0:00/0:00

Refer to your OS-specific VUI documentation:

- iOS: https://support.apple.com/guide/iphone/use-voice-control-iph2c21a3c88/ios (https://support.apple.com/guide/iphone/use-voice-control-iph2c21a3c88/ios)
- Android: https://support.google.com/accessibility/android/answer/6151848?hl=en (https://support.google.com/accessibility/android/answer/6151848?hl=en)

Architecture

Similar to Part 1, the system includes:

- An MCP server hosting several tools

- An MCP client hosting a locally-run LLM and offering a terminal-based text input

However, unlike Part 1, the MCP client now:

- Runs through a graphical user interface based on Streamlit (see **HW-RAG (https://psu.instructure.com/courses/2416221/assignments/17594184)**)
- Has speech output capability (through the pyttsx3 library)

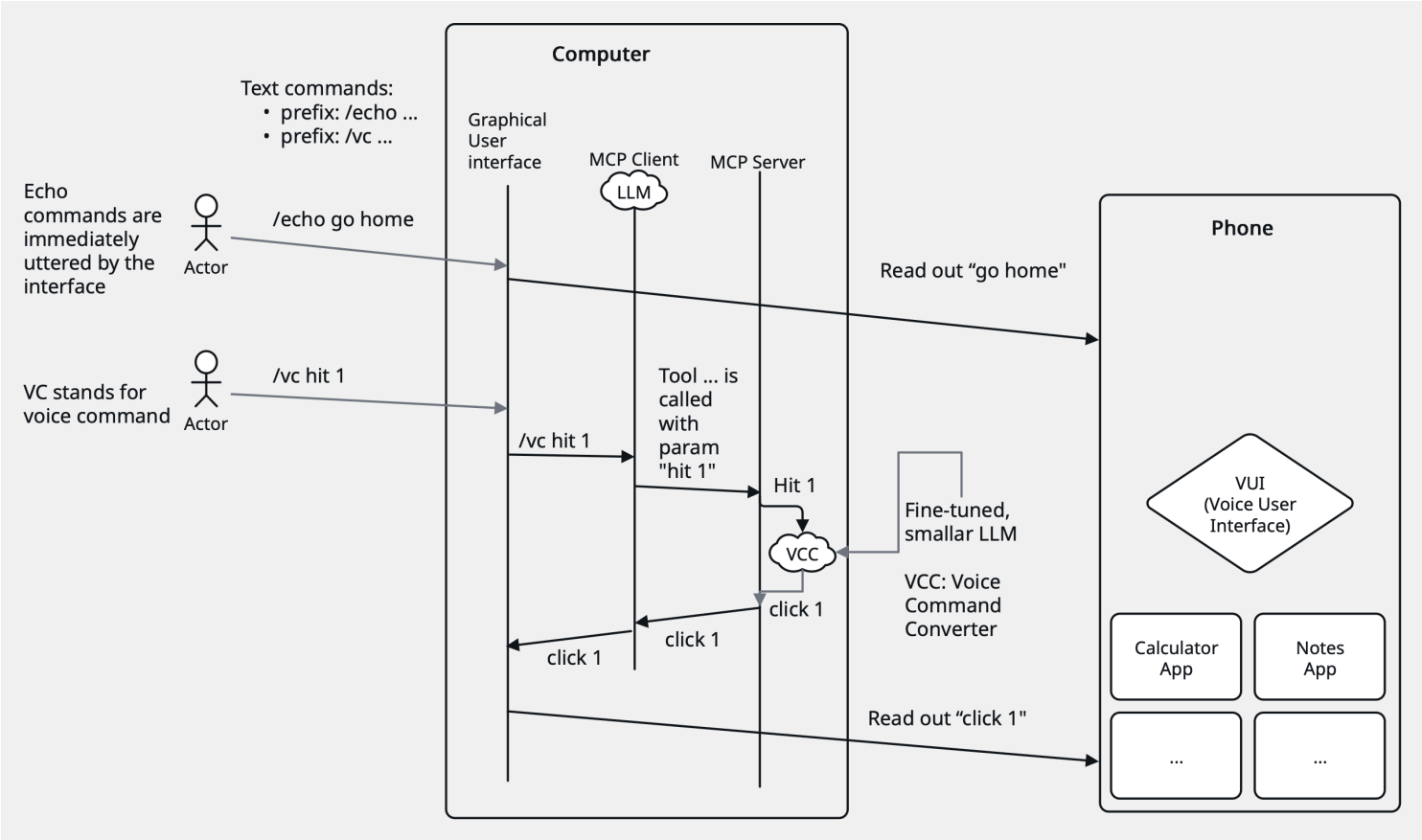
Additionally, the MCP server has a new tool that internally calls the smaller, fine-tuned LLM (let's call it Voice Command Converter or VCC). You need to fine-tune this model.

Users can type two types of commands: /echo [command] and /vc [command]. /echo commands are immediately read out by the interface.

Here is a demo of how /echo commands can be used:

/vc commands will go through the entire MCP pipeline (UI -> MCP client -> MCP server -> VCC)

The system architecture is shown below:



Expected outcome:

0:00/0:00

Implementation Guide

- Get yourself familiarize with VUI commands
- Use the Gemma 3 (270M) for VCC
- Use this Unsloth example notebook as your base: **Gemma 3 (270M) — Unsloth Colab Example** (https://colab.research.google.com/github/unslothai/notebooks/blob/main/nb/Gemma3_%28270M%29.ipynb#scrollTo=6bZsfBuZDeCL)
  - Run the notebook in Google Colab (Unsloth does not currently work on macOS). Use your PSU email to get premier tier GPU resource in Colab
  - Replace the example dataset with **your dataset (instruction below)**. You can upload your dataset to Google Colab and use its path.
  - Modify the dataset mapper, tokenization, and training code as needed to handle your text pairs.
- Train and evaluate your fine-tuned model.

  - Save the final model and tokenizer.
  - Test a few unseen voice commands and verify that the model outputs the expected structured format. You can also create a small test dataset with 100-150 samples and evaluate the model on that. In that case, you can either do string matching to see how accurate the model is or use the ROUGE score. Use this **code snippet (https://psu.instructure.com/courses/2416221/files/183477932?wrap=1)** ↓ **(https://psu.instructure.com/courses/2416221/files/183477932/download?download_frd=1)** to compute the ROUGE score
- Export the fine-tuned model (as GGUF format) to a local directory in Colab. **Please download the saved model and the generated *Modelfile* immediately; otherwise, the file might be lost since Colab terminates the session if it is inactive for a while**.
- Put the GGUF file and the Modelfile in the same directory on your local computer, and run the following command from that directory to add the model in Ollama:

```
ollama create gemma-3-270m-vc-finetuned -f ./Modelfile
```

After you have a working, fine-tuned model (VCC) integrate it as a callable tool on your MCP server. Use the model '`gemma-3-270m-vc-finetuned`' for implementing the tool.

- In rag_server.py, add a new tool named, for example:
  - ```
    @mcp.tool()
    async def correct_command(query: str) -> str:
    """
     Translate natural voice commands into standardized structured commands.
     Args:
          query: The user's natural-language command.
     Returns:
           The translated voice command as a plain string.
     """
    # your code here
    ```
    - Inside the tool:
    - Load your fine-tuned model using ollama.
    - Generate the translated command given the input prompt.
    - Return the translated command as a string.
- In the UI side, route /echo commands and /vc commands properly (see the architecture diagram)
- On the client side:
  - In rag_client.py, modify the system prompt to make sure the LLM understands which one is the voice command and which one is the previously implemented commands (e.g., terminal, my_files, arithmetic operations)
  - Implement a TTS component (e.g., using `pyttsx3`) that reads out the translated command in real time. Upon receiving the cleaned up command from the VCC, make sure it only reads out the voice command.

Dataset Creation

You must create your own dataset for training. In fact, dataset creation is the main challenge to fine-tuning any model.

- In your dataset, each entry should have the following five attributes. Feel free to use ChatGPT or other AI tools to get help with this step. Once this dataset is created, export it as a .jsonl file. A few entries from our dataset is shown below.
  - Include at least 300–500 examples of diverse natural language voice commands.
  - Cover synonyms, misspellings, or extra words users might say (e.g., “please,” “hey,” etc.).

Sample rows:

task	natural_command	input	expected_output	conversations
Convert the following natural language command	I was wondering if you could select 'tonight'	I was wondering if you could	SELECT tonight	[{"content":"Convert the following natural language command to the correct voice control command format.","role":"system"},{"content":"I was wondering if
    ```

task	natural_command	input	expected_output	conversations
to the correct voice control command format.		select 'tonight' | selection:		you could select 'tonight' | selection: ","role":"user"},{"content":"SELECT tonight","role":"assistant"}]
Convert the following natural language command to the correct voice control command format.	alright umm redo this	alright umm redo this | selection:	REDO THAT	[{"content":"Convert the following natural language command to the correct voice control command format.","role":"system"},{"content":"alright umm redo this | selection: ","role":"user"},{"content":"REDO THAT","role":"assistant"}]
Convert the following natural language command to the correct voice control command format.	get text now	get text now | selection:	SELECT now	[{"content":"Convert the following natural language command to the correct voice control command format.","role":"system"},{"content":"get text now | selection: ","role":"user"},{"content":"SELECT now","role":"assistant"}]

Below are the lines from our dataset.jsonl file representing the above table. You can download it from **here (https://psu.instructure.com/courses/2416221/files/183478448?wrap=1)** **(https://psu.instructure.com/courses/2416221/files/183478448/download?download_frd=1)** :

```
{
  "task":"Convert the following natural language command to the correct voice control command format.",
  "input":"I was wondering if you could select 'tonight' | selection: ",
  "expected_output":"SELECT tonight",
  "conversations":[
    {
      "content":"Convert the following natural language command to the correct voice control command format.",
      "role":"system"
    },
    {
      "content":"I was wondering if you could select 'tonight' | selection: ",
      "role":"user"
    },
    {
      "content":"SELECT tonight",
      "role":"assistant"
    }
  ]
}{
  "task":"Convert the following natural language command to the correct voice control command format.",
  "input":"alright umm redo this | selection: ",
  "expected_output":"REDO THAT",
  "conversations":[
    {
      "content":"Convert the following natural language command to the correct voice control command format.",
      "role":"system"
    },
    {
      "content":"alright umm redo this | selection: ",
      "role":"user"
    },
    {
      "content":"REDO THAT",
      "role":"assistant"
    }
  ]
}{
  "task":"Convert the following natural language command to the correct voice control command format.",
  "input":"get text now | selection: ",
  "expected_output":"SELECT now",
  "conversations":[
    {
      "content":"Convert the following natural language command to the correct voice control command format.",
      "role":"system"
    },
    {
      "content":"get text now | selection: ",
      "role":"user"
    },
    {
      "content":"SELECT now",
      "role":"assistant"
    }
  ]
}
```


Submission Guidelines

Submit a zip file containing:

- All your code:

    - rag_server.py (with new tool)
    - rag_client.py (updated prompt + TTS)
    - ui.py
    - voice_commands.jsonl (the dataset)
    - Training notebook (modified Colab notebook)
- A short demo video showing:
    - Model translation
    - TTS output
    - Real-world device control demonstration
- A brief text file **report.txt** summarizing:
    - Dataset creation process
    - Model used and fine-tuning parameters
    - Example outputs
    - Any issues or potential improvements