Hemant Goyal
Conversational Sleep Coach
Fine-tuned LLM with STT/TTS and Gradio Interface
24/05/24

1. Introduction

This document outlines the implementation of "NapCoach AI," a conversational AI assistant specialized in sleep coaching. The system integrates Speech-to-Text (STT) for processing voice input, a fine-tuned Large Language Model (LLM) to generate intelligent and context-aware responses, and Text-to-Speech (TTS) to deliver these responses in audible form. User interaction is facilitated through a Gradio-powered web interface. The entire project is developed as a Google Colab notebook. The core LLM, mistralai/Mistral-7B-Instruct-v0.2, is fine-tuned using Low-Rank Adaptation (LoRA) on a custom dataset comprising sleep-related questions, answers, and common scenarios to tailor its knowledge and adopt the "NapCoach" persona.

2. Project Setup and Prerequisites

2.1. Environment

The project is designed to run in a **Google Colaboratory (Colab)** environment. It is highly recommended to use a **GPU runtime (e.g., T4)** for efficient model loading, the fine-tuning process, and subsequent inference tasks.

2.2. Required Accounts and Tokens

- **Google Account**: Necessary for accessing Google Colab.
- **Hugging Face Account**: Required to download:
    - The pre-trained Large Language Model (mistralai/Mistral-7B-Instruct-v0.2).
    - The STT model components used by faster-whisper.
    - The TTS voice models from rhasspy/piper-voices for piper-tts.
- **Hugging Face User Access Token**: A token from Hugging Face is mandatory for downloading gated models such as mistralai/Mistral-7B-Instruct-v0.2. This token must be stored as a Colab Secret named **HF_TOKEN**. Ensure the token has at least 'read' permissions.

2.3. File System Structure (within Colab Environment)

The notebook creates and utilizes the following directory structure within the Colab session's temporary file system:

- ./piper_models/: Stores the downloaded TTS (Piper) model files (.onnx and .json).
- ./agent_audio_output/: Serves as the output directory for TTS-generated audio files during the text-based interaction loop (Step 11).

- ./sleep_coach_finetuning_data.jsonl: The JSON Lines file containing the instruction-response pairs used for fine-tuning the LLM. This file is generated by Step 8.
- ./sleep-coach-lora-adapter-mistral7b/: The main output directory for the fine-tuning process (Step 9). It contains:
    - Checkpoints saved during training.
    - final_adapter/: Subdirectory holding the final trained LoRA adapter weights and tokenizer configuration.
- ./gradio_tts_audio_responses/: Stores TTS audio files specifically generated for playback within the Gradio interface.
- Temporary audio files (e.g., temp_audio_for_transcription_...) are created in the root Colab directory during STT processing and are subsequently deleted.

3. How to Run the code

1. **Open in Google Colab**: Copy the entire contents of the .py file and paste it in a singular code cell in a Google Colab notebook.
2. **Set Hugging Face Token**:
    - If you haven't already, create a Hugging Face access token with 'read' permissions from https://huggingface.co/settings/tokens.
    - In the Colab notebook, click the "Key" icon (🔑) in the left sidebar.
    - Click "Add a new secret", name the secret "**HF_TOKEN**", and paste your token value into the "Value" field.
    - Ensure the "Notebook access" toggle for this secret is switched ON.
3. **Ensure GPU Runtime**:
    - Navigate to "Runtime" in the Colab menu, then select "Change runtime type".
    - Under "Hardware accelerator", choose a **GPU** (e.g., "T4 GPU"). Click "Save".
4. **Accept Gated Model Terms (Mistral-7B)**:
    - The mistralai/Mistral-7B-Instruct-v0.2 model is gated. You must visit its page on Hugging Face (https://huggingface.co/mistralai/Mistral-7B-Instruct-v0.2) and accept the terms of use while logged into your Hugging Face account. This step is crucial for the model download to succeed.
5. **Click on “Run All” from the Runtime tab.**
6. **Interact with Gradio Interface**:

  - After running, a public URL (typically ending in .gradio.live) will be printed in the output. Click this link to open the NapCoach AI chatbot in a new browser tab.
  - You can type your questions into the text input box or use the audio input options (record via microphone or upload an audio file).
  - The assistant will respond with both text in the chat window and synthesized voice output.
  - The "Clear Coach's Internal Memory" button resets the conversation history for the LLM in the current Gradio session.

4. Key Technologies and Components

- **Programming Language**: Python
- **Environment**: Google Colaboratory (with T4 GPU recommended)
- **LLM (Large Language Model)**:
  - Base Model: mistralai/Mistral-7B-Instruct-v0.2
  - Quantization: Loaded in 4-bit using bitsandbytes (bnb_4bit_quant_type="nf4", bnb_4bit_compute_dtype=torch.bfloat16).
  - Fine-tuning: Parameter-Efficient Fine-Tuning (PEFT) using **LoRA** (Low-Rank Adaptation) from the peft library.
- **STT (Speech-to-Text)**:
  - Model: faster-whisper (defaulting to "base.en" checkpoint for English).
- **TTS (Text-to-Speech)**:
  - Engine: piper-tts
  - Voice Model: rhasspy/piper-voices (defaulting to en_US-lessac-medium).
- **Core Libraries & Frameworks**:
  - transformers: For LLM loading (AutoModelForCausalLM, AutoTokenizer) and training utilities.
  - torch: Primary deep learning framework.
  - accelerate: Simplifies running PyTorch models on various hardware configurations (used for device_map="auto").
  - peft: For implementing LoRA fine-tuning.
  - bitsandbytes: Enables 4-bit quantization of models.

  - datasets: For loading and processing the fine-tuning dataset.
  - gradio: For building and launching the interactive web UI.
  - soundfile: For reading and writing audio files.
  - huggingface_hub: For programmatic interaction with the Hugging Face Hub (downloads, authentication).
  - ipython: Used for IPython.display.Audio for audio playback within the Colab notebook cells (primarily for non-Gradio interaction).
- **Conversation Memory & Prompting**:
  - Managed by a Python list (conversation_history_finetuned) that stores turns of user queries and assistant responses.
  - A **SYSTEM_PROMPT** is used to guide the LLM's persona, behavior, and safety constraints. This prompt is prepended to the conversation.
  - The prompt format is tailored for Mistral Instruct models (<s>[INST] ... [/INST] ... </s>).

5. Observations and Potential Future Enhancements

- **Fine-Tuning Impact**: The fine-tuning process (Step 9) significantly adapts the base Mistral model to the "NapCoach" persona and improves its ability to answer sleep-related queries based on the provided dataset. The quality of responses is directly tied to the comprehensiveness and relevance of the fine-tuning examples.
- **Resource Management in Colab**: The use of 4-bit quantization (bitsandbytes) and LoRA (peft) is essential for fitting and fine-tuning the 7B parameter model within Colab's T4 GPU memory limits. Explicit GPU cache clearing (torch.cuda.empty_cache()) is used after fine-tuning.
- **TTS Output Quality**: piper-tts provides efficient and generally clear speech synthesis. The script includes a regex (re.sub(r'(\d+)-(\d+)', r'\1 to \2', ...)) to improve the pronunciation of numerical ranges (e.g., "7-9 hours" becomes "7 to 9 hours"). Further pre-processing could handle other specific phonetic challenges.
- **STT Performance**: faster-whisper ("base.en") offers a good trade-off between transcription accuracy and speed for English. For noisy environments or more diverse accents, larger Whisper variants could be considered if resources allow.
- **LLM Context Window**: Mistral-7B-Instruct-v0.2 has a specific context window size. Very long conversations might eventually lead to older information being pushed out of context, affecting long-term memory recall. The max_seq_length for fine-tuning also plays a role here.

- **Error Handling**: Basic error handling is implemented for critical operations like model loading. More robust error handling could be added for network issues during downloads or unexpected user inputs in the Gradio interface.
- **Scalability for Production**: The current Colab-based setup is designed for demonstration and development. A production deployment would require more robust infrastructure, such as dedicated model serving solutions (e.g., Text Generation Inference, vLLM), managed databases for user data (if applicable), and a scalable web application framework.
- **Gradio UI Functionality**: The Gradio interface is functional for chat and audio I/O. Enhancements could include displaying conversation history more persistently, user authentication, or visual aids for sleep data if that were integrated.
- **No RAG Implemented**: It's important to note that this system, as per the provided code, is a **fine-tuned LLM** and **not a Retrieval Augmented Generation (RAG) system**. It does not dynamically retrieve information from external vector stores or data collections at inference time. All its specialized knowledge is derived from its pre-training and the fine-tuning dataset. Future work could involve integrating RAG capabilities to pull in real-time or highly specific external information.
- **System Prompt Efficacy**: The SYSTEM_PROMPT is crucial for guiding the LLM's behavior, ensuring safety, and maintaining its persona. Its effectiveness can be further tuned through iteration.