Complainers Я Us — Magical Retellers Peddling Rubies

Plan of Action for Creating a Basic Voice AI Agent

Project Overview:

Objective: Develop a basic voice AI application using open-source tools to meet Level 1 requirements for the Ruby, Magical Toys, and Retell AI tracks. This includes processing the dataset from the Ruby track to generate voice responses.

Tools:

- **Voice-to-Text:** Whisper
- **Text Classification:** Hugging Face Transformers
- **Text-to-Speech (TTS):** Tortoise TTS
- **Language:** Python

Plan of Action:

1. Preparation (30 minutes):

- **Setup Environment:**

Install required libraries:
bash
Copy code

```
pip install git+https://github.com/openai/whisper.git
pip install transformers torch soundfile
# Install Tortoise TTS from the appropriate repository or precompiled binaries
```

    -
    - Prepare the development environment (e.g., virtual environment or Docker container).

2. Download and Process Dataset (1 hour):

- **Download Dataset:**
    - Download the Ruby dataset from Google Drive link.
- **Transform Dataset:**
    - Convert the JSON data into a format that can be used for text-to-speech.

Example code snippet to load and process the dataset:
python
Copy code

```
import json

def load_ruby_data(json_path):
    with open(json_path, 'r') as f:
        data = json.load(f)
    return data

def process_data_for_tts(data):
    # Extract relevant fields for TTS
    texts = [item['text'] for item in data['items']]
    return texts
```

-

3. Implementation (4 hours):

- **Voice-to-Text Conversion (1 hour):**
    - Install and configure Whisper.

Example code snippet for transcribing audio:
python
Copy code

```
import whisper

def load_whisper_model():
    model = whisper.load_model('base')
    return model

def transcribe_audio(model, audio_path):
    result = model.transcribe(audio_path)
    return result['text']
```

    -
- **Text Classification (1.5 hours):**
    - Install and configure Hugging Face Transformers.

Example code snippet for text classification:
python

Copy code

```
from transformers import pipeline

def load_classifier_model():
    classifier = pipeline('text-classification',
model='distilbert-base-uncased')
    return classifier

def classify_text(classifier, text):
    result = classifier(text)
    return result[0]['label']
```

- **Text-to-Speech (TTS) Response (1 hour):**
  - Install and configure Tortoise TTS.

Example code snippet for generating speech:
python
Copy code

```
from tortoise.api import TTS

def load_tts_model():
    tts = TTS('path/to/tortoise/model')  # Update with actual path
    return tts

def synthesize_speech(tts, text, output_path):
    audio = tts.synthesize(text)
    with open(output_path, 'wb') as f:
        f.write(audio)
```

- **Integration (30 minutes):**

Combine Components into a Single Script:
python
Copy code

```
def process_voice_command(audio_file, json_file):
    whisper_model = load_whisper_model()
    classifier = load_classifier_model()
    tts = load_tts_model()
```

```
    # Load and process the dataset
    data = load_ruby_data(json_file)
    texts = process_data_for_tts(data)

    for text in texts:
        # Transcribe audio to text
        transcription = transcribe_audio(whisper_model, audio_file)
        print(f"Transcribed text: {transcription}")

        # Classify text
        classification = classify_text(classifier, transcription)
        print(f"Classification result: {classification}")

        # Generate TTS response
        if classification == 'LABEL_COMPLAINT':  # Adjust based on
model output
            response_text = "We have received your complaint. Thank
you for reaching out."
        else:
            response_text = "Your message has been received."

        synthesize_speech(tts, response_text, 'response.wav')
        print("Response has been synthesized to response.wav")
```

        ○

4. Testing & Refinement (2 hours):

- **Testing (1.5 hours):**
    - Test with various audio inputs and text data.
    - Verify accuracy of transcription, classification, and TTS responses.
- **Refinement (30 minutes):**
    - Adjust models or thresholds to improve performance.

5. Documentation & Submission (1 hour):

- **Documentation (30 minutes):**
    - Write setup and usage instructions.
    - Document the code and any additional setup steps.
- **Demo & Submission (30 minutes):**
    - Record a demo video showing the voice AI agent in action.
    - Upload the project to GitHub with documentation.

- Prepare a Reddit post or other required submission formats.

Complete Python Code Example:

python
Copy code

```
import whisper
from transformers import pipeline
from tortoise.api import TTS
import json

# Load Ruby dataset
def load_ruby_data(json_path):
    with open(json_path, 'r') as f:
        data = json.load(f)
    return data

def process_data_for_tts(data):
    texts = [item['text'] for item in data['items']]
    return texts

# Function to load the Whisper model
def load_whisper_model():
    model = whisper.load_model('base')
    return model

# Function to transcribe audio to text using Whisper
def transcribe_audio(model, audio_path):
    result = model.transcribe(audio_path)
    return result['text']

# Function to load the text classification model
def load_classifier_model():
    classifier = pipeline('text-classification',
model='distilbert-base-uncased')
    return classifier

# Function to classify text
def classify_text(classifier, text):
    result = classifier(text)
```

```
    return result[0]['label']

# Function to load the Tortoise TTS model
def load_tts_model():
    tts = TTS('path/to/tortoise/model')  # Update this path
    return tts

# Function to synthesize speech from text using Tortoise TTS
def synthesize_speech(tts, text, output_path):
    audio = tts.synthesize(text)
    with open(output_path, 'wb') as f:
        f.write(audio)

# Main function to process voice command
def process_voice_command(audio_file, json_file):
    whisper_model = load_whisper_model()
    classifier = load_classifier_model()
    tts = load_tts_model()

    # Load and process the dataset
    data = load_ruby_data(json_file)
    texts = process_data_for_tts(data)

    for text in texts:
        # Transcribe audio to text
        transcription = transcribe_audio(whisper_model, audio_file)
        print(f"Transcribed text: {transcription}")

        # Classify text
        classification = classify_text(classifier, transcription)
        print(f"Classification result: {classification}")

        # Generate TTS response
        if classification == 'LABEL_COMPLAINT':  # Adjust based on
model output
            response_text = "We have received your complaint. Thank
you for reaching out."
        else:
```

```
            response_text = "Your message has been received."

        synthesize_speech(tts, response_text, 'response.wav')
        print("Response has been synthesized to response.wav")

# Example usage
if __name__ == "__main__":
    process_voice_command('input_audio.wav',
'ruby_hackathon_data.json')
```

Explanation of Code:

- **Whisper Model:** Converts voice to text.
- **Text Classification:** Determines if the transcribed text is a complaint.
- **Tortoise TTS:** Converts text responses to speech.
- **Integration:** Combines the three steps into a single process.

Testing & Adjustments:

- Ensure the path to the Tortoise model is correctly set.
- Replace `LABEL_COMPLAINT` with the actual label returned by your classification model.
- Test with valid audio files and the Ruby dataset.

This plan should help you build a functional voice AI agent that processes and transforms the dataset into voice responses, meeting the requirements for all three tracks.


COMPLAINERS R US
MAGICAL RETELLERS
Peddling Rubies
VOICE AI TECHNOLOGY