# Report

## Introduction

This document provides a comprehensive description of an end-to-end (E2E) AI voice assistant system. The system integrates several components to process audio input, generate textual responses, and convert those responses into audio. The implementation leverages state-of-the-art models and libraries for speech recognition, text generation, and text-to-speech synthesis.

## System Components

### 1. Audio Transcription

**Model:** Whisper (by OpenAI)

**Library:** whisper (OpenAI's implementation)

**Purpose:** Convert spoken language into written text.

**Functionality:** The audio transcription process begins with loading and preprocessing the audio file. This involves padding or trimming the audio to ensure it fits a standard length of 30 seconds. Once the audio is prepared, the next step is feature extraction, where the audio is converted into a log-Mel spectrogram. This spectrogram represents the audio's frequency content visually and serves as the input for the transcription model. Finally, the decoding stage utilizes Whisper's specialized decoding options to convert the log-Mel spectrogram into text. This process transforms the visual representation of the audio into readable, transcribed text, providing a foundation for further processing.

```
def translate_audio(audio):
    audio = whisper.load_audio(audio)
    audio = whisper.pad_or_trim(audio)
    mel = whisper.log_mel_spectrogram(audio).to(model.device)
    options = whisper.DecodingOptions(language='en', task="transcribe", temperature=0)
    result = whisper.decode(model, mel, options)
    return result.text
```

## 2. Text Generation

**Model:** Meta LLaMA 3 (Instruct variant)

**Library:** transformers (Hugging Face's implementation)

**Purpose:** Generate a textual response based on the transcribed input.

**Functionality:** It handles the process of generating a textual reply from the AI model where custom templates are used to structure the input and output in a way that aligns with the model's requirements. This step ensures that the input text is presented to the model in a clear and contextually relevant format. Once the prompt is formatted, the function proceeds with generation by making a request to the LLaMA model via the Hugging Face Inference API.

```
def text_response(t):
    time.sleep(1)
    response = Settings.llm.complete(t)
    message = response.text
    return message
```

## 3. Text-to-Speech Synthesis

**Model:** SpeechT5 (by Microsoft) with HiFi-GAN vocoder

**Library:** transformers, datasets, soundfile

**Purpose:** Convert generated text back into spoken language.

**Functionality:** The text-to-speech synthesis process involves two key stages. First, processing utilizes SpeechT5 in combination with HiFi-GAN to convert the input text into spoken audio. SpeechT5 generates the initial speech output, which is then refined by HiFi-GAN to enhance its quality and naturalness. Once the speech synthesis is complete, the final stage is saving, where the generated speech is stored as a WAV file. This saved file contains the synthesized audio, ready for playback or further use.

```
def audio_response(text, output_path="speech.wav"):
    processor = SpeechT5Processor.from_pretrained("microsoft/speecht5_tts")
    model = SpeechT5ForTextToSpeech.from_pretrained("microsoft/speecht5_tts")
    vocoder = SpeechT5HifiGan.from_pretrained("microsoft/speecht5_hifigan")
```

```
    inputs = processor(text=text, return_tensors="pt")
    embeddings_dataset = load_dataset("Matthijs/cmu-arctic-xvectors", split="validation")
    speaker_embeddings = torch.tensor(embeddings_dataset[7306]["xvector"]).unsqueeze(0)
    with torch.no_grad():
        speech = model.generate_speech(inputs["input_ids"], speaker_embeddings,
vocoder=vocoder)
    sf.write(output_path, speech.numpy(), samplerate=16000)
    return output_path
```

**4. User Interface**

**Library:** gradio

**Purpose:** Create a web-based interface for interacting with the voice assistant.

**Functionality:** The Gradio interface setup provides an interactive platform for users to engage with the AI voice assistant. It allows users to record audio directly through their microphone, and once the audio is processed, the interface displays the transcribed text, the response generated by the language model, and the synthesized speech. This setup facilitates a seamless user experience by presenting all relevant outputs in a user-friendly format, enabling easy interaction and feedback from the assistant.

```
output_1 = gr.Textbox(label="Speech to Text")
output_2 = gr.Textbox(label="LLM Output")
output_3 = gr.Audio(label="LLM output to audio")
gr.Interface(
    title='AI Voice Assistant',
    fn=transcribe_,
    inputs=[
        gr.Audio(sources="microphone", type="filepath"),
    ],
    outputs=[
        output_1, output_2, output_3
```

```
    ]
).launch(share=True)
```

```python
import torch
from llama_index.core.prompts import PromptTemplate
from transformers import AutoTokenizer
from llama_index.core import Settings
import os
import time
from llama_index.llms.text_generation_inference import TextGenerationInference
import whisper
import gradio as gr
from gtts import gTTS
from transformers import SpeechT5Processor, SpeechT5ForTextToSpeech, SpeechT5HifiGan
import soundfile as sf
from datasets import load_dataset
model = whisper.load_model("base")
HF_API_TOKEN = os.getenv("HF_TOKEN")

def translate_audio(audio):

    # load audio and pad/trim it to fit 30 seconds
    audio = whisper.load_audio(audio)
    audio = whisper.pad_or_trim(audio)

    # make log-Mel spectrogram and move to the same device as the model
    mel = whisper.log_mel_spectrogram(audio).to(model.device)

    # decode the audio
    options = whisper.DecodingOptions(language='en', task="transcribe", temperature=0)
    result = whisper.decode(model, mel, options)
    return result.text

def audio_response(text, output_path="speech.wav"):
    # Load the processor, model, and vocoder
    processor = SpeechT5Processor.from_pretrained("microsoft/speecht5_tts")
    model = SpeechT5ForTextToSpeech.from_pretrained("microsoft/speecht5_tts")
    vocoder = SpeechT5HifiGan.from_pretrained("microsoft/speecht5_hifigan")

    # Process the input text
    inputs = processor(text=text, return_tensors="pt")

    # Load xvector containing speaker's voice characteristics
    embeddings_dataset = load_dataset("Matthijs/cmu-arctic-xvectors", split="validation")
    speaker_embeddings = torch.tensor(embeddings_dataset[7306]["xvector"]).unsqueeze(0)

    # Generate speech
    with torch.no_grad():
        speech = model.generate_speech(inputs["input_ids"], speaker_embeddings, vocoder=vocoder)

    # Save the audio to a file
    sf.write(output_path, speech.numpy(), samplerate=16000)  # Ensure the sample rate matches your needs

    return output_path

def messages_to_prompt(messages):
    # Default system message for a chatbot
    default_system_prompt = "You are an AI chatbot designed to assist with user queries in a friendly and conversational manner."

    prompt = default_system_prompt + "\n"

    for message in messages:
        if message.role == 'system':
            prompt += f"\n{message.content}</s>\n"
        elif message.role == 'user':
            prompt += f"\n{message.content}</s>\n"
        elif message.role == 'assistant':
            prompt += f"\n{message.content}</s>\n"

    # Ensure we start with a system prompt, insert blank if needed
    if not prompt.startswith("\n"):
        prompt = "\n</s>\n" + prompt

    # Add final assistant prompt
    prompt = prompt + "\n"

    return prompt

def completion_to_prompt(completion):
    return f"<|system|>\n</s>\n<|user|>\n{completion}</s>\n<|assistant|>\n"

Settings.llm = TextGenerationInference(
    model_url="https://api-inference.huggingface.co/models/meta-llama/Meta-Llama-3-8B-Instruct",
    token=HF_API_TOKEN,
    messages_to_prompt=messages_to_prompt,
    completion_to_prompt=completion_to_prompt
)
def text_response(t):
    time.sleep(1)  # Adjust the delay as needed
```

```
    response = Settings.llm.complete(t)
    message = response.text
    return  message

def transcribe_(a):
    t1 = translate_audio(a)
    t2 = text_response(t1)
    t3 = audio_response(t2)
    return (t1, t2, t3)

output_1 = gr.Textbox(label="Speech to Text")
output_2 = gr.Textbox(label="LLM Output")
output_3 = gr.Audio(label="LLM output to audio")

gr.Interface(
    title='AI Voice Assistant',
    fn=transcribe_,
    inputs=[
        gr.Audio(sources="microphone", type="filepath"),
    ],
    outputs=[
        output_1, output_2, output_3
    ]
).launch(share=True)
```