Phase 4-Speech and Text Processing Implementation

1. Introduction

In this phase of our chatbot project, we integrated speech-based functionalities to make the system more interactive and accessible. The features included:
Text-to-Speech (TTS): Converting chatbot responses into natural-sounding audio.
Speech-to-Text (STT): Converting user speech input into text that the chatbot can process.
This document explains the steps, implementation details, and the theoretical background of both functionalities.

2. Text-to-Speech (TTS)

The Text-to-Speech feature was implemented so that the chatbot's textual responses could also be played as audio. This improves accessibility, especially for younger users or those with reading difficulties.

2.1 Theoretical Background

TTS works by converting written text into speech using a synthesis engine. Modern TTS systems rely on neural networks to generate human-like voices. We leveraged browser APIs and JavaScript to achieve this.

2.2 Implementation

We used the Web Speech API's `speechSynthesis` feature in JavaScript. Whenever the chatbot generated a text response, we optionally converted it into audio. We also added a sound button near each message so the user could replay the speech.

Reference Code Snippet:

```
function speakText(text) {
  const utterance = new SpeechSynthesisUtterance(text);
  speechSynthesis.speak(utterance);
}

// Example usage
speakText('Hello, this is the chatbot speaking!');
```

3. Speech-to-Text (STT)

The Speech-to-Text feature was integrated to allow users to interact with the chatbot using their voice. This makes the system more natural and reduces the need for typing.

3.1 Theoretical Background

STT systems use Automatic Speech Recognition (ASR) to transcribe audio into text. ASR involves several steps: audio preprocessing, feature extraction (e.g., MFCCs), acoustic modeling, language modeling, and decoding. In our project, we used the Web Speech API.

3.2 Implementation

We integrated the Web Speech API's `SpeechRecognition` object in JavaScript. When the microphone button was pressed, it started listening for speech and converted it into text, which was then passed to the chatbot backend for processing.

Reference Code Snippet:

```
const recognition = new(window.SpeechRecognition ||
window.webkitSpeechRecognition)();
recognition.lang = 'en-US';
recognition.start();

recognition.onresult = function(event) {
  const transcript = event.results[0][0].transcript;
  console.log('User said:', transcript);
  sendMessage(transcript); // pass to chatbot
};
```

4. Integration with Chatbot

Both TTS and STT were integrated seamlessly with the chatbot interface. Each bot message had a sound button for replaying speech (TTS), and a microphone button was provided for recording user speech (STT). This made the interaction more dynamic and user-friendly.

5. Challenges and Solutions

Background noise: We handled noise by using the browser's built-in filters.
Multiple languages: We ensured that the recognition object supported multiple language codes.
Browser compatibility: We added fallbacks for browsers without Web Speech API support.

6. Conclusion

The addition of Text-to-Speech and Speech-to-Text significantly enhanced our chatbot. Users can now both listen to responses and speak their inputs. This multimodal interaction is closer to human communication, improving engagement and accessibility.