Design Document for Phase-2

1. Introduction

This document outlines the Phase-2 design of the Speech-to-Speech System, focusing on integrating dedicated Python servers for reliable API communication with sandbox services (MT and TTS). These servers manage translation and synthesis through efficient API handling with retries and error control. Additionally, a buffering mechanism ensures smooth playback by maintaining 4-5 audio chunks, while Persistent Messages and Durable queue management protects against data loss and system failures.

2. System Architecture

The system maintains a modular pipeline architecture, with improvements targeting error resilience, buffer management, Persistent data handling and API integration. Message queues are used for inter-process communication, while buffering ensures smooth playback during interruptions.

2.1. Components Overview

1. **ASR_service (Python-Server-1A)**
   a. Reads from ASR_input.
   b. Makes API call to the Automatic speech recognition (ASR) service.
   c. Pushes the recognized text to ASR_output.

2. **MT_service (Python-Server-1B)**
   a. Reads JSON from MT_input.
   b. Makes API call to the Machine Translation (MT) service.
   c. Pushes the translated text to MT_output.

3. **TTS_service (Python-Server-1C)**
   a. Reads JSON from TTS_input.
   b. Makes API call to the Text-to-Speech (TTS) service.
   c. Pushes the synthesized speech to TTS_output.

4. **ASR_MT_bridge(Python-Server-2A)**
   a. Reads JSON from ASR_output.
   b. Processes it and writes JSON to MT_input.

5. **MT_TTS_bridge (Python-Server-2B)**
   a. Reads JSON from MT_output.
   b. Processes it and writes JSON to TTS_input.

6. **TTS_Buffer_bridge (Python-Server-2C)**
   a. Reads JSON from TTS_output.
   b. Processes it and writes the final Audio BLOB to the Buffer.

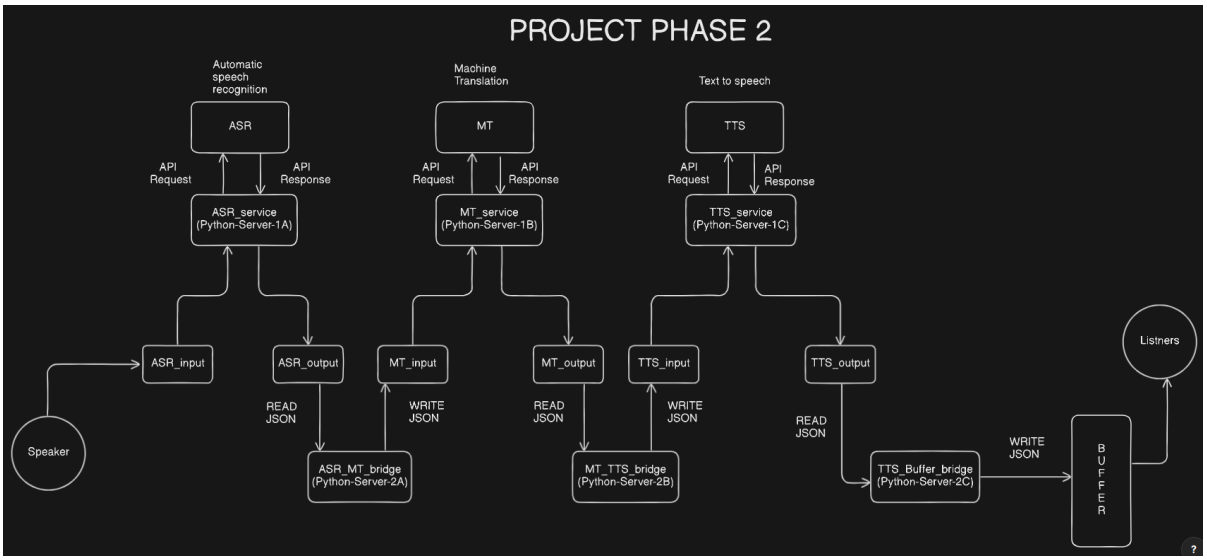
7. **Message Queues (RabbitMQ)**
   a. ASR_inpuy: Stores ASR input.
   b. ASR_output: Stores ASR output.
   c. MT_input: Stores JSON-formatted text for MT processing.
   d. MT_output: Stores translated text from MT.
   e. TTS_input: Stores JSON-formatted text for TTS processing.
   f. TTS_output: Stores synthesized speech for final processing.

8. **Buffer**
   a. Stores processed speech with a rolling buffer of 4-5 chunks.
   b. Ensures smooth playback and recovery during failures.

3. Data Flow (Phase-2)

1. Speaker provides input speech → Audio is pushed to ASR_input for ASR processing.
2. ASR_service (Python-Server-1A) reads from ASR_input, makes an API call to the ASR service, and pushes the recognized text to ASR_output.
3. ASR_MT_bridge (Python-Server-2A) reads JSON from ASR_output, processes it, and writes JSON-formatted text to MT_input for translation.
4. MT_service (Python-Server-1B) reads from MT_input, makes an API call to the MT service, and pushes the translated text to MT_output.
5. MT_TTS_bridge (Python-Server-2B) reads from MT_output, processes it, and writes JSON-formatted text to TTS_input for TTS.

6. TTS_service (Python-Server-1C) reads from TTS_input, makes an API call to the TTS service, and pushes the synthesized speech to TTS_output.
7. TTS_Buffer_bridge (Python-Server-2C) reads from TTS_output, processes the data, and writes the final Audio BLOB to the Buffer.
8. Buffer holds the last 4–5 chunks of audio to ensure smooth playback.
9. Listeners receive the final speech output seamlessly.



4. Error Handling & Logging

- **Invalid Data Handling**: Malformed inputs are logged and pushed to an error queue.
- **RabbitMQ Failure Handling**:
  - Automatic reconnection and queue recreation.
- **API Failure Handling** (in Python-Server-2A and 2B):
  - Retry logic for failed API calls.
  - Timeout handling and error logging.
- **Buffer Overflow/Underflow Handling**:
  - Rolling deletion of old chunks.
  - Pre-buffering to minimize playback gaps.

5. Technology Stack

- **Programming Language**: Python

- **Framework**: FastAPI
- **API management**: RestAPI
- **Message Queue**: CloudAMQP (via Pika)
- **ASR, MT, TTS**: Third-party APIs or custom models
- **Logging**: Python logging for centralized
- **Testing**: Pytest,Unittest

6. Conclusion

Phase-2 enhances the system with robust API handling for sandbox services (ASR, MT and TTS), continuous audio delivery through buffering and reliability even in case of message processing failures. These improvements ensure seamless, real-time speech processing with high system stability and minimal interruptions.