Low-Latency Conversational Voice Assistant with Asynchronous Streaming and User Interruption

Piyush Kumar(23801) **Kritika Gupta(25886)** **Snehal Biswas(23922)** **Ravi Ranjan(23660)**

Generative and Agentic AI in Practice
Indian Institute of Science

Abstract

This project presents a low-latency, real-time conversational voice assistant designed for natural, human-like interactions. The system integrates Voice Activity Detection (VAD) for instant listening, FasterWhisper for efficient speech-to-text, the Gemma-3 (4B) model for fast local language generation, and Coqui-XTTSv2 for high-quality speech synthesis. By running all components offline and enabling token-level streaming, the assistant begins speaking as soon as the first sentence is generated. This design ensures privacy, responsiveness, and smooth conversational flow, achieving a Time-to-First-Sentence-Played (TTFSP) of approximately 2–3 seconds.

1 Introduction

Voice assistants are becoming a part of everyday life, but most of them still feel slow, rigid, or unnatural to talk to. Even small delays in responding can break the flow of a conversation and make the interaction feel mechanical. These delays usually happen because traditional systems work in a strict sequence: record audio, transcribe it, process it through an LLM, and finally generate speech. Each stage adds its own waiting time, and the problem becomes even harder when everything must run offline on limited hardware.

Our goal in this project was to build a voice assistant that feels instant and natural to speak with something that listens the moment the user begins talking and responds while it is still thinking. To achieve this, we designed an on-device system that uses streaming, concurrency, and fast speech detection to reduce latency at every step. By overlapping STT, LLM generation, and TTS, and by allowing the user to interrupt at any time, the assistant aims to create a fluid, human-like conversational experience rather than a turn-by-turn command system.

2 Methodology

We evaluated multiple STT models and selected FasterWhisper for its balance of accuracy and speed. Gemma-3 (4B) was chosen for its efficient token-streaming capabilities, while XTTSv2 provided the best trade-off between synthesis quality and latency. Each component was optimized to operate within a tightly coupled real-time pipeline

3 System Architecture and Data Flow

The system employs a multi-threaded architecture where VAD detects speech boundaries, Faster-Whisper manages transcription, Gemma-3 generates streaming responses, and XTTS synthesizes audio. Asynchronous queues ensure each stage overlaps effectively with the next.

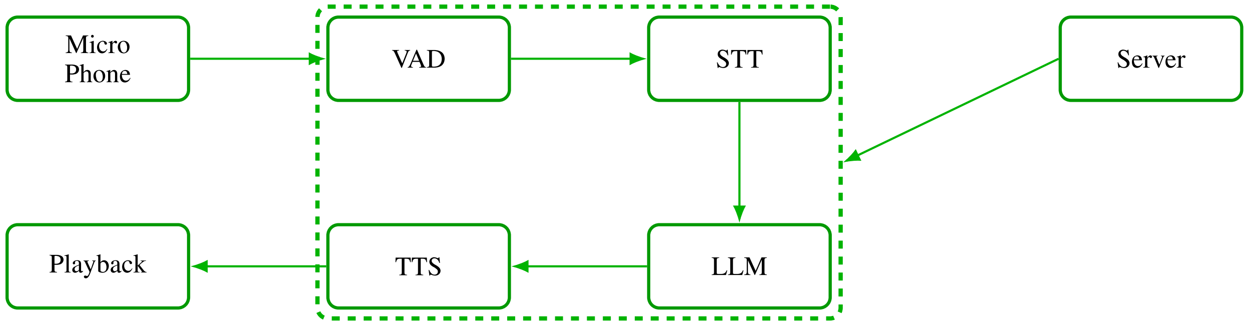

Figure 1: Compact System Architecture with Processing Pipeline Box Connected to Server

3.1 Input Stage: VAD and Utterance Capture

`webrtcvad` (level 2) detects speech boundaries. A vad triggers recording when user starts speaking and stops recording once the user has stopped speaking.

3.2 Processing Stage: STT and LLM Streaming

Audio is normalized and transcribed via FasterWhisper Medium. Transcription is streamed to Gemma 3 (4B) through Ollama API, enabling token-level streaming and immediate TTS synthesis.

4 Concurrent Output Pipeline

Streaming synthesis overlaps LLM generation, TTS, and playback.

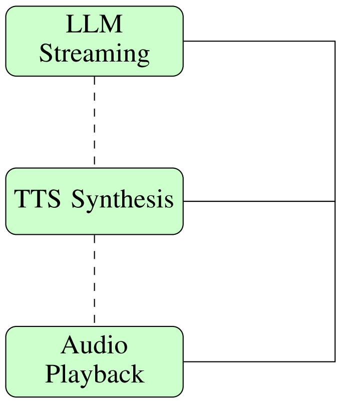

Figure 2: Concurrent Output Pipeline showing overlapping LLM, TTS, and Playback.

4.1 Streaming Synthesis and Segmentation

LLM tokens are buffered in a queue. Sentences are extracted on terminal punctuation and offloaded to TTS in separate threads via `asyncio` library in python, ensuring the main event loop remains free.

4.2 Audio Playback Worker

The text from LLM is put into another queue which is serviced by the TTS model and which is heard by the user in the audio form. This queue works until Ollama generates the response.

5 User Interruption Mechanism

A high-priority thread monitors the microphone during LLM-TTS playback.

5.1 Dedicated VAD Interrupt Thread

Runs independently with lower aggression VAD (level 1) and a 2-3 second ring buffer. Interruption triggers when sufficient user speech is detected.

5.2 Halt Protocol Execution

When speech is detected:

1. Playback terminates.
2. Audio queue is drained non-blockingly.
3. LLM/TTS generation halts immediately interrupt flag.

6 Implementation Tools and Performance

6.1 Software Stack

- STT: FasterWhisper Medium for low latency.
- LLM: Gemma 3 (4B) via Ollama with streaming.
- TTS: Coqui-XTTSv2 in separate thread for high-fidelity synthesis.

6.2 Latency Metrics

Time-to-First-Sentence-Played (TTFSP) measures from user endpoint to first played word:

$$T_{TTFSP} = T_{STT} + T_{LLM-First-Sentence} + T_{TTS-First-Sentence} \tag{1}$$

Concurrent execution pipelines subsequent sentences, achieving TTFSP $\approx$ 2–3 seconds which is mostly GPU based.

6.3 Concurrency Management

`asyncio` handles LLM streaming and audio queue; `threading` handles blocking audio I/O and VAD. `nest_asyncio` ensures proper event loop integration.

7 Conclusion and Future Work

This project demonstrates an offline conversational assistant with near real-time responsiveness. Future improvements include integrating streaming STT, enhancing context awareness for multi-turn conversations, and providing cloud-assisted fallback options for low-resource devices.

8 Individual Member Contribution

Clearly define each team member's expected contributions and responsibilities within the project.

- **Piyush Kumar:** System architecture design, asynchronous pipeline integration, and latency optimization, LLM streaming via Ollama and token-level response management, Coqui-XTTSv2 implementation and Majority part of abstractmaking
- **Kritika Gupta:**FasterWhisper-based STT and VAD fine-tuning, Error handling of vad and pipeline, some part of abstract making.
- **Ravi Ranjan:** Implemented Coqui-XTTSv2, tested and debugged latency metrics, and implemented the server model.
- **Snehal Biswas:** Developed the audio playback module, handled real-time playback, performed performance evaluation, and contributed to creating the server model.

Acknowledgment

We acknowledge FasterWhisper, Coqui-TTS, and Ollama ecosystem models and libraries.