Real-Time On-Device Voice Translator

An Offline Speech-to-Speech Hindi-English Translation System Optimized for Arm CPUs

Ashirbad Sahu Animish Sharma
Divyansh Atri
International Institute of Information Technology Hyderabad, India

1 Introduction

Recent advances in edge Artificial Intelligence have made it possible to deploy large neural models directly on mobile devices without relying on cloud services. However, achieving real-time performance on resource-constrained smartphones remains challenging due to limitations in compute capability, memory bandwidth, and thermal constraints.

This project presents the design and implementation of a fully **on-device, bidirectional voice translator** that performs speech-to-speech translation between English and Hindi entirely offline. The system is built as a Flutter mobile application and optimized specifically for Arm-based processors using NEON SIMD acceleration and quantized neural models.

Unlike cloud-based translators, the proposed system guarantees:

- Complete privacy (no internet dependency)
- Low latency
- Energy-efficient inference
- Reliable operation in offline environments

The application was developed and tested on a **Moto G62 (Snapdragon 695, 6GB RAM)**, which represents a typical mid-range mobile device. Demonstrating acceptable performance on such hardware validates the practicality of edge AI deployment for real-world use.

2 Problem Statement

The objective is to build a real-time speech-to-speech translation system that runs completely on-device and leverages Arm CPU acceleration (NEON/SME2 where available). The system must perform:

1. Speech-to-Text (STT)
2. LLM-based Translation
3. Text-to-Speech (TTS)

while satisfying:

- Near real-time latency
- Efficient quantized inference
- Low memory footprint
- Mobile-friendly power and thermal behavior

3 System Overview

The application follows a three-stage AI pipeline:

$$\text{Speech} \rightarrow \text{Text} \rightarrow \text{Translation} \rightarrow \text{Speech}$$

Each stage runs locally using optimized models. Flutter is used for cross-platform UI, while native Android layers handle high-performance inference through JNI and C++ backends.

4 Application Interface

Figure 1 shows the final mobile interface. The design focuses on simplicity and single-tap interaction for conversational usage.

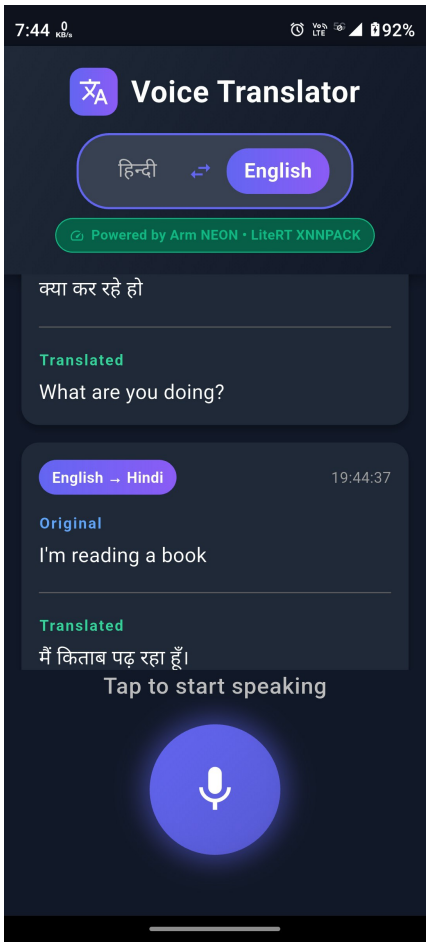

Figure 1: Voice Translator Application UI

5 System Architecture

Figure 2 illustrates the complete data flow of the system. Audio is captured, transcribed using Whisper, translated using Gemma, and synthesized using Android TTS.

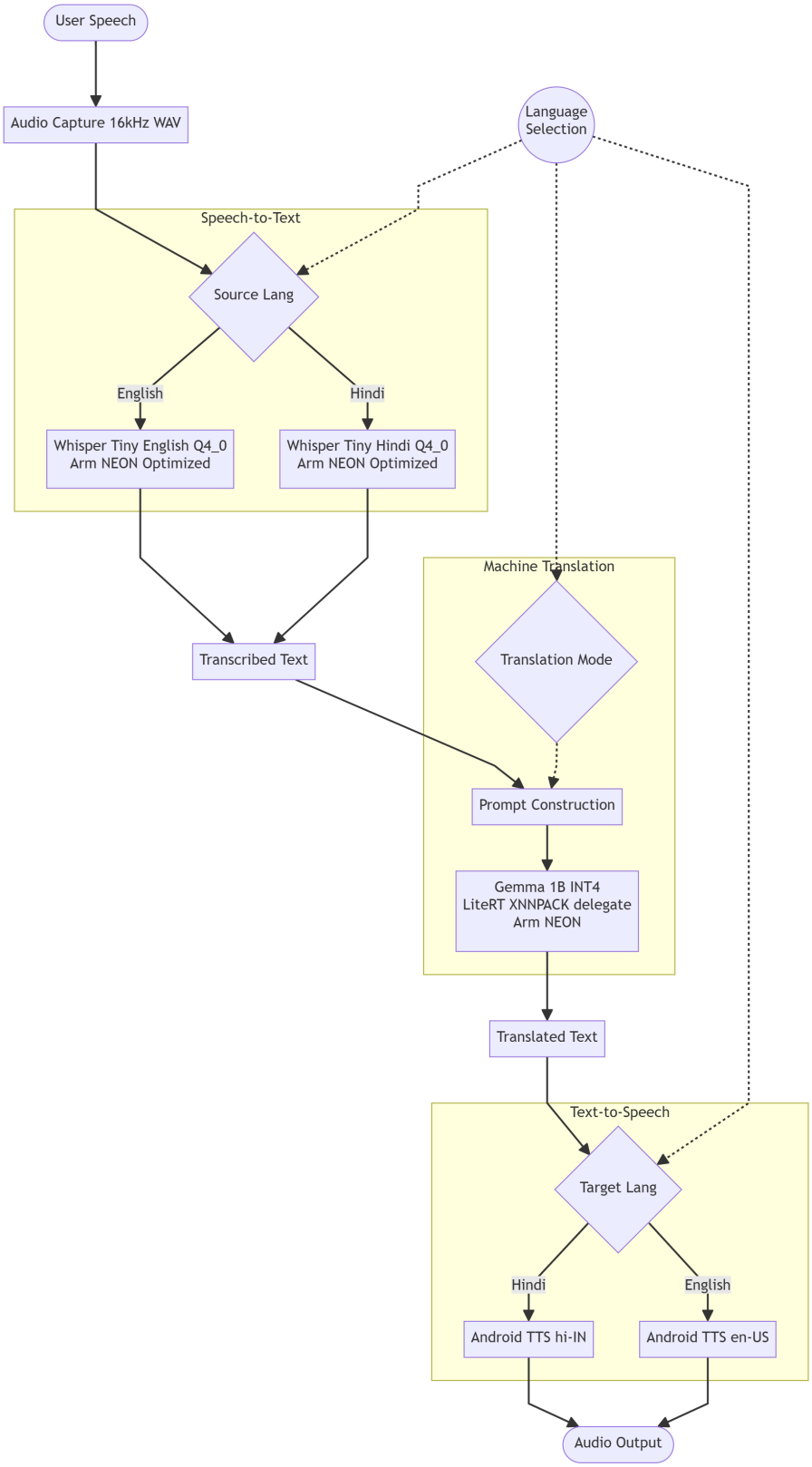

Figure 2: System Architecture and Data Flow

5.1 Pipeline Stages

5.1.1 Audio Capture

- 16 kHz mono WAV recording
- Temporary local storage
- Triggered by microphone button

5.1.2 Speech-to-Text (Whisper Tiny)

- Quantized Q4_0 Whisper Tiny model
- Implemented using `whisper.cpp`
- Compiled with Arm NEON optimizations
- Provides fast on-device transcription

5.1.3 Machine Translation (Gemma 1B INT4)

- 1B parameter LLM
- INT4 quantization to reduce memory bandwidth
- LiteRT + XNNPACK delegate
- Uses Arm NEON SIMD instructions
- Generates translated text

5.1.4 Text-to-Speech

- Native Android TTS engine
- Hindi (hi-IN) and English (en-US)
- Real-time audio playback

6 Framework and Model Selection

6.1 Framework Choice: Flutter

Flutter was chosen as the primary development framework for several key reasons:

- **Cross-Platform Native Performance:** It compiles to native Arm code (AOT compilation) ensuring that the UI thread runs at 60 FPS without the overhead of a JavaScript bridge.
- **Seamless Native Integration:** Using Platform Channels (MethodChannels), Flutter easily delegates heavy computational tasks (like the `whisper.cpp` inference and `MediaPipe LiteRT` bindings for Gemma) directly to the native Android C++/Kotlin layers, combining beautiful UI with C++ performance.

6.2 Model Choices

6.2.1 Speech-to-Text: Whisper Tiny

OpenAI's Whisper Tiny was selected for audio transcription because:

- **Size vs. Accuracy Trade-off:** At roughly 39 million parameters, it is small enough (∼70MB quantized) to fit in mobile RAM while providing sufficient accuracy for clear dictation.
- **C++ Ecosystem:** The availability of `whisper.cpp` allows for dependency-free, highly optimized C++ inference directly on the CPU without relying on heavy Python runtimes.

6.2.2 Machine Translation: Gemma 1B

Google's Gemma 1B (Instruct) model was chosen because:

- **Edge Readiness:** Released specifically with edge deployment in mind, the 1B parameter size hits the sweet spot for modern mobile CPUs.
- **Instruction Following:** The instruction-tuned variant explicitly follows the strict system prompts required to output *only* the translation without conversational filler.

6.3 Why Quantization?

Memory bandwidth is the primary bottleneck for mobile inference. INT4/Q4_0 quantization reduces:

- Model size
- RAM usage
- Cache misses
- Power consumption

This enables LLM execution on mid-range CPUs without GPUs or NPUs.

6.4 Arm Optimization

Both Whisper and Gemma utilize:

- Arm NEON SIMD instructions
- XNNPACK optimized kernels
- Parallel matrix multiplication

These improvements significantly accelerate inference on Arm v8 processors.

7 Benchmarking Methodology

The system includes a built-in benchmarking suite that records:

- STT latency
- Translation latency
- TTS latency
- End-to-end latency
- Tokens per second (TPS)
- CPU utilization
- Memory usage

Metrics are logged via Logcat and processed using an external Python script for aggregation and visualization.

8 Performance Results

8.1 Latency Breakdown

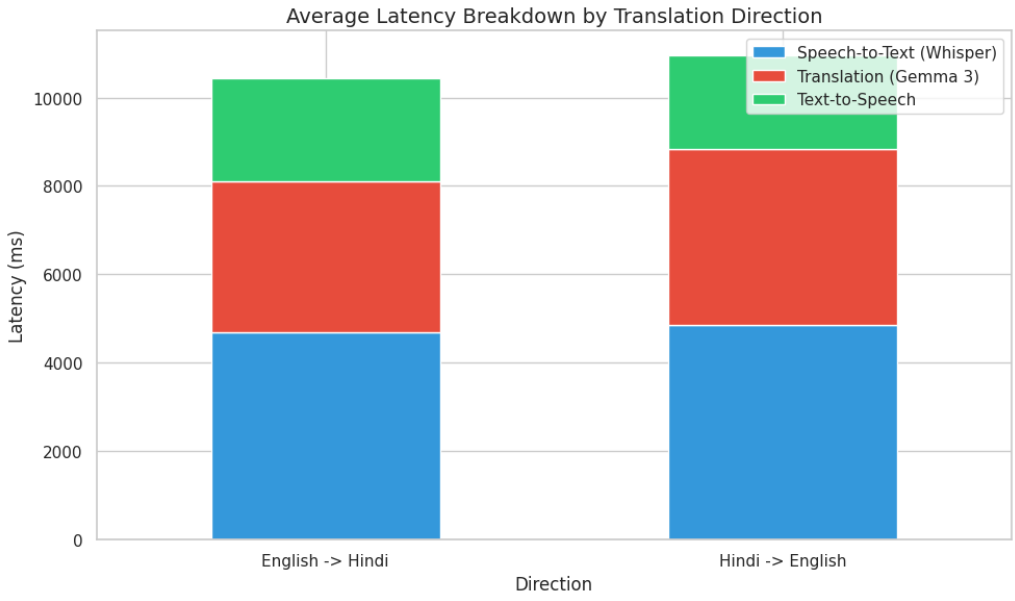

Figure 3: Average Latency Breakdown by Translation Direction

The Whisper STT stage dominates the pipeline, accounting for approximately 45% of the total latency. Overall turnaround time remains between 10–11 seconds.

8.2 Resource Utilization

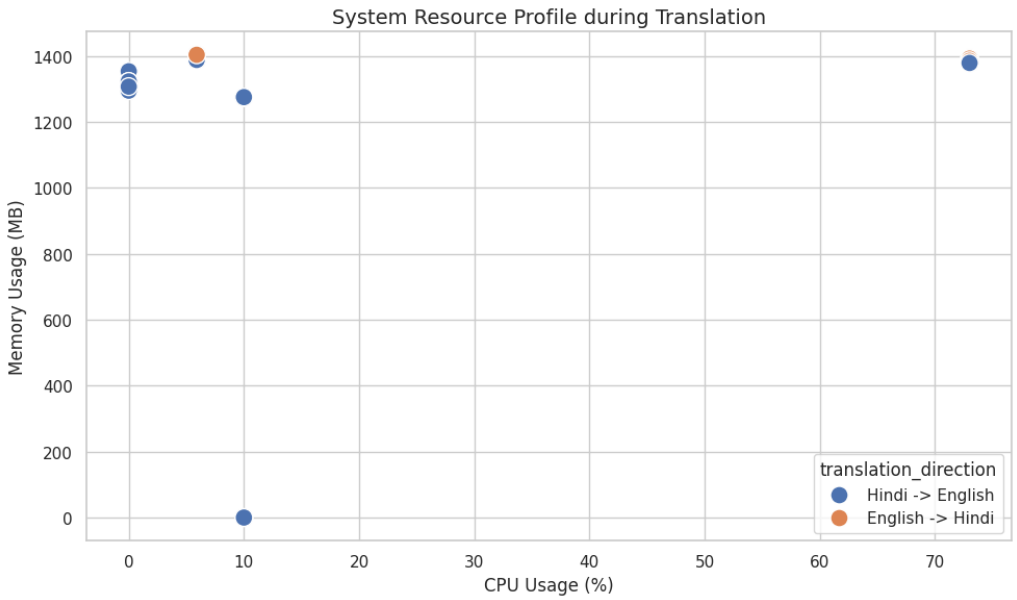

Figure 4: CPU and Memory Usage During Translation

Memory remains stable between 1138–1227 MB, demonstrating that the application operates reliably without memory pressure or crashes.

8.3 LLM Generation Speed

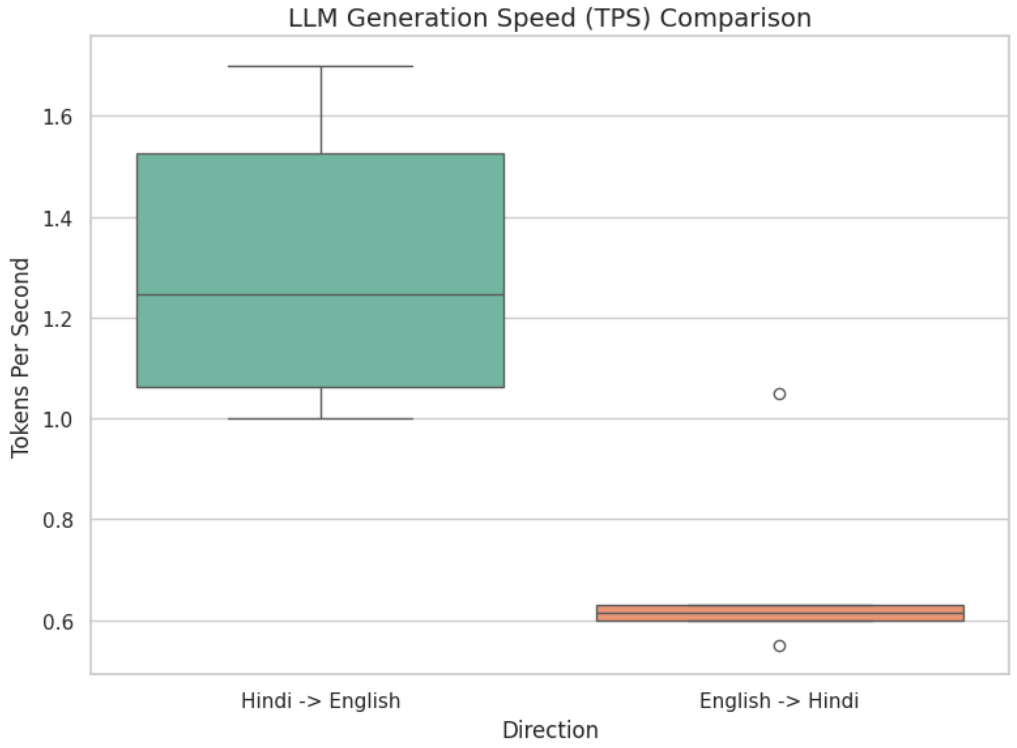

Figure 5: Gemma Tokens Per Second Comparison

Hindi-to-English translation achieves higher throughput:

$$1.29 \text{ TPS} \quad \text{vs} \quad 0.67 \text{ TPS}$$

English token decoding requires lower CPU utilization, indicating better computational efficiency for English generation.

9 Discussion

The results highlight several important observations:

- STT is the largest latency contributor
- LLM translation is feasible on CPU-only devices
- Quantization enables large models within limited RAM
- CPU utilization remains within acceptable mobile limits
- Stable memory footprint ensures reliability

Despite running three neural models simultaneously, the system maintains consistent performance on a mid-range smartphone from 2022. This demonstrates that modern edge devices are capable of handling real-time AI workloads when carefully optimized.

10 Novelty and Contributions

The key contributions of this work include:

- Fully offline speech-to-speech translation
- CPU-only LLM inference on mobile
- Arm NEON accelerated pipeline
- Integrated real-time benchmarking suite
- Practical Flutter-based deployment

Most existing solutions rely on cloud APIs. In contrast, this project demonstrates a complete local alternative suitable for privacy-sensitive and low-connectivity environments.

11 Future Work

Future improvements include:

- Streaming Whisper transcription
- Faster distilled STT models
- Fully local neural TTS
- Additional Indian language support
- Dynamic model download management

12 Conclusion

This project successfully demonstrates that real-time speech-to-speech translation can be achieved entirely on-device using optimized neural networks and Arm CPU acceleration. Even under the constraints of a mid-range smartphone, the system delivers practical latency, stable memory usage, and reliable offline performance.

The work validates the feasibility of deploying compact LLM-based applications at the edge and highlights the effectiveness of quantization and NEON optimizations for mobile AI systems.

In summary, the application shows that efficient on-device translation is not only possible but practical, paving the way for privacy-preserving, low-cost AI assistants on everyday smartphones.