Project Documentation

Offline Hindi Voice Assistant on Raspberry Pi 4 REPORT

Optimized ASR–NLP–TTS Pipeline with Threaded Architecture and INT8 Inference

College Name:

Shri Ramdeobaba University ,Nagpur

Team Members:

- **Taran Wadhawan**
- **Anurag Deshmukh**
- **Mahi Wegad**

1. Project Overview

1.1 Objective

To design and deploy a **low-latency, offline Hindi voice assistant** capable of:

- Wake word detection
- Hindi & English speech recognition
- Intent classification
- Command execution
- Speech synthesis (TTS)
- Running efficiently on Raspberry Pi 4 (ARM64)

The system is optimized for:

- Low computational power
- Minimal latency
- High NEON SIMD utilization
- Fully offline execution

---

2. System Architecture

The assistant follows a **streaming pipeline architecture**:

Microphone
↓
Audio Capture Thread
↓
Audio Queue (Ring Buffer)
↓
Streaming ASR Engine (Vosk / Coqui / wav2vec2)
↓
Wake Word Detection
↓
Intent Classification (SVM / ML Model)
↓
Intent Handler
↓
TTS Engine
↓
Speaker Output

2.1 High-Level Architectural Blocks

1. Audio Input Layer

- PyAudio streaming
- 16kHz mono
- 20ms frames (320 samples)
- Non-blocking buffered capture

2. ASR Layer

Supports multiple engines:

Model	**Type**	**Language**	**Use Case**
Vosk Small Hindi	Kaldi-based	Hindi	Real-time streaming
Coqui STT	DeepSpeech variant	Hindi/English	Lightweight inference
Fine-tuned wav2vec2	Transformer	Hindi	High accuracy
Whisper (OpenAI)	Transformer	Multi-language	High accuracy, heavy
Vosk English	Kaldi	English	Mixed-language support

Primary deployment: **Vosk Hindi Small Model** for real-time embedded performance.

---

3. NLP Layer (Intent Classification)

- SVM classifier
- TF-IDF vectorization
- Multi-class classification
- Confidence threshold filtering
- Hindi + English mixed token support

---

4. TTS Layer

Supported engines:

Engine	**Type**	**Notes**
eSpeak-NG	Formant-based	Lightweight
Festival	Unit selection	Moderate quality
Veena TTS	Neural Hindi TTS	Natural output

Primary deployment: **eSpeak-NG for low compute**, Veena for high-quality mode.

---

5. State Machine

The assistant operates in 2 states:

IDLE → ACTIVE → IDLE

- IDLE: Wake word monitoring
- ACTIVE: Command processing
- Timeout auto-return to IDLE

---

3. Pipeline Design

3.1 Streaming ASR Pipeline

Step 1: Continuous Audio Capture

A dedicated thread captures audio frames:

```
while running:
    frame = stream.read(320)
    audio_queue.put(frame)
```

This prevents blocking the ASR thread.

Step 2: Streaming Recognition

Vosk streaming decoder:

```
if recognizer.AcceptWaveform(audio):
    result = recognizer.Result()
```

Only final results are processed to prevent repetition.

Step 3: Wake Word Detection

Simple text match:

"assistant"
"hello"
"नमस्ते"

Lightweight detection avoids heavy neural wake models.

Step 4: Intent Classification

Pipeline:

Text → TF-IDF → SVM → Decision Function → Confidence Filter

Mathematically:

For SVM:

$$f(x) = w^T x + b$$

Decision margin:

$$confidence = \max(f(x))$$

Low-confidence (< threshold) rejected.

Step 5: Intent Execution

Mapped to handler functions:

- Time query
- Calculation
- Music control
- System info
- Sleep command

Step 6: TTS Output

Text → TTS engine → Audio output

---

4. Threading & Multiprocessing Architecture

4.1 Thread Design

The assistant uses multithreading:

Thread	Purpose
Audio Thread	Continuous capture
Listener Thread	ASR + NLP processing
TTS Thread	Non-blocking speech output

This avoids:

- Audio drop
- Blocking I/O
- Latency accumulation

4.2 Why Not Full Multiprocessing?

Raspberry Pi 4 has limited:

- 4 ARM Cortex-A72 cores
- 4GB RAM

Multiprocessing would:

- Increase memory footprint
- Duplicate model loading

5. Optimization for Raspberry Pi 4

5.1 Hardware Specs

- CPU: ARM Cortex-A72
- Architecture: ARM64
- NEON SIMD support
- 4 cores @ 1.5 GHz
- 4GB RAM

5.2 INT8 Quantized TensorFlow Lite

For neural models:

- Post-training quantization
- 8-bit weights
- Reduced memory by ~75%
- Faster inference

5.3 Arm Compute Library (KleidiAI Backend)

Deployment configuration:

- arch = arm64-v8a
- neon = 1
- build = native
- OpenCL disabled
- num_threads = 4

- CPU governor = performance

Benefits:

- Optimized NEON vectorization
- Better GEMM acceleration
- Lower latency matrix multiplication

5.4 Warm-Up Inference

Before real use:

Run 5 dummy inferences

Purpose:

- Cache loading
- Memory page allocation
- Kernel initialization

Reduces first-response latency.

---

6. Challenges in Hindi ASR

Hindi presents specific challenges:

6.1 Phonetic Complexity

Hindi has:

- Aspirated consonants
- Retroflex sounds
- Nasalization
- Compound words

Example:

"भाषा" vs "बासा"

Acoustic similarity increases WER.

6.2 Code-Switching

Users mix:

Hindi + English

Example:

"music band karo"

ASR must support multilingual decoding.

6.3 Limited High-Quality Datasets

Compared to English:

- Smaller corpus
- Fewer dialect representations
- Accent variability

6.4 Morphological Richness

Hindi words inflect heavily:

"खाना", "खाओ", "खा लिया"

Intent models must generalize semantic variants.

---

7. Challenges in Hindi TTS

7.1 Prosody Modeling

Hindi requires:

- Proper stress placement
- Sentence-level intonation
- Natural vowel length

Formant-based TTS struggles here.

7.2 Neural TTS on Edge Devices

High-quality TTS models:

- Require >500MB memory
- Heavy transformer blocks
- Not feasible without quantization

---

8. Intent Recognition Challenges

8.1 Short Utterances

Example:

"समय?"

Low lexical information → classification unstable.

8.2 Noise Sensitivity

Small ASR errors propagate:

"समय क्या है"
→ "समय कहा है"

Intent misclassification possible.

8.3 Threshold Selection

Too low → false triggers
Too high → missed commands

Balance required via validation.

---

9. Performance Metrics

Measured on Raspberry Pi 4:

Metric	**Value**
ASR Latency	0.7–1.3 sec
CPU Usage	45–70%
RAM Usage	~600MB
Wake Detection Delay	< 500ms
Intent Accuracy	~90% (controlled dataset)
TTS Response Time	300–800ms

9.1 Word Error Rate (WER)

$$WER = \frac{S + D + I}{N}$$

Where:

- S = substitutions
- D = deletions
- I = insertions
- N = total words

Hindi small model WER ≈ 18–22% (clean audio)

10. Safety & Reliability Measures

- Confidence threshold filtering
- Debounce logic for repeated transcripts
- Recognizer reset after each utterance
- Thread-safe audio queue
- Timeout-based auto sleep

---

11. Future Improvements

- VAD-based speech segmentation
- Transformer intent model (quantized)
- Edge TPU acceleration
- Custom Hindi wake word CNN
- On-device continual learning

---

12. Conclusion

This project demonstrates:

- Efficient embedded AI deployment
- Real-time Hindi ASR on ARM
- Threaded streaming architecture
- Quantized inference for edge devices
- Practical NLP intent system

It bridges:

Speech Processing
Embedded Systems
Machine Learning
Edge Optimization Under constrained hardware.