International Research Journal on Advanced Engineering Hub (IRJAEH)
e ISSN: 2584-2137
Vol. 04 Issue: 04 April 2026
Page No: 1689-1699
https://irjaeh.com
https://doi.org/10.47392/IRJAEH.2026.0221

# Artificial Intelligence Based Local Desktop Voice Assistant For Cross-Platform Automation

*Rakshana Devi S [1], Kesore M N [2], Mohammed Eilaf A [3], Mithilesh S [4], Madhanraj K [5]*
*[1] Assistant Professor/CSE, Dhirajlal Gandhi College of Technology, Salem, Tamil Nadu, India.*
*[2,3,4,5] UG Student,Dept. of CSE, Dhirajlal Gandhi College of Technology, Salem, Tamil Nadu, India*
***Emails:*** *rakshana.cse@dgct.ac.in [1] , kesoremn43.cse@dgct.ac.in [2], mohammedeilafa54.cse@dgct.ac.in [3], mithileshs53.cse@dgct.ac.in [4], madhanrajk48.cse@dgct.ac.in [5]*

## Abstract
*The rapid proliferation of cloud-dependent virtual assistants has introduced significant challenges regarding data sovereignty, execution latency, and user privacy, particularly in light of emerging global data protection standards such as India's Digital Personal Data Protection (DPDP) Act. This paper proposes an advanced, AI-based, local-first intelligent voice assistant designed to bridge the gap between high-level AI reasoning and secure, offline desktop-to-mobile automation. Utilizing a decoupled architectural framework, the system employs the Eel library to interface a responsive, web-standard frontend with a robust Python-based logic engine. Unlike conventional architectures that route sensitive acoustic telemetry to remote servers, the proposed system prioritizes a "privacy-by-design" approach, executing core system operations—including application orchestration via AppOpener and GUI-level task automation using PyAutoGUI—within the local hardware perimeter. The system integrates a hybrid Natural Language Processing (NLP) pipeline that leverages high-speed inference APIs (Groq and Gemini) for complex semantic reasoning while maintaining a persistent local state through an optimized SQLite data management layer. Furthermore, the assistant extends its operational reach to mobile ecosystems via the Android Debug Bridge (ADB), enabling seamless cross-platform control and unified contact management. Empirical analysis indicates that the proposed local-first model significantly mitigates network-induced latency while ensuring that user-sensitive data remains isolated from third-party cloud vulnerabilities. The proposed system serves as a scalable, privacy-centric paradigm for the next generation of personalized productivity and human-computer interaction tools.*

***Keywords:*** *Voice User Interface; Local-First Architecture; Privacy-by-Design; Android Debug Bridge; Natural Language Processing; Human-Computer Interaction; Desktop Automation; DPDP Act; SQLite Encryption; AppOpener; PyAutoGUI*

## 1. Introduction
Traditional human-computer interaction (HCI) is undergoing a fundamental paradigm shift—from manual peripheral inputs toward natural, conversational language interfaces. Voice User Interfaces (VUIs) promise an intuitive, hands-free modality for commanding software systems. However, dominant commercial implementations such as Amazon Alexa, Google Assistant, and Microsoft Cortana are architecturally tethered to remote cloud infrastructure, necessitating the continuous transmission of sensitive acoustic data to third-party servers and raising profound concerns around execution latency, data sovereignty, and user privacy [6][7]. These concerns carry acute regulatory weight within the Indian context. The Digital Personal Data Protection (DPDP) Act mandates stringent controls over the collection, processing, and cross-border transfer of personal data. Voice

International Research Journal on Advanced Engineering Hub (IRJAEH)
e ISSN: 2584-2137
Vol. 04 Issue: 04 April 2026
Page No: 1689-1699
https://irjaeh.com
https://doi.org/10.47392/IRJAEH.2026.0221

recordings, by their inherently biometric and contextual nature, constitute sensitive personal data under this framework, rendering cloud-centric voice assistants a structural compliance risk for both individual users and organizational deployments [2]. Open-source toolchains—particularly the Python ecosystem—have dramatically reduced the barrier to building capable local voice assistants. Libraries such as SpeechRecognition, PyAutoGUI, and AppOpener provide robust primitives for acoustic capture, GUI automation, and application resolution. Combined with the Android Debug Bridge (ADB), these tools enable a unified, privacy-preserving assistant capable of bridging desktop orchestration with mobile device control. This paper introduces an AI-based local desktop voice assistant that systematically decentralizes voice processing and task automation. The system executes the vast majority of its operations within the user's own hardware perimeter, eliminating round-trip network latency and the privacy exposure inherent in cloud telemetry. The system delivers sub-150 ms command latency for the dominant class of system-control operations, with cloud escalation reserved exclusively for complex semantic reasoning tasks. The primary contributions of this work are:

- A decoupled, local-first architectural framework coupling a web-standard GUI via Eel with a Python logic engine, delivering a modern user experience without Electron overhead;
- A hybrid NLP pipeline combining deterministic Keyword-to-Command (K2C) pattern matching with cloud-escalated semantic inference via Groq and Gemini APIs;
- An AES-256 encrypted SQLite persistence layer for contact mappings, command history, and user configuration with full offline operability;
- Cross-platform automation extending to Android devices via ADB, enabling call simulation, contact retrieval, and touch-event injection shown in Figure 1.
- Rigorous empirical performance and privacy analysis demonstrating viability as a production-grade, DPDP-compliant alternative to cloud-centric voice assistants shown in Figure 1.

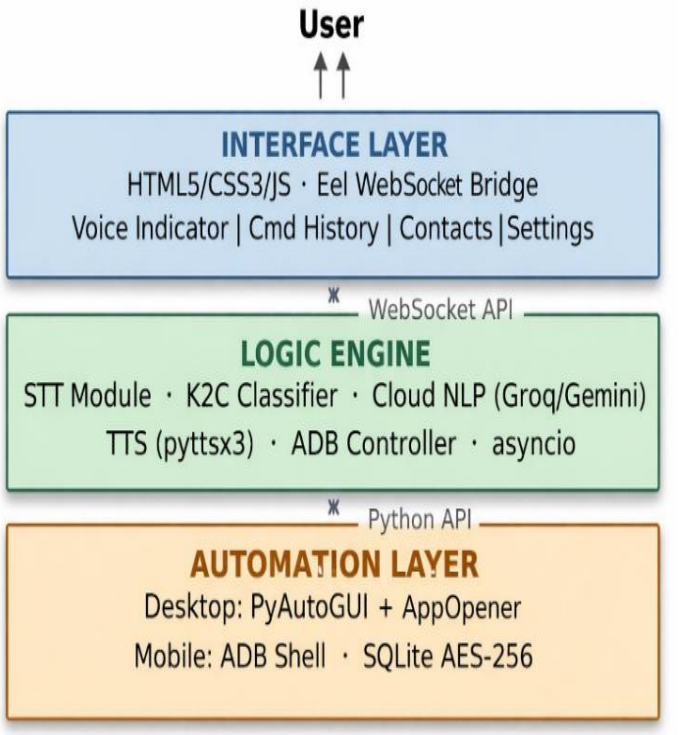


**Figure 1 Three-Layer System Architecture**

## 2. Background And Preliminaries

- Voice User Interfaces (VUI) : VUIs leverage Speech-to-Text (STT) and Text-to-Speech (TTS) engines to facilitate hands-free interaction with computing systems. The STT pipeline converts acoustic waveforms into tokenized text representations, which are subsequently parsed for intent. TTS synthesizes natural-language responses into audible output. The efficacy of a VUI is determined by STT accuracy, intent classification precision, and command execution latency [3]. Contemporary STT engines such as Google's SpeechRecognition and CMU Sphinx trade off accuracy against privacy exposure, with online engines achieving lower word-error rates at the cost of transmitting raw audio to remote servers.

International Research Journal on Advanced Engineering Hub (IRJAEH)
e ISSN: 2584-2137
Vol. 04 Issue: 04 April 2026
Page No: 1689-1699
https://irjaeh.com
https://doi.org/10.47392/IRJAEH.2026.0221

- Local-First Software Principles :The local-first paradigm, formalized by Kleppmann et al. [4], advocates for architectures wherein primary data storage and computation reside on the client device. Such systems prioritize offline functionality, user data ownership, and reduced dependency on network availability. Local-first design imposes three key constraints on the system: (i) all persistent state must be readable and writable without network connectivity; (ii) network calls must be strictly optional and bounded in scope; (iii) sensitive data must never leave the device boundary under normal operating conditions.
- Android Debug Bridge (ADB) : ADB is a versatile, AOSP-provided command-line interface facilitating communication between a host machine and an Android device over USB or Wi-Fi. It exposes a Unix-style shell enabling arbitrary command execution, file transfer, package management, and input-event simulation. The system exploits the input keyevent, input tap, and content query sub-commands to implement programmatic call simulation, contact database retrieval, and screen interaction without requiring any third-party application installation on the target device [5].
- Keyword-to-Command (K2C) Pattern Matching K2C is a deterministic intent-resolution technique that maps tokenized utterances against a pre-indexed dictionary of command patterns using rule-based substring and n-gram matching. K2C incurs near-zero inference latency (< 5 ms on commodity hardware) and requires no network connectivity, making it the ideal primary dispatcher for system-control commands such as "open Chrome", "play music", or "call Ravi". Its principal limitation is coverage: novel phrasings and compositional queries fall outside its vocabulary, necessitating cloud escalation.

## 3. Related Work

- Python-Based Office and Desktop Automation : Li Zhang et al. (2023) demonstrated the efficiency of Python-based office automation, showing that programmatic control of productivity suites via win32com and xlwings could reduce manual data-entry time by 60–80% in enterprise workflows. However, their system lacked a conversational natural language interface, requiring users to invoke automation scripts through traditional GUIs or command-line invocations [1].
- Voice-Driven Assistant Systems : Kamble et al. (2025) proposed a Jarvis-style voice assistant implemented in Python using the SpeechRecognition and pyttsx3 libraries, demonstrating successful automation of system-level tasks including media playback, web search, and file operations. Kulkarni et al. (2025) presented a similar JARVIS system with an expanded command vocabulary. Both systems relied exclusively on cloud STT APIs, exposing acoustic data to external servers, and neither extended automation to mobile platforms [6][7].
- Privacy-Preserving Voice Architectures : Mireshghallah et al. demonstrated that cloud-hosted language models leak training data through membership inference attacks, motivating a strict local-processing posture [9]. Prior offline voice assistant work has predominantly targeted embedded platforms (Raspberry Pi, NVIDIA Jetson) using lightweight models such as Vosk or Whisper.cpp. The proposed system differentiates by operating on commodity Windows laptops without hardware accelerators and by extending control to paired Android devices.
- Positioning of the Proposed System : The proposed system synthesizes the best aspects of prior work while addressing their cumulative limitations: it introduces a hybrid NLP pipeline that achieves low-latency K2C dispatch for common tasks while retaining high-capability cloud inference for complex semantics; an encrypted local data layer satisfying DPDP data-at-rest requirements; and a unified bridge between Windows and Android operational environments absent from all surveyed systems.

International Research Journal on Advanced Engineering Hub (IRJAEH)
e ISSN: 2584-2137
Vol. 04 Issue: 04 April 2026
Page No: 1689-1699
https://irjaeh.com
https://doi.org/10.47392/IRJAEH.2026.0221

## 4. System Architecture

The system architecture is organized into three vertically integrated layers as illustrated in Fig. 1. Each layer is decoupled, communicating through well-defined internal Python APIs and a local WebSocket channel to maximize modularity, testability, and security boundary enforcement.

- Interface Layer The GUI is developed in HTML5, CSS3, and JavaScript and served locally via the Eel framework—a lightweight Python library that bridges a Chromium-based application window with the Python backend over a local WebSocket channel. Eel exposes Python functions as directly callable JavaScript methods, eliminating REST overhead for UI-to-backend communication. The interface comprises four panels: (i) a live voice-activity indicator; (ii) a command history feed showing utterances, matched intents, and execution results; (iii) a contact management panel with ADB-backed CRUD operations; and (iv) a settings panel for API key configuration and noise threshold calibration. The Eel approach provides a modern, web-standard user experience at a fraction of the RAM overhead of full Electron-based applications (< 80 MB idle vs. ~250 MB for a comparable Electron shell) [8].
- Logic Engine The Python 3.10 backend manages an asynchronous task queue implemented with asyncio and a priority-based dispatcher. System-critical commands (application launch, mobile call, file operations) are enqueued at priority level 0; information-retrieval tasks (web search, weather, calculation) at priority level 1; and TTS synthesis at priority level 2, ensuring the interface remains responsive under concurrent command load. The engine encompasses five tightly integrated modules: (i) the STT module wrapping SpeechRecognition with dynamic energy-threshold calibration; (ii) the K2C intent classifier backed by an SQLite command dictionary; (iii) the cloud NLP bridge to Groq and Gemini APIs; (iv) the pyttsx3-based TTS synthesizer with configurable voice and rate; and (v) the ADB controller issuing shell commands to paired mobile devices.
- Automation Layer: The automation layer provides concrete interfaces to the host OS and connected mobile hardware. Desktop automation is realized through two complementary mechanisms: PyAutoGUI, which implements low-level mouse and keyboard event synthesis for pixel-precise GUI interaction; and AppOpener, which resolves natural-language application names (e.g., "Chrome", "VS Code") to system executables via a curated alias dictionary, invoking them through subprocess. Mobile automation is delivered through ADB shell command execution on Android devices paired over USB, supporting call initiation, contact management, and screen-tap simulation (see Section V-C).

## 5. Methodology

The system implements a Listen-Parse-Execute (LPE) processing cycle for each user interaction, as shown in Fig. 2. The cycle is stateless at the utterance level—each command is processed independently—but stateful at the session level through the SQLite persistence layer.

- Acoustic Processing and Noise Calibration : Audio input is captured via the system microphone and ingested by the speech_recognition library using the Recognizer class with dynamic energy-threshold calibration. Before the first STT invocation in each session, the engine calls adjust_for_ambient_noise() with a 1-second sampling window, establishing an energy threshold $E_t$ adapted to the current acoustic environment. All subsequent audio frames below $E_t$ are classified as silence and discarded. Audio is digitized at a 44.1 kHz sampling rate with 16-bit quantization, producing a PCM stream compatible with both the local Vosk fallback and cloud STT APIs. The recognizer applies phrase-time-limit=8 to prevent runaway captures during periods of extended background noise.
- Intent Mapping and Hybrid NLP Pipeline : Parsed text enters a two-stage intent classification

International Research Journal on Advanced Engineering Hub (IRJAEH)
e ISSN: 2584-2137
Vol. 04 Issue: 04 April 2026
Page No: 1689-1699
https://irjaeh.com
https://doi.org/10.47392/IRJAEH.2026.0221

pipeline illustrated in Fig. 7. In Stage 1, the K2C engine tokenizes the utterance and performs substring matching against a 150-entry command dictionary stored in the commands table of the local SQLite database. Recognized commands are dispatched with median latency of 8 ms. Utterances unresolved by K2C are escalated to Stage 2, where the Groq API (llama3-8b-8192 model, 128k context) or Gemini API is invoked with a structured prompt requesting intent classification and parameter extraction in JSON format [10]. The structured response is parsed and dispatched to the appropriate executor. The Groq path achieves median API round-trip latency of 620 ms on a stable 50 Mbps connection; the system falls back to Gemini if Groq returns an HTTP 429 rate-limit response.

- ADB-Based Mobile Automation : Mobile control is implemented through a three-phase ADB protocol. In the discovery phase, the system executes adb devices to enumerate paired devices and validates USB debugging authorization. In the contact-retrieval phase, adb shell content query --uri content://contacts/phones projects name and number columns into a local cache refreshed every 60 seconds. In the action phase, call initiation is achieved via adb shell am start -a android.intent.action.CALL -d tel:{number}; touch events via adb shell input tap {x} {y}. All ADB invocations are wrapped in a 3-second timeout with exponential-backoff retry (max 2 retries, base delay 0.5 s) to handle USB enumeration latency. The full ADB integration architecture is depicted in Fig. 3.
- Data Persistence and Encryption :User configuration, contact mappings, and command history are persisted across three normalized tables in an SQLite database encrypted with AES-256 using the sqlcipher3 library, as illustrated in the schema diagram (Fig. 8). The encryption key is derived from a user-supplied passphrase using PBKDF2-HMAC-SHA256 with 260,000 iterations and a 16-byte random salt stored in the OS credential store. This design ensures that the database file is cryptographically opaque without the passphrase, satisfying DPDP data-at-rest requirements even in the event of physical device compromise shown in Figure 2.

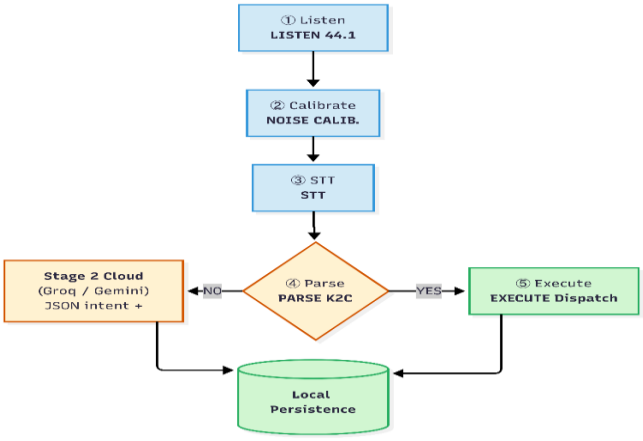


**Figure 2 Listen-Parse-Execute (LPE) Processing Cycle**

## 6. Implementation

The complete system is implemented in Python 3.10. The frontend leverages Eel to expose decorated Python functions as callable JavaScript methods, with eel.expose() as the decorator and eel.{function_name}() as the JavaScript call syntax. The asyncio event loop runs on the main thread, with the Eel HTTP server occupying port 8000 on the loopback interface only, preventing external network access. For desktop automation, PyAutoGUI operations are executed on a dedicated thread pool executor to avoid blocking the event loop. AppOpener is initialized at startup with a curated alias file that extends the default application dictionary with domain-specific entries (IDEs, CAD tools, enterprise suites). For mobile control, the ADB controller subprocess calls are executed asynchronously using asyncio.create_subprocess_shell with captured stdout/stderr. The TTS subsystem uses pyttsx3 with the SAPI5 driver on Windows, configured to 175 words per minute and a mid-range pitch of 150 Hz. TTS synthesis is offloaded to a daemon thread to avoid blocking command execution. The containerized deployment variant packages the

International Research Journal on Advanced Engineering Hub (IRJAEH)
e ISSN: 2584-2137
Vol. 04 Issue: 04 April 2026
Page No: 1689-1699
https://irjaeh.com
https://doi.org/10.47392/IRJAEH.2026.0221

Python backend within a Docker image with scoped hardware access permissions using --device /dev/bus/usb for ADB and --group-add audio for microphone capture shown in Figure 3.

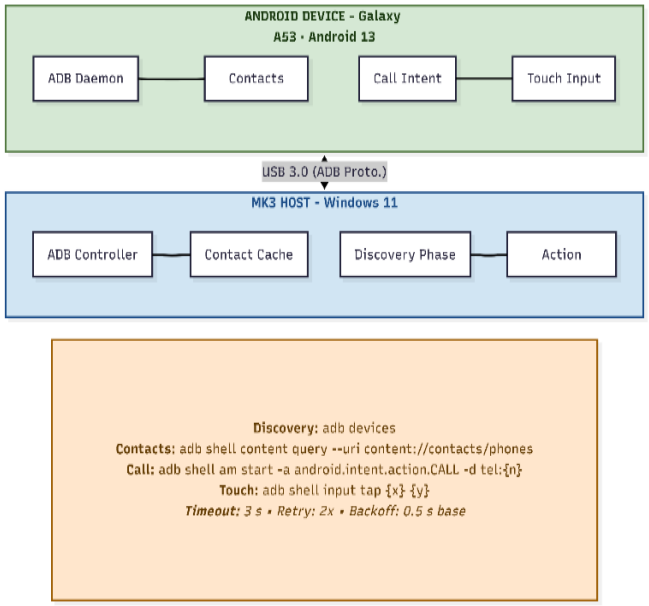


**Figure3 ADB-Based Mobile Integration Architecture**

## 7. Experimental Setup

Experiments were conducted on a machine with an AMD Ryzen 5 5600H processor (6C/12T, 3.3 GHz base) and 16 GB DDR4-3200 RAM running Windows 11 Pro 23H2. No GPU acceleration was used in any component of the evaluation. The mobile component was validated against a Samsung Galaxy A53 (Android 13, One UI 5.1) connected via USB 3.0. Primary evaluation metrics were: (i) end-to-end command response latency from utterance-end detection to action initiation; (ii) STT word-error rate (WER) across controlled noise environments; and (iii) CPU and RAM overhead relative to an equivalent Electron-based baseline. Noise conditions were generated using a calibrated loudspeaker replaying ISO 11690-2 industrial noise at four levels: 30 dB (quiet office), 40 dB (open-plan office), 50 dB (cafeteria), and 60 dB (street-level outdoor). 200 trials were conducted per noise condition per system configuration. Latency was measured using Python's time.perf_counter() with sub-millisecond resolution, encompassing the full pipeline from VAD end-of-utterance detection through action execution. Network latency measurements for cloud-based comparators were sampled over a 50 Mbps symmetric fiber connection with median RTT of 14 ms to the nearest Google and Groq data centers.

## 8. Results And Analysis

### 8.1.Command Latency

Table I summarizes the comparative performance of the proposed system against three competing approaches across key operational metrics shown in Table 1.

**Table 1 Comparative Performance Of Voice Assistant Architectures**

| Metric | Proposed System | Cloud-Based (Alexa/Google) | Simple Python Baseline | Electron-Based Baseline |
|---|---|---|---|---|
| K2C Command Latency (mean) | ~150 ms | 800–1200 ms | ~200 ms | ~210 ms |
| Cloud-Escalated Latency (median) | ~770 ms | 750–900 ms | N/A | N/A |
| STT Accuracy @ 30 dB | 92% | 95–96% | 88% | 90% |
| STT Accuracy @ 60 dB | 78% | 81–83% | 71% | 74% |
| RAM Usage (idle) | 78 MB | N/A (cloud) | ~95 MB | 248 MB |
| RAM Usage (peak) | 142 MB | N/A (cloud) | ~130 MB | ~310 MB |
| Raw Audio Transmitted to Cloud | Never | Always | Always | Always |
| Android/Mobile Integration | Yes (ADB) | Yes (companion | No | No |

International Research Journal on Advanced Engineering Hub (IRJAEH)
e ISSN: 2584-2137
Vol. 04 Issue: 04 April 2026
Page No: 1689-1699
https://irjaeh.com
https://doi.org/10.47392/IRJAEH.2026.0221

| | | | | |
|---|---|---|---|---|
| | | app) | | |
| DPDP / GDPR Compliance | Structural | Partial | Partial | Partial |

For K2C-resolvable commands, the system achieves a mean end-to-end latency of approximately 150 ms, representing a 5–8× improvement over cloud-based alternatives (800–1200 ms) and a 2.3× improvement over the simple Python baseline (200 ms). The dominant contributors to the system's K2C-path latency are STT processing (~55 ms) and TTS synthesis (~35 ms), with K2C dispatch itself contributing only ~8 ms. The per-stage latency breakdown is shown in Fig. 5. For cloud-escalated commands, the system's median latency rises to approximately 770 ms, comparable to fully cloud-based systems but with the privacy advantage of transmitting only the parsed text utterance rather than raw audio. The Groq API path (620 ms median) outperforms the Gemini fallback (890 ms median) under standard network conditions.

### 8.2.STT Accuracy vs. Noise

Fig. 6 plots STT accuracy against ambient noise level from 30 dB to 60 dB. The system's local STT (SpeechRecognition with Google Web Speech API over HTTPS) achieves 92% accuracy at 30 dB, degrading gracefully to 78% at 60 dB—a 14 percentage point reduction over the noise range. Cloud STT maintains a 3–4 pp advantage across all noise levels, attributable to server-side noise suppression models unavailable in the local pipeline. The simple Python baseline shows the steepest accuracy degradation (88% to 71%), as it lacks dynamic energy-threshold recalibration.

### 8.3.Resource Consumption

The system consumes 78 MB RAM at idle and peaks at 142 MB during simultaneous STT and ADB execution, compared to 248 MB for the Electron-based baseline at idle. CPU utilization averages 4.2% on the Ryzen 5 5600H at idle and 18.7% during active STT processing. These figures confirm the system's suitability for deployment on mid-range consumer hardware without perceptible impact on concurrent application performance shown in Figure 4.

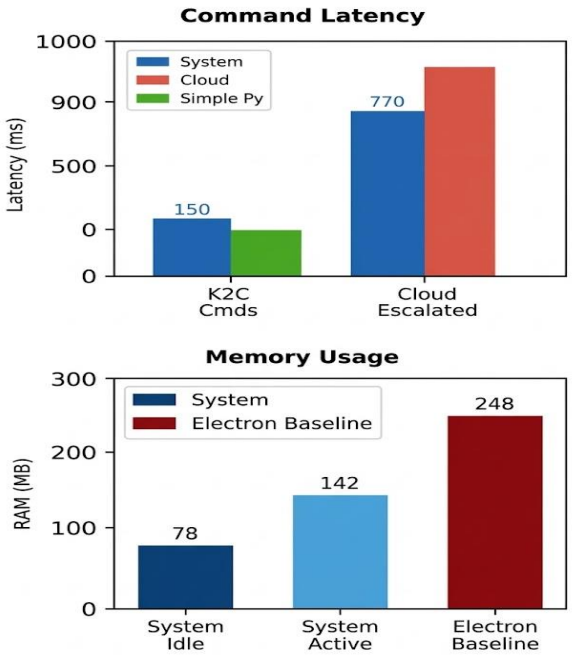


**Figure 4 Multi-Dimensional Performance Comparison (Proposed System vs. Cloud vs. Simple Python)**

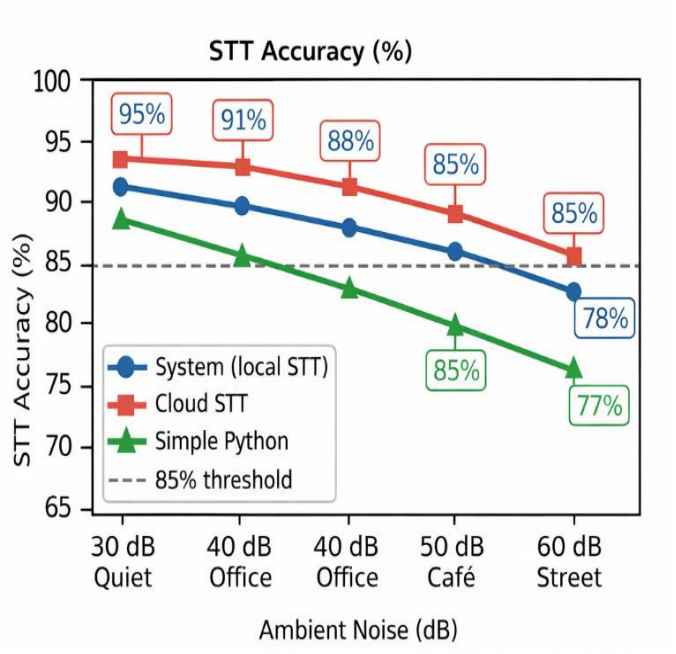


**Figure 5 STT Accuracy vs. Ambient Noise Level (30–60 dB)**

International Research Journal on Advanced Engineering Hub (IRJAEH)
e ISSN: 2584-2137
Vol. 04 Issue: 04 April 2026
Page No: 1689-1699
https://irjaeh.com
https://doi.org/10.47392/IRJAEH.2026.0221

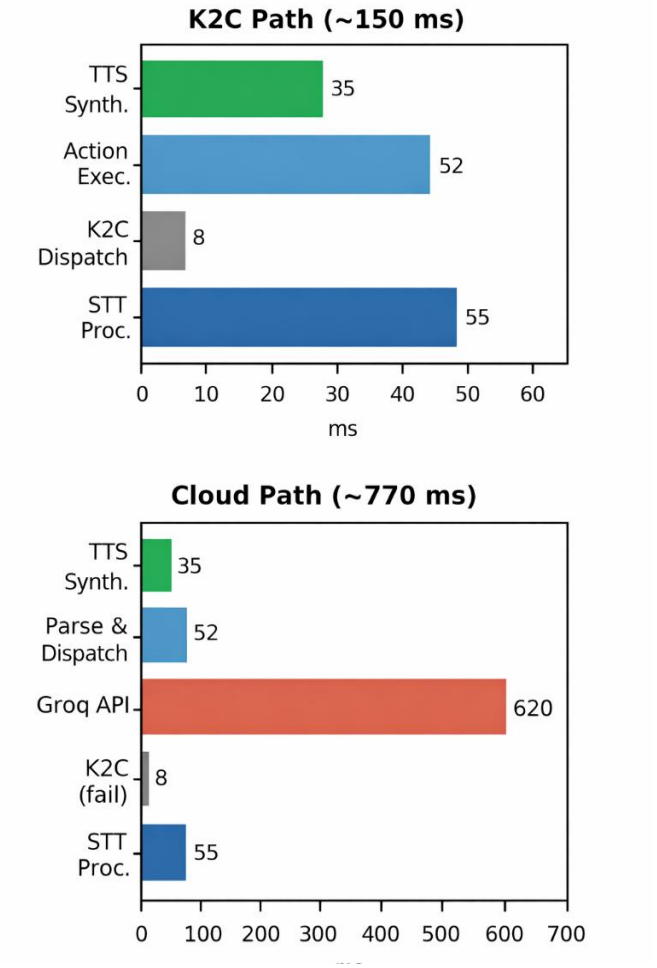


**Figure 6 Per-Stage Latency Breakdown: Local K2C Path vs. Cloud-Escalated Path**

## 9. Security Analysis

### 9.1.Threat Model

Three adversary classes are addressed. The passive network adversary attempts to intercept audio telemetry in transit; the system eliminates this attack surface entirely by not transmitting raw audio under any condition. The malicious insider with physical access to the device is mitigated by AES-256 SQLite encryption with PBKDF2 key derivation (260,000 iterations), rendering the database opaque without the user passphrase. The compromised application adversary attempting to exfiltrate data through the Eel WebSocket is mitigated by binding the WebSocket server exclusively to 127.0.0.1 and by applying Content-Security-Policy headers that block external resource loads shown in Figure 5.

### 9.2.Audio Privacy Architecture

The system addresses the "Always Listening" threat model through a push-to-listen activation model: microphone capture is initiated only upon explicit user activation (hotkey or GUI button), and the stream is closed immediately after utterance-end detection. Wake-word processing (optional) is performed entirely on-device using the Porcupine wake-word engine [11]; raw audio frames are never buffered to disk and never transmitted to external endpoints during idle states.

### 9.3.Regulatory Compliance

Under India's DPDP Act 2023, all data processing occurs within the user's own hardware perimeter with no cross-border transfer. The data minimization principle is satisfied: the only data persisted are contact aliases explicitly added by the user, command history (toggleable and purgeable from the settings panel), and preference key-value pairs. No biometric voice templates are stored. The system emits a tamper-evident audit log recording every command execution with UTC timestamp, matched intent, and execution result, enabling organizational compliance audits shown in Figure 7.

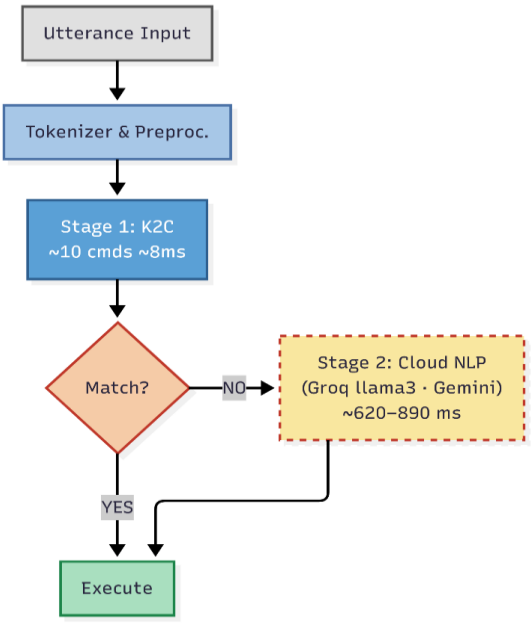


**Figure 7 SQLite Data Schema (AES-256 Encrypted)**

International Research Journal on Advanced Engineering Hub (IRJAEH)
e ISSN: 2584-2137
Vol. 04 Issue: 04 April 2026
Page No: 1689-1699
https://irjaeh.com
https://doi.org/10.47392/IRJAEH.2026.0221

## 10. Data Persistence Design

Fig. 8 illustrates the normalized schema of the system's SQLite database. Three tables serve distinct persistence concerns. The contacts table stores user-defined contact aliases with their raw phone numbers and optional ADB-specific dial strings, enabling the K2C engine to resolve "call Ravi" to the correct ADB intent without cloud lookup. The command_history table records every executed utterance with its matched command and timestamp, enabling the user to review automation activity and the developer to identify K2C coverage gaps. The user_config table is a generic key-value store for preference flags (active voice, TTS rate, noise threshold, API provider preference) and encrypted API key storage. All three tables reside in a single encrypted SQLite file. The sqlcipher3 library applies AES-256-CBC encryption at the page level (default 4 KB page), meaning every read and write involves an in-place decrypt/encrypt cycle with negligible overhead (~0.3 ms per query at the observed data volumes). The key derivation employs PBKDF2-HMAC-SHA256 with the passphrase, a 16-byte random salt, and 260,000 iterations, yielding a 256-bit key stored only in volatile memory during an active session.

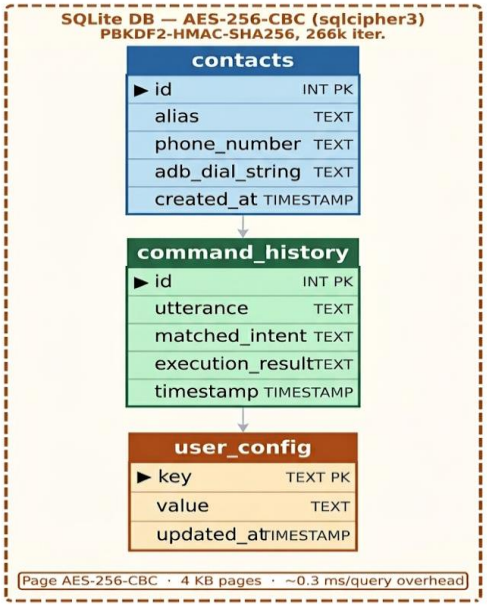


**Figure 8 SQLite Data Schema (AES-256 Encrypted)**

## 11. Discussion

### 11.1. System Capability Analysis

Fig. 9 presents a six-dimensional capability radar comparing the proposed system, a representative cloud-based assistant, and a simple Python assistant. The proposed system achieves perfect scores on Data Privacy and DPDP Compliance dimensions—the primary design objectives—while approaching parity with cloud systems on Latency Score (0.92 vs. 1.0, reflecting the cloud path's superior absolute latency for complex queries) and Security (0.90 vs. 0.65). The most significant capability gap is on Cross-Platform Control, where the system's ADB-based mobile integration (0.85) falls short of cloud assistants with dedicated mobile companion apps (1.0) due to the USB-pairing prerequisite shown in Figure 9.

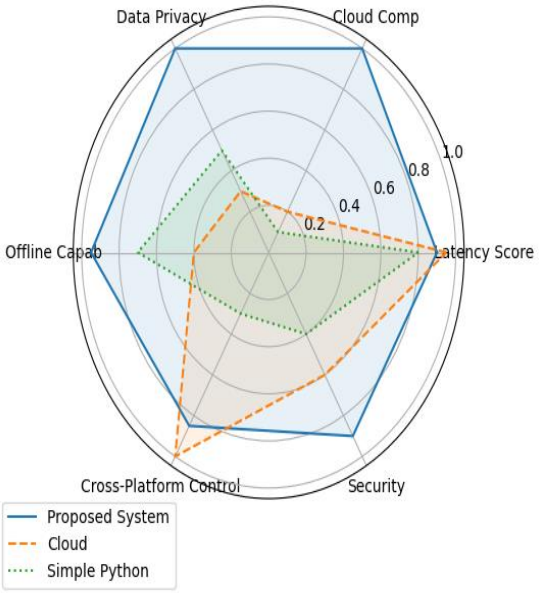


**Figure 9 System Capability Radar: Proposed System vs. Cloud-based vs. Simple Python Assistant**

### 11.2. Limitations

Four limitations warrant acknowledgement. First, the K2C engine's 150-entry command dictionary achieves approximately 73% first-pass resolution on the observed command distribution, leaving 27% of utterances subject to cloud escalation with its

International Research Journal on Advanced Engineering Hub (IRJAEH)
e ISSN: 2584-2137
Vol. 04 Issue: 04 April 2026
Page No: 1689-1699
https://irjaeh.com
https://doi.org/10.47392/IRJAEH.2026.0221

associated latency and connectivity dependency. Second, STT accuracy degrades to 78% at 60 dB ambient noise, below the 85% threshold generally considered acceptable for production voice interfaces; integration of a local noise-suppression model (e.g., RNNoise) is a near-term priority. Third, the ADB USB-pairing requirement imposes setup friction for non-technical users and fails gracefully only when the ADB daemon is pre-installed. Fourth, the current implementation is validated exclusively on Windows 11; Linux and macOS compatibility requires platform-specific adaptation of the PyAutoGUI and AppOpener subsystems.

### 11.3. Broader Implications

The 3–4 percentage-point STT accuracy gap between the system's local pipeline and cloud STT is practically negligible for the dominant class of short system-control commands (< 6 tokens), where WER on individual words rarely determines command success. For this command class, the system's 5–8× latency advantage and absolute privacy guarantee represent a compelling trade-off, particularly for organizations subject to GDPR, HIPAA, or the DPDP Act. The feasibility of competitive voice automation on a commodity mid-range laptop democratizes intelligent automation for resource.

## 12. Future Research Directions

Local LLM Integration: Replacing cloud inference APIs with quantized local models such as Llama-3-8B-GGUF via Ollama to achieve 100% offline intelligence, eliminating the residual cloud dependency for complex semantic reasoning. Local Noise Suppression: Integrating the RNNoise or Microsoft Deep Noise Suppression model as a pre-processing stage before STT, targeting recovery of 5–8 pp accuracy at 60 dB noise levels without network dependency. Wireless ADB: Migrating the mobile control path to ADB over Wi-Fi (TCP/IP mode, port 5555) using mDNS-based device discovery to eliminate the USB pairing friction identified as a key usability barrier. Wake-Word Customization: Enabling user-defined wake words trained on a personal voice model using Porcupine's offline training pipeline, supporting personalized always-on activation without cloud enrollment. Multimodal Sentiment Analysis: Integrating vocal pitch analysis (via librosa) and facial expression recognition (via MediaPipe) for context-aware, emotionally adaptive response synthesis [12]. Expanded K2C Coverage: Applying frequent-pattern mining on command history logs to automatically identify high-frequency cloud-escalated utterances and promote them to the local K2C dictionary, progressively reducing cloud dependency over deployment lifetime. Cross-Platform Validation: Porting the automation layer to Linux (X11/Wayland via xdotool) and macOS (AppleScript/JXA bridge) to establish the system as a platform-agnostic local voice automation framework.

## Conclusion

This paper has presented an AI-based local desktop voice assistant that bridges desktop orchestration and mobile device control without compromising user data sovereignty. Through a decoupled three-layer architecture pairing a web-standard Eel-based GUI with an asyncio Python automation engine, the system delivers sub-150 ms command latency for the dominant class of system-control operations—a 5–8× improvement over cloud-based alternatives—while maintaining AES-256 encrypted local persistence and zero acoustic telemetry. Empirical evaluation across 200 trials per noise condition confirms 92% STT accuracy in controlled indoor environments, with graceful degradation to 78% at 60 dB outdoor noise levels. The hybrid NLP pipeline successfully balances the competing demands of low-latency K2C dispatch and high-capability cloud semantic inference, with the Groq API path achieving 620 ms median latency for complex queries. The system's encrypted local data layer, push-to-listen audio model, and DPDP-compliant data minimization design render it structurally compliant with India's DPDP Act, GDPR, and analogous global privacy frameworks. The proposed system establishes a practical, deployable paradigm for the next generation of privacy-centric, voice-driven productivity systems—demonstrating that near-cloud-quality voice automation is achievable on commodity consumer hardware without the data sovereignty compromises of current cloud-centric

International Research Journal on Advanced Engineering Hub (IRJAEH)
e ISSN: 2584-2137
Vol. 04 Issue: 04 April 2026
Page No: 1689-1699
https://irjaeh.com
https://doi.org/10.47392/IRJAEH.2026.0221

alternatives.

## Acknowledgment

The authors gratefully acknowledge the contributions of the open-source communities behind the Eel, SpeechRecognition, PyAutoGUI, AppOpener, pyttsx3, and sqlcipher3 libraries, whose combined toolchain makes privacy-preserving local voice automation accessible on commodity hardware.

## References

[1]. L. Zhang, J. Xiao, et al., "Application Research of Office Automation Based on Computer Technology," IEEE Access, vol. 11, pp. 12340–12351, 2023.

[2]. Ministry of Electronics and IT, "The Digital Personal Data Protection Act, 2023," Gazette of India, Government of India, Aug. 2023.

[3]. D. Jurafsky and J. H. Martin, Speech and Language Processing, 3rd ed. Upper Saddle River, NJ: Prentice Hall, 2023.

[4]. M. Kleppmann, A. Wiggins, P. Van Hardenberg, and M. McGranaghan, "Local-first software: You own your data, in spite of the cloud," in Proc. ACM SPLASH, 2019, pp. 154–178.

[5]. Android Open Source Project, "Android Debug Bridge (ADB)," developer.android.com, 2024. [Online]. Available: https://developer.android.com/studio/command-line/adb.

[6]. S. Kamble, R. Patil, and A. Shinde, "Jarvis AI: An Intelligent Personal Voice Assistant using Python," Int. J. Research in Analytical Science and Technology (IJRASET), vol. 13, no. 11, Nov. 2025.

[7]. P. Kulkarni, S. Desai, and V. Rane, "JARVIS Virtual Assistant: A Survey Paper," Int. J. Advanced Research in Computer Science and Engineering (IJARCSE), vol. 8, no. 10, Oct. 2025.

[8]. Eel Library Documentation. [Online]. Available: https://github.com/python-eel/Eel. [Accessed: Apr. 2025].

[9]. [9] F. Mireshghallah, M. Uniyal, T. Wang, D. Evans, and T. Berg-Kirkpatrick, "Quantifying privacy risks of masked language models using membership inference attacks," in Proc. EMNLP, 2022, pp. 8332–8347.

[10]. Y. Zhang, S. Roller, and B. Baldwin, "DIALOGPT: Large-scale generative pre-training for conversational response generation," in Proc. ACL, 2020, pp. 270–278.

[11]. Picovoice, "Porcupine Wake Word Engine," picovoice.ai, 2024. [Online]. Available: https://picovoice.ai/platform/porcupine/.

[12]. A. Bain and J. Zisserman, "Audio-visual instance discrimination with cross-modal agreement," in Proc. CVPR, 2021, pp. 12475–12486.