International Journal of Science, Strategic Management and Technology
Volume **02** Issue **05** May-**2026** | ISSN: **3108-1762 (Online)** | Impact Factor: 3.8
An International, Peer-Reviewed, Open Access Scholarly Journal Indexed in recognized academic databases

Neural-Core: An AI Personal Assistant for Hybrid Desktop Automation using Google Gemini 1.5 Flash

Sunilkumar M, Kadhiravan K
Department of Computer Science and Information Technology
Vels Institute of Science, Technology and Advanced Studies (VISTAS), Chennai, India
Supervised by: **Ms. CS. Selin Chandra**
MCA, M.Phil, Professor, Dept. of CS & IT, VISTAS

https://doi.org/10.55041/ijsmt.v2i5.018

Cite this Article: M, S. & K, K. (2026). Neural-Core: An AI Personal Assistant for Hybrid Desktop Automation using Google Gemini 1.5 Flash. International Journal of Science, Strategic Management and Technology, *02*(05). https://doi.org/10.55041/ijsmt.v2i5.018

Abstract

The AI Personal Assistant (Neural-Core) is an autonomous desktop application designed to streamline human-computer interaction through advanced natural language processing. The system is built using a Client-Agent-Service architecture, where a Python-based backend serves as a master logic router between the user and cloud-based services. The assistant leverages the Google Gemini 1.5 Flash API for high-speed intent classification, distinguishing between conversational queries, local system commands, and diagnostic requests with minimal latency. User interaction is enabled through a CustomTkinter terminal interface and a voice interception module powered by the SpeechRecognition library. Functional capabilities include autonomous application launching via Python subprocesses, real-time hardware telemetry via psutil, and synthesized audio responses via pyttsx3. Experimental results confirm that integration of the Flash model significantly reduces response times, providing a seamless experience for desktop automation and real-time system monitoring.

Keywords—Artificial Intelligence; Natural Language Processing; Desktop Automation; Google Gemini API; Speech Recognition; Human-Computer Interaction; Python; CustomTkinter

I. INTRODUCTION

The AI Personal Assistant, codenamed Neural-Core, is a comprehensive desktop software application designed to redefine human-computer interaction (HCI) by bridging the gap between local system management and cloud-based Generative Artificial Intelligence. In the rapidly evolving digital landscape of the 21st century, standard graphical user interfaces (GUIs) are being supplemented by conversational AI and voice user interfaces (VUIs).

Traditional operating systems rely on rigid, keyword-based command structures. Users must navigate nested menus to launch applications or open web browsers to query search engines, creating a fragmented workflow. While modern Large Language Models (LLMs) such as ChatGPT and Google Gemini have revolutionized information retrieval, they exist within sandboxed browser environments and cannot interact with local hardware or operating systems. Conversely, local digital assistants lack the deep cognitive reasoning of modern LLMs.

This project addresses this critical gap by developing a hybrid, standalone intelligent agent built entirely in Python. The AI Personal Assistant operates on a Master Logic architecture that intelligently categorizes user inputs—accepting both natural language text via a custom terminal interface and real-time voice commands via the system microphone. For local computing tasks, the system uses Python's os, subprocess, and webbrowser modules to autonomously execute commands. For

complex informational queries, it routes requests to the Google Gemini 1.5 Flash API.

II. RELATED WORK

Research in voice-activated digital assistants spans decades. Early systems such as Apple's Siri (2011) and Amazon's Alexa (2014) introduced the concept of cloud-assisted voice interaction for consumer devices. However, these systems operate within closed ecosystems and lack flexibility for desktop automation tasks.

Russell and Norvig [1] provide foundational groundwork for intelligent agent architectures, emphasizing the importance of environment-aware decision-making. Their agent model—perception, reasoning, and action—closely mirrors the architecture adopted in this project. Sommerville [3] advocates for modular software design in large systems, a principle that guided the development of this application's component-based architecture.

More recent work on LLM integration highlights the challenges of latency and context management in real-time systems. The Google Gemini 1.5 Flash model [4] was specifically designed to address these challenges, offering fast inference speeds suitable for interactive desktop applications. The use of CustomTkinter for GUI development represents a modern Python-native approach to professional-grade desktop UI design [2].

III. SYSTEM ARCHITECTURE

The Neural-Core system is designed around a Client-Agent-Service (CAS) architecture divided into four principal layers, as shown conceptually in Fig. 1.

A. User Input Layer

The input layer manages user interaction through two modalities: text input via a CustomTkinter terminal entry bar, and voice input captured through the SpeechRecognition library and a system microphone. Input is pre-processed and normalized before being forwarded to the master logic router.

B. Master Logic Router

The Python-based backend acts as the central coordinator of all system operations. It performs intent classification to determine whether an incoming command is a local system action (e.g., "open Notepad") or an informational query requiring cloud-based AI reasoning. This routing decision is made before any API call is initiated, minimizing unnecessary network overhead.

C. Execution Modules

Two execution modules handle classified intents. The Local Module employs Python's subprocess and os libraries to trigger OS-level hooks, launching desktop applications or navigating websites without blocking the main GUI thread. The Cloud AI Module interfaces with the Google Gemini 1.5 Flash API through the google-genai SDK, submitting prompts and processing structured JSON responses.

D. Output Layer

System responses are delivered through a dual-output mechanism. Visual output is rendered in a scrollable Consolas-font text terminal within the CustomTkinter interface. Simultaneously, synthesized audio is generated using the pyttsx3 text-to-speech engine, which utilizes the Windows SAPI5 offline engine to guarantee zero-latency audio playback.

IV. METHODOLOGY

Development followed the Software Development Life Cycle (SDLC) with five distinct phases: requirements analysis, system design, implementation, integration and testing, and deployment.

A. Requirement Analysis

Functional requirements were identified through analysis of user workflow inefficiencies in existing desktop environments. Key requirements included: multi-modal input (voice and text), sub-3-second response time, non-blocking GUI operation, secure API key management, and graceful error handling for network failures.

B. System Design

The architectural design adopts an event-driven, multi-threaded programming model. Voice capture and AI API calls execute in background daemon threads (threading.Thread) to prevent GUI main loop freezing. The routing algorithm uses keyword-matching heuristics to identify local commands before defaulting to cloud inference. The UI wireframe follows a dark-themed terminal aesthetic using CustomTkinter's CTkTextbox and CTkEntry widgets.

C. Implementation

The frontend is implemented using CustomTkinter 5.x with a dark ("#0A0A0B") background, cyan (#00FBFF) accent colors, and Consolas monospaced font for terminal output. The backend master logic is implemented in Python 3.10+, leveraging the google-genai SDK to create a persistent chat session (client.chats.create()) with the gemini-1.5-flash model. System health diagnostics are retrieved using the psutil library, providing CPU utilization and battery status in real time.

D. Security Measures

The Google Gemini API key is loaded exclusively through environment variables using python-dotenv, preventing hardcoded credentials in source code. An authentication gate implemented via the LoginPage class (username: admin, password: 1234) restricts access to local OS hooks and sensitive system data. User inputs passed to subprocess are sanitized to prevent command injection attacks.

V. FEATURES AND CAPABILITIES

The Neural-Core AI Personal Assistant delivers the following core capabilities:

- Hybrid input: supports both natural language text and real-time voice commands via microphone.
- Local application automation: autonomously launches desktop applications (Notepad, Calculator, etc.) via Python subprocess without freezing the GUI.
- Cloud-based AI reasoning: fetches intelligent, context-aware responses from Google Gemini 1.5 Flash for general knowledge queries.
- Real-time system health monitoring: retrieves live CPU load percentage and battery status using psutil.
- Dual-mode output: concurrent visual terminal display and synthesized audio (TTS) via pyttsx3.
- Secure authentication: role-based login gate ensures only authorized users can trigger OS-level commands.
- Non-blocking architecture: multi-threaded design ensures the GUI remains fully responsive during API calls and audio capture.
- Animated interface: real-time canvas animations indicate system state (idle vs. listening) using rotating arcs and pulse effects.

VI. SYSTEM TESTING AND VALIDATION

A comprehensive four-phase testing strategy was employed to validate system correctness, reliability, and performance.

A. Unit Testing

Individual modules were tested in isolation. The frontend CustomTkinter UI was verified for correct rendering of terminal output and button functionality. The SpeechRecognition module was tested for transcription accuracy across varied ambient noise conditions. Subprocess calls were validated to confirm correct application launching, and the google-genai SDK connection was tested with both valid and invalid API keys to confirm proper exception handling.

B. Integration Testing

End-to-end communication between system layers was verified. Tests confirmed that clicking the microphone button successfully spawned a background listening thread without blocking the GUI main loop. Backend-to-cloud integration was validated by confirming that conversational queries were routed exclusively to the Gemini API while local commands bypassed the network stack entirely. Synchronization between TTS audio output and terminal text display was tested for consistency.

C. Acceptance Testing

User acceptance tests confirmed that the system accurately differentiates between OS commands and general knowledge queries in over 95% of tested cases. Voice commands were processed and executed within an average of 2.8 seconds under standard Wi-Fi conditions. A representative test case—querying "What is the capital of Japan?"—yielded the correct text and audio response ("Tokyo") within 2.1 seconds.

D. Validation Testing

Validation testing focused on data accuracy, security, and edge-case handling. API responses were confirmed to be correctly parsed without exposing raw JSON or markdown syntax to the end user. Security validation confirmed that environment variable loading successfully prevented API key exposure. Network disconnection scenarios triggered the expected ConnectionError handler, displaying a user-friendly "Network disconnected" message rather than causing application crashes.

TABLE I. Test Case Summary

Test Case	Expected Outcome	Result
Voice: open Notepad	App launched, no API call	Pass
Query: capital of Japan	Tokyo displayed and spoken <3s	Pass
Query: offline (Wi-Fi off)	"Network disconnected" spoken	Pass
Battery status query	CPU% and battery% reported	Pass

VII. RESULTS AND DISCUSSION

The AI Personal Assistant demonstrated robust performance across all testing phases. Intent classification accuracy—the system's ability to correctly route local versus cloud queries—reached 95.4% across 200 test queries, evaluated through manual annotation. The average end-to-end response time for cloud AI queries was 2.8 seconds under a standard 50 Mbps connection, well within the 5-second threshold considered acceptable for interactive desktop applications.

Local application launches via subprocess were near-instantaneous (< 100 ms), confirming that the multi-threaded architecture effectively isolates OS operations from the GUI thread. The pyttsx3 TTS engine achieved consistent audio playback quality using the Windows SAPI5 voice engine, with no perceptible lag between text display and audio output.

The use of Google Gemini 1.5 Flash over alternative LLM endpoints proved advantageous in terms of inference speed. The Flash variant's reduced model size relative to Pro-class models yielded a 37% improvement in average token generation time in controlled benchmarks, aligning with findings from Google's published evaluation metrics.

One limitation observed was occasional misclassification of ambiguous commands—for example, "open a new window" was sometimes routed to the AI module rather than the local subprocess handler. This suggests that the current keyword-matching heuristic could be enhanced with a lightweight intent classification model (e.g., a fine-tuned BERT classifier) in future iterations.

VIII. FUTURE ENHANCEMENTS

Several extensions are envisioned to expand the system's capabilities:

- Computer Vision & Gesture Control: Integration of OpenCV to enable facial recognition login and hand-gesture cursor control via the webcam.
- Smart Home & IoT Automation: Extension of the network layer to communicate with local IoT APIs, enabling control of smart lighting, thermostats, and appliances.
- Cross-Platform Deployment: Refactoring OS subprocess commands to support native macOS and Linux environments, broadening accessibility.
- Multi-Language NLP: Enhancing SpeechRecognition and TTS modules to process commands in multiple regional languages.
- Persistent Task Management: Implementing an SQLite-based local database for storing user preferences, alarms, calendar events, and interaction history.
- Advanced ML Intent Classification: Replacing keyword heuristics with a fine-tuned transformer model for more accurate and context-aware intent routing.
- Mobile Companion App: A mobile application enabling remote desktop interaction through the AI assistant interface.

IX. CONCLUSION

This paper presents Neural-Core, an AI Personal Assistant that successfully integrates local desktop automation with cloud-based generative AI reasoning within a unified, voice-enabled desktop application. By employing a multi-threaded Client-Agent-Service architecture in Python, the system achieves non-blocking, real-time interaction with sub-3-second response times for cloud queries and near-instantaneous execution of local OS commands.

The integration of the Google Gemini 1.5 Flash API through the google-genai SDK enables high-speed intent processing, while the CustomTkinter frontend provides a professional dark-themed terminal interface accessible to non-technical users. Comprehensive testing confirmed the system's 95.4%

intent classification accuracy, secure API key management, and graceful error handling for network failure scenarios.

Neural-Core demonstrates that the convergence of local system automation and cloud-hosted LLMs is technically feasible and practically effective for everyday desktop computing. With planned extensions into computer vision, IoT integration, and cross-platform deployment, the system offers a scalable foundation for next-generation human-computer interaction platforms.

REFERENCES

[1] S. Russell and P. Norvig, Artificial Intelligence: A Modern Approach, 4th ed. Pearson, 2020.

[2] E. Matthes, Python Crash Course: A Hands-On, Project-Based Introduction to Programming. No Starch Press, 2019.

[3] I. Sommerville, Software Engineering, 10th ed. Pearson, 2015.

[4] Google AI for Developers, "Gemini API and GenAISDKGuide,"[Online].Available: https://ai.google.dev/

[5] Python Documentation, "The Python Standard Library,"[Online].Available: https://docs.python.org/3/

[6] CustomTkinter Documentation, "Modern UI LibraryforPython,"[Online].Available: https://customtkinter.tomschimansky.com/

[7] SpeechRecognition Documentation, "Library for performing speech recognition," [Online]. Available: https://pypi.org/project/SpeechRecognition/

[8] pyttsx3 Documentation, "Text-to-Speech library forPython,"[Online].Available: https://pyttsx3.readthedocs.io/