International Journal of Computational Research in Science and Technology
E-ISSN: 3108-0871, Volume 2 Issue 3
May-Jun 2026, PP 01-10

AVIA: A Multi-Agent Multimodal Virtual Assistant for Autonomous Digital Task Automation

Hitesh Vinod Dadlani, Ayaan Badshah Khan,
Munaf Irfan Shaikh, Prof. Jayshri Kandekar
Dept. Artificial Intelligence and Data Science MET's Institute of Engineering Nashik, Maharashtra, India

Abstract. The rapid advancement of Large Language Models (LLMs) has significantly expanded the capabilities of intelligent software agents. Traditional virtual assistants are limited to reactive query-response interactions and lack persistent memory, flexible tool integration, and autonomous task execution. This paper presents AVIA (Autonomous Virtual Intelligent Assistant), a locally-hosted multi-agent multimodal assistant designed for autonomous digital task automation through an extensible skill-based architecture. AVIA integrates LLM reasoning with a persistent markdown-based memory system, a modular skill execution framework, and proactive scheduling via heartbeat loops and cron-based task management. The system supports multimodal interaction through text and voice interfaces and integrates with external platforms including email, calendar, document management, and social media. A local-first design philosophy ensures user privacy and independence from cloud infrastructure. Experimental evaluation across five task categories demonstrates a Task Completion Rate (TCR) of 91.3%, an Automation Success Rate (ASR) of 88.6%, mean response latency of 2.8 s, and memory retrieval accuracy exceeding 93%, validating AVIA as a flexible, privacy-preserving foundation for next-generation intelligent personal assistants.

Keywords: Autonomous Agents, Large Language Models, Multi-Agent Systems, Multimodal Interaction, Skill-Based AI Architecture, Task Automation, Virtual Personal Assistants

I. Introduction

Virtual personal assistants (VPAs) such as Amazon Alexa, Google Assistant, Apple Siri, and Microsoft Cortana allow users to interact with computing environments through natural language [1], [2]. These systems excel at well-defined, single-turn tasks: setting reminders, answering factual queries, and controlling smart home devices. However, a fundamental limi-tation constrains their practical utility: they are predominantly reactive, responding to isolated commands without planning or executing multi-step workflows [2]. A user who wishes to research a topic, compile a summary, send

emails, and schedule a follow-up meeting must issue each action as a separate, manually sequenced command.

The emergence of powerful LLMs—including OpenAI's GPT-4 [5], Anthropic's Claude, and Meta's LLaMA series [6]—has created new possibilities for AI-driven task au-tomation. Platforms such as ChatGPT and Claude.ai have demonstrated that LLMs can serve as reasoning engines for more capable assistant systems. Nevertheless, deploying LLMs as practical autonomous assistants requires: (1) persistent memory across sessions, (2) modular tool and skill execution frameworks, (3) multi-agent coordination for distributing com-plex tasks [11], and (4) autonomous scheduling for proactive background operation.

This paper presents AVIA (Autonomous Virtual Intelligent Assistant), a locally-hosted, open-architecture virtual assistant designed from first principles to address these requirements. AVIA adopts a local-first design philosophy—all reasoning, memory management, and task execution occur on the user's hardware whenever possible—raising concerns about data pri-vacy, latency, and offline availability that motivate this work. The principal contributions are:

- A novel multi-agent architecture with specialized agents for research, automation, communication, and software development.
- A modular, plug-and-play skill-based execution frame-work expandable without modifying core logic.
- A hybrid persistent memory architecture combining short-term context, long-term markdown storage, task memory, and semantic vector retrieval [13], [14].
- An autonomous scheduling subsystem implementing heartbeat loops and cron-based task management.
- A local-first architecture prioritizing user privacy and independence from cloud services.
- Systematic experimental evaluation across representative real-world task categories.

II. Related Work

1. Evolution of Virtual Personal Assistants

Early virtual assistants emerged in the 1990s as speech-driven command interfaces. The introduction of Apple's Siri in 2011 marked a significant inflection point, establishing the template for modern voice assistants [1]. Subsequent platforms—Google Assistant (2016), Amazon Alexa (2014), and Microsoft Cortana (2014)—improved speech recognition and integration but remain fundamentally input-output systems without autonomous behavior or multi-step planning [2].

2. LLMs as Reasoning Engines

The Transformer architecture [3] and subsequent large-scale pretraining produced models such as GPT-3 [4], GPT-4 [5], and the LLaMA family [6] with unprecedented instruction-following and reasoning capabilities. These models can inter-pret complex

instructions, generate structured outputs, produce executable code, and engage in extended dialogues. However, their standard deployments remain largely stateless and single-agent, limiting utility for complex autonomous workflows.

3. Autonomous and Agentic AI Systems

AutoGPT [7] and BabyAGI demonstrated early attempts at autonomous LLM-driven task execution. The ReAct frame-work [8] formalized the interleaving of reasoning traces and action execution. Toolformer [9] trained models to invoke external APIs autonomously, while LangChain [10] provided developer infrastructure for LLM-powered applications with tool integration, memory management, and agent orchestration.

4. Multi-Agent and Memory Systems

Microsoft's AutoGen framework [11] enables conversa-tional multi-agent pipelines. Generative Agents [12] demon-strated that LLM-powered agents with persistent memory produce realistic long-horizon behaviors. MemGPT [13] introduced a hierarchical memory architecture analogous to OS memory management. Vector databases including FAISS enable semantic similarity search supporting Retrieval-Augmented Generation (RAG) [14] patterns. Despite progress in individual areas, no integrated, locally deployable sys-tem combines all these capabilities within a single privacy-preserving architecture—the gap AVIA addresses.

III. Avia System Architecture

The AVIA architecture is a layered, modular system follow-ing a pipeline pattern: user input enters through the Interface Layer, is interpreted by the Controller Agent, delegated to specialized agents via the Agent Manager, executed through the Skill Execution Layer, and resolved through external service integrations. Results propagate back to the user through the same chain.

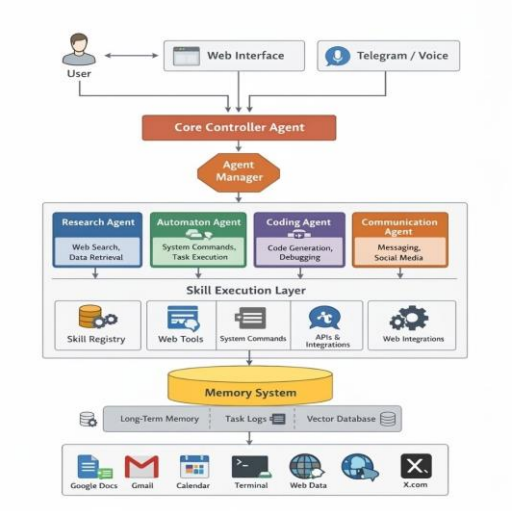

Fig. 1. AVIA Multi-Agent System Architecture

Table 1: Avia Architectural Layers

Layer	Component	Responsibility
1 Interface	Voice & Text UI	Input acquisition and response rendering
2 Controller	Core Agent	Intent parsing, decomposition, routing
3 Coordination	Agent Manager	Lifecycle, dispatch, result aggregation
4 Execution	Specialized Agents	Domain-specific task execution
5 Skill API	Skill Layer	Standardized tool invocation interface
6 Integration	External Services	Email, calendar, web, file, social media

Multi-Agent Framework

The AVIA agent system comprises a Core Assistant Agent serving as the system orchestrator, supported by specialized agents. Each agent maintains its own operational context, skill access permissions, and communication protocols, interacting through a structured message-passing interface. Table II lists agent roles and representative capabilities.

When a user submits a request, the Core Agent performs in-tent classification using structured chain-of-thought prompting [8], decomposing it into atomic subtasks assigned to the most appropriate specialized agent. Agents operate sequentially for dependent tasks or in parallel for independent subtasks, with the Agent Manager coordinating execution and aggregating results. For example, the request “Research the top five AI

Table 2: Avia Agent Roles And Capabilities

Agent	Representative Skills
Core Assistant	Intent parsing, planning, routing, synthesis
Research	Web search, scraping, summarization, fact-checking
Automation	File management, command execution, scripting
Coding	Code generation, debugging, test writing
Communication	Email composition, calendar management, notifications
Memory	Memory read/write, semantic retrieval, context curation

conferences in 2025, summarize the key themes, and email the summary to my research team” triggers: (1) the Research Agent queries academic databases; (2) the Core Agent syn-thesizes results; (3) the Communication Agent retrieves team contacts from memory; and (4) composes and dispatches the email—all from a single command.

Memory Architecture
AVIA implements a four-tier hybrid memory architecture inspired by MemGPT [13] and Generative Agents [12]. Short-Term Memory (STM) maintains active conversation history in a sliding context window; when approaching the context limit, the Memory Agent summarizes older exchanges and promotes key facts. Long-Term Memory (LTM) is structured Markdown files organized by topic, project, and contact, se-lected for human readability and version control compatibility. Task Memory records com-pleted, failed, and pending tasks with execution logs, enabling workflow resumption and audit trails. Semantic Memory employs vector embeddings (all-MiniLM-L6-v2 by default) stored in a FAISS index, retrieving relevant fragments via cosine similarity without exhaustive scanning.

Skill-Based Execution Framework
Each skill is a self-contained module exposing a standard-ized interface: name, descrip-tion, JSON Schema input/output, and an execute function. Skills span six domains: In-formation (web search, fetch, summarize, academic search); Commu-nication (email compose/send, calendar create, notifications); File System (read, write, search, ar-chive); System (command execute, process list, clipboard); Development (code exe-cute, lint, git commit, test run); and Social Media (post, read feed, schedule, analyze engagement). New skills are added by placing a conformant module in the skills direc-tory and restarting the skill registry—no modifications to core logic required.

Autonomous Scheduling and Local-First Architecture AVIA implements two schedul-ing mechanisms: a Heart-beat Loop (default 60 s interval) that polls a pending task queue and dispatches due actions without user intervention; and a Cron-based Sched-uler for time-based triggers (e.g., daily briefings at 08:00, weekly status reports). Both are im-plemented in Go for reliable concurrency. For LLM inference, a configurable backend supports local model servers (Ollama, LM Studio, llama.cpp) and optional cloud providers (OpenAI, Anthropic [5], Cohere), abstracted behind a common inter-face. Local targets include Mistral-7B, LLaMA-3-8B [6], and Phi-3, ensuring sensitive data never leaves the device unless explicitly configured.

IV. System Implementation

AVIA uses a polyglot architecture where each component is built in the language best suited to its requirements (Ta-ble III). The Rust core runtime implements the Agent Manager and execution loops; Tokio enables non-blocking concurrent agent operation. The Go service layer exposes a REST API, handles inter-process communication, and implements the Heartbeat and Cron schedulers as goroutine-based workers. The Type-Script/React frontend supports text conversation, real-time streaming via WebSocket, voice input via the Web Speech API, and a task dashboard. The Python AI integration layer wraps LLM SDKs (openai, anthropic, ollama) behind a common interface and uses sentence-transformers [14] to generate 384-dimensional dense vectors for FAISS indexing.

Table 3: Implementation Technology Stack

Component	Technology	Rationale
Core Runtime / Agent Engine	Rust 1.77	Memory safety, high throughput
Networking / Scheduling	Go 1.22	Goroutine concurrency, low latency
Web User Interface	TypeScript/React	Real-time streaming UI
AI / ML Pipeline	Python 3.11	LLM SDKs, embedding models
Vector Store	FAISS (Python)	Efficient local similarity search
Memory Storage	Markdown / Git	Human-readable, version-controlled
Local LLM Backend	Ollama / llama.cpp	Consumer hardware inference

Voice interaction follows a five-stage pipeline: (1) wake word detection via Porcupine (offline); (2) audio capture via PortAudio; (3) speech-to-text via OpenAI Whisper (local base.en, ≈140 MB); (4) LLM processing and agent execu-tion; (5) text-to-speech via Coqui TTS (local) or ElevenLabs (cloud). End-to-end voice latency averages 3.2 s on an Intel Core i7 12th-gen system, dominated by speech synthesis (≈1.1 s) and LLM inference (≈1.6 s).

V. Experimental Methodology

Setup and Task Categories
All experiments used: Intel Core i7-12700H (14-core), 32 GB DDR5 RAM, NVIDIA RTX 3060 (6 GB VRAM),

NVMe SSD, Ubuntu 22.04 LTS, LLaMA-3-8B-Instruct Q4 K M via Ollama, and all-MiniLM-L6-v2 embeddings. External API integrations used sandbox accounts. Five task categories (20 tasks each, 100 total) were evaluated across three complexity tiers: Simple (1–2 skills, 1 agent; 40%),

Moderate (3–5 skills, 2 agents; 40%), and Complex (5+ skills,
3+ agents; 20%).

- Cat. 1 – Information Retrieval & Summarization: Locating, extracting, and condensing information from multiple web sources.
- Cat. 2 – Communication Automation: Composing, per-sonalizing, and dispatching emails or calendar invitations from natural language.
- Cat. 3 – Calendar & Schedule Management: Temporal reasoning, conflict detection, and cross-timezone event creation.
- Cat. 4 – Code Generation & Analysis: Generating functional code snippets, identifying bugs, or producing unit tests.

- Cat. 5 – Automated Content Publication: Multi-step workflows combining research, content generation, and social media posting.

Evaluation Metrics and Baselines
Metrics recorded per task: TCR (binary task objective completion, judged by two independent evaluators); ASR (pro-portion of sub-steps completed without manual intervention); ART (wall-clock time from submission to delivery); MRA (memory retrieval accuracy); PMU (peak RAM); and CPU Utilization. Results were compared against: (1) Single-Agent Baseline—equivalent system without multi-agent coordina-tion; and (2) No-Memory Baseline—full multi-agent system with persistent memory disabled.

VI. Results and Discussion

Overall Performance
Table IV presents aggregate metrics across all 100 tasks. AVIA outperformed the single-agent baseline [7] by 17.1 pp in TCR and 18.8 pp in ASR. Despite greater architectural complexity, AVIA achieved lower average response times (2.8 s vs. 3.4 s) due to parallel agent execution. The 8.2 pp TCR improvement over the no-memory baseline directly quantifies the memory architecture's contribution [13].

Table 4: Overall System Performance Metrics

Metric	AVIA Full	Single-Agent	No-Memory
Task Completion Rate	91.3%	74.2%	83.1%
Automation Success Rate	88.6%	69.8%	81.4%
Avg. Response Time	2.8 s	3.4 s	2.9 s
Memory Retrieval Acc.	93.2%	N/A	N/A
Peak RAM Usage	3.8 GB	2.1 GB	3.2 GB
Avg. CPU Utilization	34.7%	41.2%	35.1%

Performance by Task Category and Complexity
Information retrieval achieved the highest TCR (95.0%), reflecting the maturity of the Research Agent's web-search and summarization chain [10]. Calendar management performed strongly (93.0%) owing to structured scheduling operations. Code generation (88.0%) showed higher latency (3.9 s) due to

Table 5: Task Completion Rate By Category

Category	TCR	ASR	ART (s)
Information Retrieval	95.0%	92.3%	2.1
Communication Automation	90.0%	88.7%	2.4
Calendar Management	93.0%	89.4%	2.2
Code Generation	88.0%	85.1%	3.9
Content Publication	85.0%	82.3%	4.8

Table 6: Performance By Task Complexity Tier

Tier	N	TCR	ASR	ART (s)
Simple (1–2 skills)	40	97.5%	95.8%	1.4
Moderate (3–5 skills)	40	92.5%	89.3%	2.9
Complex (5+ skills)	20	75.0%	72.1%	5.6

iterative reasoning [9]. Content publication workflows, being the most complex multi-agent chains [11], exhibited the lowest TCR (85.0%) and highest latency (4.8 s). Performance exhibits a clear inverse relationship with task complexity: simple tasks achieved near-perfect completion (97.5%), while complex workflows scored 75.0%. Analysis of failures revealed 68% were attributable to cascading errors in long skill chains, motivating future work on error recovery and checkpoint mechanisms.

Memory System and Resource Utilization

Table 7: Memory System Performance

Metric	Result
Semantic Retrieval Recall@1	93.2%
Semantic Retrieval Recall@3	97.4%
Avg. Retrieval Latency	187 ms
Memory Write Throughput	≈420 entries/sec
FAISS Index Size (10K entries)	≈15 MB

Memory retrieval was evaluated on 50 memory-dependent tasks across previous sessions. The semantic retrieval sys-tem [14] achieved 93.2% recall@1, improving to 97.4% at recall@3. Average retrieval latency was 187 ms—negligible relative to overall task time. Peak RAM usage of 3.8 GB and 34.7% CPU utilization confirm AVIA operates within consumer laptop specifications. The dominant memory con-sumer is the quantized LLaMA-3-8B model [6] (≈2.6 GB); the remaining 1.2 GB is allocated to the Rust runtime, Go services, React frontend, Python layer, and FAISS index. These figures suggest deployment is feasible on 8 GB hardware using more aggressively quantized models.

Discussion

The results validate the core architectural hypotheses. Multi-agent design demonstrably outperforms single-agent execution [7], [11] by enabling concurrent processing and specialized domain reasoning. Persistent memory [13] provides a measur-able, quantifiable contribution to task completion, particularly

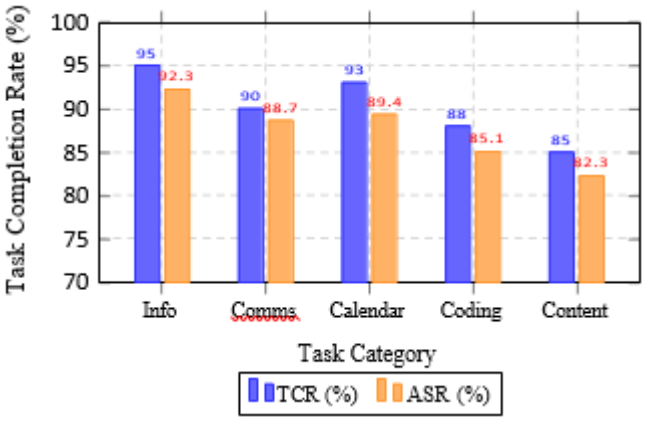

Fig. 2. TCR and ASR across task categories.

for cross-session continuity tasks. The primary limitation is degradation in complex multi-step workflows where cas-cading skill-chain errors propagate across agent boundaries. Secondary limitations include dependency on external API availability and the absence of fine-tuned domain-specific models for tasks such as legal document analysis.

VII. Conclusion

This paper presented AVIA, an Autonomous Virtual Intel-ligent Assistant for local-first, privacy-preserving digital task automation. By integrating a multi-agent coordination archi-tecture [11], [12], a hybrid persistent memory system [13], [14], a modular skill-based execution framework [9], [10], and an autonomous scheduling subsystem, AVIA demonstrates that practical, extensible, and capable assistant systems can be constructed outside commercial cloud platforms.

Evaluation across 100 tasks demonstrated a TCR of 91.3%, ASR of 88.6%, mean response latency of 2.8 s using local LLM inference [6], a 17.1 pp TCR improvement over the single-agent baseline [7], [8], and 93.2% memory recall@1 with sub-200 ms latency at 3.8 GB peak RAM.

Future directions include: (1) robust error recovery and checkpointing for complex skill chains; (2) fine-tuned domain-specific models for specialized task categories; (3) longitudinal multi-user deployments to assess memory management at scale; (4) federated or encrypted memory-sharing for collab-orative scenarios; and (5) benchmarking against AgentBench and GAIA evaluation frameworks.

Acknowledgment
The authors express sincere gratitude to Jayshri Kandekar for expert guidance and continued support throughout this research. The authors also thank the Department of Artificial Intelligence and Data Science at MET Institute of Engineer-ing, Nashik, for providing the computational resources that supported this work.

References

1. A. Lunden, “Apple’s Siri: A history of the world’s first mainstream AI assistant,” TechCrunch, 2021.
2. M. Lopez and A. Ranasinghe, “Comparative analysis of commercial virtual assistants: Alexa, Siri, Google Assistant, and Cortana,” in Proc. IEEE Int. Conf. on Artificial Intelligence and Knowledge Engineering (AIKE), 2020, pp. 1–8.
3. A. Vaswani et al., “Attention is all you need,” in Advances in Neural Information Processing Systems (NeurIPS), vol. 30, 2017.
4. T. Brown et al., “Language models are few-shot learners,” in Advances in Neural Information Processing Systems (NeurIPS), vol. 33, pp. 1877–1901, 2020.
5. OpenAI, “GPT-4 technical report,” arXiv preprint arXiv:2303.08774, 2023.
6. H. Touvron et al., “LLaMA 2: Open foundation and fine-tuned chat models,” arXiv preprint arXiv:2307.09288, 2023.
7. Significant Gravitas, “Auto-GPT: An Autonomous GPT-4 Experiment,” GitHub Repository, 2023. [Online]. Available: https://github.com/Significant-Gravitas/AutoGPT
8. S. Yao et al., “ReAct: Synergizing reasoning and acting in language models,” in Proc. Int. Conf. on Learning Representations (ICLR), 2023.
9. T. Schick et al., “Toolformer: Language models can teach themselves to use tools,” in Advances in Neural Information Processing Systems (NeurIPS), vol. 36, 2023.
10. H. Chase, “LangChain: Building applications with LLMs through composability,” GitHub Repository, 2022. [Online]. Available: https://github.com/langchain-ai/langchain
11. Q. Wu et al., “AutoGen: Enabling next-gen LLMs applications via multi-agent conversation,” arXiv preprint arXiv:2308.08155, 2023.
12. J. S. Park et al., “Generative agents: Interactive simulacra of human behavior,” in Proc. ACM Symposium on User Interface Software and Technology (UIST), 2023.
13. C. Packer et al., “MemGPT: Towards LLMs as operating systems,” arXiv preprint arXiv:2310.08560, 2023.
14. P. Lewis et al., “Retrieval-augmented generation for knowledge-intensive NLP tasks,” in Advances in Neural Information Processing Systems (NeurIPS), vol. 33, pp. 9459–9474, 2020.