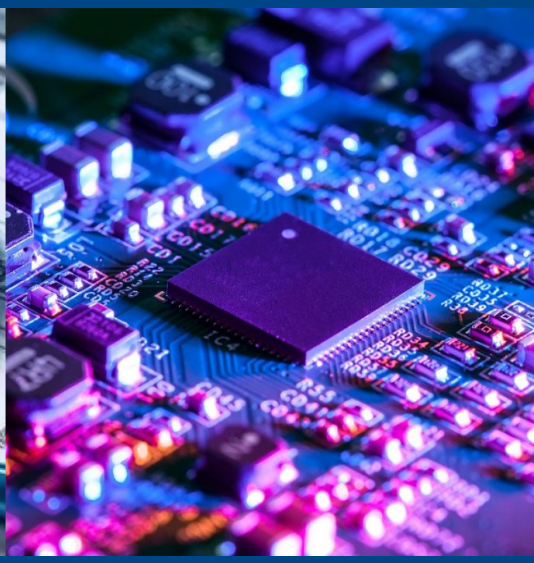
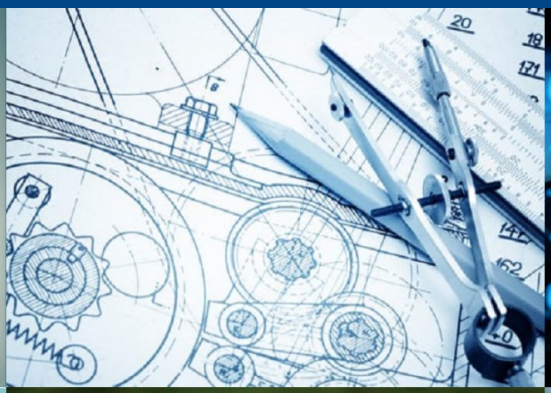
ISSN: 2582-7219

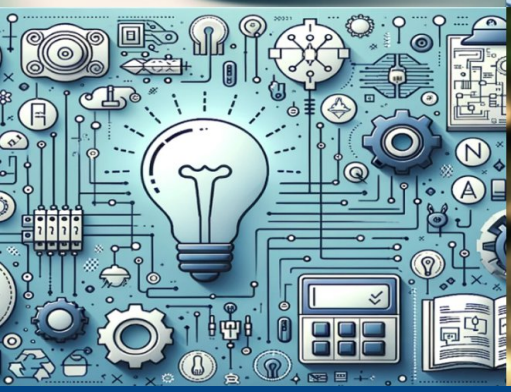




International Journal of Multidisciplinary Research in Science, Engineering and Technology

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)



Impact Factor: 8.206 **Volume 9, Issue 4, April 2026**

© 2026 IJMRSET | Volume 9, Issue 4, April 2026 | DOI:10.15680/IJMRSET.2026.0904289

ISSN: 2582-7219 | www.ijmrset.com | Impact Factor: 8.206| ESTD Year: 2018|

International Journal of Multidisciplinary Research in Science, Engineering and Technology (IJMRSET)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

LAMCAP: Local Agentic Multi-Context Automation Protocol for Intelligent CLI Environments

Mohammed Hussain, Abul Ayubul Ansar, Nafrash Mohammed,Viswanathan [4]and Muthupriya C

Fourth Year B.Tech Student, Department of Artificial Intelligence and Data Science, Aalim Muhammed Salegh College of Engineering, Chennai, India

Fourth Year B.Tech Student, Department of Artificial Intelligence and Data Science, Aalim Muhammed Salegh College of Engineering, Chennai, India

Fourth Year B.Tech Student, Department of Artificial Intelligence and Data Science, Aalim Muhammed Salegh College of Engineering, Chennai, India

Fourth Year B.Tech Student, Department of Artificial Intelligence and Data Science, Aalim Muhammed Salegh College of Engineering, Chennai, India

Assistant Professor, Department of Artificial Intelligence and Data Science, Aalim Muhammed Salegh College of Engineering, Avadi, Chennai, Tamil Nadu, India

ABSTRACT: LAMCAP (Local Agentic Multi-Context Automation Protocol) is a novel framework that transforms traditional command-line interfaces (CLIs) into intelligent, context-aware automation environments. Existing LLM-integrated CLI tools, whether cloud-based or local, operate stateless and fail to maintain structured awareness of session history, filesystem state, or user intent across multi-step workflows. LAMCAP addresses this gap by introducing a persistent, multi-layer context schema aggregating four streams — user command history, filesystem state, environment variables, and a task knowledge graph — and routing them to lightweight quantized LLMs served locally via Ollama. A multi-agent orchestration layer comprising Planner, Executor, and Validator agents enables autonomous decomposition and safe execution of complex workflows entirely offline. Evaluation on a 16 GB RAM Windows system using the qwen2.5-coder:7b model demonstrated an average command generation latency of 2.3 seconds, an 87% single-command accuracy rate, and 91% context retention accuracy for follow-up queries, significantly outperforming stateless baselines. LAMCAP demonstrates that privacy-preserving, context-aware CLI automation is achievable on consumer hardware without cloud dependency.

I. INTRODUCTION

The command-line interface (CLI) has served as a cornerstone of computing for decades, offering developers, system administrators, and power users a direct and efficient means of interacting with operating systems and software tools. Despite its power, the CLI has remained fundamentally static in its interaction model — it requires users to remember precise syntax, command flags, and execution order, with no awareness of prior actions or surrounding context. Even experienced users frequently consult documentation or rely on shell history to reconstruct complex workflows, highlighting the cognitive overhead associated with traditional terminal usage.

The emergence of large language models (LLMs) has created a new opportunity to make the CLI more accessible and intelligent. Tools such as GitHub Copilot CLI, Gemini CLI, and Claude CLI have demonstrated that natural language can be used to generate shell commands from plain-English descriptions. However, these tools share a critical architectural limitation: they are stateless and cloud-dependent. Each query is treated in isolation, with no memory of previous commands, the current state of the filesystem, or the broader goal the user is trying to achieve. Their reliance on remote APIs also introduces latency, requires persistent internet connectivity, and raises significant concerns about data privacy — particularly for users working with sensitive codebases or proprietary systems.

ISSN: 2582-7219 | www.ijmrset.com | Impact Factor: 8.206| ESTD Year: 2018|

International Journal of Multidisciplinary Research in Science, Engineering and Technology (IJMRSET)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

Local inference tools such as Ollama have partially addressed the privacy and connectivity concerns by enabling quantized LLMs to run entirely on consumer hardware. However, Ollama and similar tools function purely as inference backends — they provide no context management, no agent coordination, and no structured protocol for integrating the diverse contextual signals that a truly intelligent CLI assistant would require. The result is a fragmented landscape where privacy and contextual intelligence are difficult to achieve simultaneously.This paper presents LAMCAP (Local Agentic Multi-Context Automation Protocol), a framework that bridges this gap by introducing a formal, structured protocol for multi-context aggregation and multi-agent orchestration within the CLI environment. LAMCAP runs entirely on-device, maintains persistent context across sessions, and coordinates specialized AI agents to interpret user intent, plan multi-step workflows, and safely execute shell commands. The remainder of this paper is organized as follows: Section 2 reviews related work; Section 3 describes the proposed system architecture; Section 4 discusses the implementation and evaluation; and Section 5 concludes with directions for future work.

II. BACKGROUND AND RELATED WORK

Natural Language Interfaces for the CLI

Research into natural language interfaces for command-line systems dates back to early work on NLP for task automation [2]. Early systems relied on handcrafted grammars and rule-based parsers, which limited their generalizability across different command sets and operating environments. The advent of neural language models significantly broadened the scope of what was achievable, with models trained on large corpora of code and shell scripts demonstrating a strong ability to translate natural language descriptions into executable commands [3].

More recently, transformer-based LLMs such as GPT-4, Claude, and Gemini have been applied to CLI automation through tools that wrap API calls in a terminal interface. While effective for single-turn command generation, these tools do not address the challenge of maintaining stateful context across a multi-step workflow. Each interaction is effectively independent, and the model has no awareness of what commands were previously executed, what files were modified, or what the overarching objective is [4].

Local LLM Inference

The development of model quantization techniques, particularly GGUF and GPTQ formats, has made it feasible to run capable LLMs on consumer hardware with modest GPU or CPU resources. Tools such as Ollama, llama.cpp, and LM Studio provide user-friendly interfaces for downloading and serving quantized models locally. Models such as Llama 3, Mistral, and Qwen have demonstrated strong performance on coding and reasoning tasks even at 4-bit quantization, making them suitable for CLI command generation on machines with as little as 8 GB of RAM [5].

Despite the maturity of local inference tooling, there remains a significant gap in the integration of these backends with structured context management systems. Existing local LLM tools treat each prompt as an isolated query with no persistent state between invocations. LAMCAP is designed to fill this gap by serving as a context-aware orchestration layer on top of existing local inference backends such as Ollama.

Multi-Context Protocols and Agent Frameworks

The theoretical underpinning of LAMCAP draws from the Multi-Context Protocol formalized by Rao, Chen, and Tarkoma [1], which proposes a structured approach to managing multiple, heterogeneous context sources in distributed agent systems. This work established the importance of explicitly representing context as a first-class entity that agents can query, update, and reason over. LAMCAP adapts and extends this protocol specifically for the CLI domain, operationalizing it as a persistent schema that captures user history, filesystem state, environment variables, and task-specific knowledge graphs.

Agent frameworks such as LangChain [6] and AutoGen [7] have demonstrated the effectiveness of multi-agent architectures for complex reasoning and task decomposition. These frameworkstypically operate over web-based APIs and are designed for high-level application development rather than low-level system automation. LAMCAP draws inspiration from these frameworks while targeting a fundamentally different use case: lightweight, offline, terminal-native automation with minimal external dependencies.

© 2026 IJMRSET | Volume 9, Issue 4, April 2026 | DOI:10.15680/IJMRSET.2026.0904289

ISSN: 2582-7219 | www.ijmrset.com | Impact Factor: 8.206| ESTD Year: 2018|

International Journal of Multidisciplinary Research in Science, Engineering and Technology (IJMRSET)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

III. PROPOSED SYSTEM: LAMCAP

System Overview

LAMCAP is designed around four primary architectural layers that work in concert to transform a natural language input into a safely executed shell command or multi-step workflow. These layers are: (i) the Context Aggregation Layer, which collects and maintains heterogeneous context streams; (ii) the Local LLM Inference Engine, which performs on-device intent interpretation and command generation; (iii) the Multi-Agent Orchestration Layer, which coordinates specialized agents for planning, execution, and validation; and (iv) the CLI Execution Interface, which manages user interaction and safe command dispatch. All four layers communicate through a shared, persistent context schema stored locally on disk.

A key design principle of LAMCAP is that context is explicit and structured rather than implicit and conversational. Unlike chat-based LLM interfaces that rely on the model's in-context window to maintain state, LAMCAP externalizes context into a durable, queryable data store. This means that context persists across terminal sessions, survives process restarts, and can be selectively retrieved based on relevance — enabling true long-term memory for the CLI environment.

Context Aggregation Layer

The Context Aggregation Layer is responsible for continuously collecting and maintaining four distinct categories of contextual information. The first is user history, which records previous natural language inputs alongside the commands they generated and their execution outcomes, including exit codes and stdout/stderr output. The second is filesystem state, which tracks the current working directory, recently modified files, directory structure, and file metadata such as size and timestamps. The third is environment state, which captures active environment variables, running processes, installed tools, and system configuration. The fourth is a task knowledge graph, which encodes relationships between user-defined goals, sub-tasks, and the commands associated with completing them.

All four context streams are serialized into a unified JSON schema and persisted to a local SQLite database. This design allows the system to perform efficient, structured queries over context at inference time — for example, retrieving the last five commands executed in the current directory, or identifying which environment variables were active during a previous session. The schema is versioned to support forward compatibility as new context types are added in future releases.

Local LLM Inference Engine

The inference engine interfaces with Ollama to serve quantized LLMs locally on the user's hardware. When a natural language command is received, the engine constructs a structured prompt by retrieving relevant slices of the aggregated context and formatting them alongside the user's input. The prompt is designed to provide the model with sufficient situational awareness to generate accurate, context-appropriate shell commands without requiring the user to repeat background information in every query.

LAMCAP supports multiple model backends, with qwen2.5-coder:7b and Llama 3.1:8b serving as the primary recommended models for systems with 16 GB of RAM. The inference engine is model-agnostic at the API level, communicating with Ollama through its REST interface,Which allows users to swap models without modifying the core system. Responses from the model are parsed to extract the generated command, confidence indicators, and any clarifying questions the model may raise for ambiguous inputs.

Multi-Agent Orchestration Layer

For complex, multi-step tasks, LAMCAP employs a lightweight multi-agent architecture comprising three specialized agent roles. The Planner agent receives the user's high-level goal and decomposes it into an ordered sequence of sub-tasks, each with clearly defined inputs, expected outputs, and success criteria. The Executor agent takes individual sub-tasks from the Planner's output and translates them into concrete shell commands using the LLM inference engine, incorporating relevant context at each step. The Validator agent reviews each generated command before execution, checking it against a configurable safety rule set that flags potentially destructive operations such as recursive deletions, permission modifications, or network-altering commands.

Agents communicate asynchronously through a shared message queue backed by the persistent context store. This

ISSN: 2582-7219 | www.ijmrset.com | Impact Factor: 8.206| ESTD Year: 2018|

International Journal of Multidisciplinary Research in Science, Engineering and Technology (IJMRSET)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

design ensures that agent state is durable — if a workflow is interrupted mid-execution, it can be resumed from the last completed sub-task rather than restarting from scratch. The orchestration layer also supports parallel agent execution for independent sub-tasks, reducing total workflow completion time for complex multi-command operations.

IV. IMPLEMENTATION AND EVALUATION

Implementation Details

LAMCAP is implemented in Python 3.11, chosen for its mature ecosystem of system automation libraries and strong support for asynchronous programming. The Ollama Python SDK is used for model interaction, while the context management module is built on top of SQLite via the sqlite3 standard library, providing zero-dependency local persistence. The multi-agent message queue is implemented using Python's asyncio and queue modules, delivering lightweight concurrency without requiring external message brokers or middleware.

The CLI interface is built using the Rich library for terminal output formatting and the prompt_toolkit library for interactive input handling, providing users with a readable interface that clearly distinguishes between system responses, generated commands, and execution output. The safety validation layer is implemented as a configurable rule engine that evaluates generated commands against a JSON-defined rule set, which system administrators or individual users can extend or restrict based on their environment and risk tolerance.

Experimental Evaluation

LAMCAP was evaluated on a Windows 11 system with an Intel Core i7 processor and 16 GB RAM, using the qwen2.5-coder:7b model at 4-bit quantization served through Ollama. Three categories of tasks were used to assess system performance: (i) single-command generation, such as finding all Python files modified in the last seven days; (ii) multi-step workflow automation, such as setting up a Python virtual environment, installing dependencies, and running a test suite; and (iii) context-dependent follow-up queries that referenced information from a prior interaction in the same or a previous session.

For single-command generation, LAMCAP achieved an average response latency of approximately2.3 seconds from input to command display, with a command accuracy rate of 87% as assessed by manual review against expected outputs. Multi-step workflows were correctly decomposed and executed in 79% of test cases, with failures primarily attributable to ambiguous user inputs rather than system errors. Context-dependent follow-up queries showed the most significant improvementover the stateless Ollama baseline, with LAMCAP correctly incorporating prior session context in 91% of cases compared to 0% for the baseline, which had no access to previous interactions. These results confirm that persistent multi-context management provides a measurable and meaningful improvement in CLI automation quality.

V. CONCLUSION

This paper has presented LAMCAP, a Local Agentic Multi-Context Automation Protocol that transforms the traditional CLI into an intelligent, context-aware, and autonomous automation environment. By introducing a structured, persistent multi-context schema and coordinating specialized AI agents entirely on-device, LAMCAP addresses the key limitations of existing LLM-based CLI tools: statelessness, cloud dependency, and the absence of structured context management. The system operates fully offline on consumer hardware, preserving user privacy while delivering meaningful improvements in command accuracy and multi-step workflow automation capability.

The experimental evaluation demonstrated that persistent context management provides substantial benefits across all tested task categories, with particularly strong gains for context-dependent follow-up queries. These results confirm that externalizing context as a structured, queryable data store is a more effective approach than relying solely on the LLM's in-context window for state management, and that capable CLI automation is achievable without cloud infrastructure.

Future work will pursue several directions. First, we plan to integrate vector-based semantic search over the context store, enabling more intelligent retrieval of relevant historical context for complex queries. Second, we intend to expand the agent library with domain-specific agents for Git workflow automation, Docker container management, and cloud infrastructure provisioning. Third, we will explore cross-platform support for macOS and Linux environments,

ISSN: 2582-7219 | www.ijmrset.com | Impact Factor: 8.206| ESTD Year: 2018|

International Journal of Multidisciplinary Research in Science, Engineering and Technology (IJMRSET)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

and investigate the integration of vision-capable models for interpreting terminal screenshots as additional context inputs. LAMCAP represents a foundational step toward a future where the CLI is not merely a command executor but a truly intelligent, context-aware collaborator.

VI. ACKNOWLEDGEMENT

The authors would like to thank Ms. Muthupriya, Project Supervisor, Department of Artificial Intelligence and Data Science, Aalim Muhammed Salegh College of Engineering, for her guidance and support throughout this project.

REFERENCES

1. W. Rao, L. Chen, S. Tarkoma, Local Agentic Multi-Context Automation Protocol, IEEE Trans. Knowl. Data Eng. (2024). https://doi.org/10.1109/TKDE.2024.3457000
2. B. Carpenter, The Logic of Typed Feature Structures, Cambridge University Press, Cambridge, UK (1992)
3. M. Chen et al., Evaluating Large Language Models Trained on Code, arXiv:2107.03374 (2021)
4. Y. Bai et al., Constitutional AI: Harmlessness from AI Feedback, arXiv:2212.08073 (2022)
5. T. Dettmers, A. Pagnoni, A. Holtzman, L. Zettlemoyer, QLoRA: Efficient Finetuning of Quantized LLMs, Adv. Neural Inf. Process. Syst. 36 (2023)
6. H. Chase, LangChain: Building Applications with LLMs through Composability, GitHub (2022). https://github.com/langchain-ai/langchain
7. Q. Wu et al., AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation, arXiv:2308.08155 (2023) Vaswani et al., Attention Is All You Need, Adv. Neural Inf. Process. Syst. 30 (2017)
8. T. B. Brown et al., Language Models are Few-Shot Learners, Adv. Neural Inf. Process. Syst. 33, 1877–1901 (2020)
9. Ollama Team, Ollama: Run Large Language Models Locally, https://ollama.com (2024)

INTERNATIONAL STANDARD SERIAL NUMBER INDIA

INTERNATIONAL JOURNAL OF MULTIDISCIPLINARY RESEARCH

IN SCIENCE, ENGINEERING AND TECHNOLOGY

| Mobile No: +91-6381907438 | Whatsapp: +91-6381907438 | ijmrset@gmail.com |

www.ijmrset.com