Grenze International Journal of Engineering and Technology, June Issue

Context-Aware Task Automation using Offline LLM

J. Nagaraju[1], K. Lakshmi Gayathri[2], R. Bhanu Venkat[3] and B. Omkar Lakshmi Lokesh[4]
[1-4]Lakireddy Bali Reddy College of Engineering, Mylavaram, India
Email: jnrcse1@gmail.com, gayatrikoppisetti479@gmail.com, bhanuvenkat204@gmail.com, lokeshbommasani@gmail.com

***Abstract*— The rapid growth of intelligent automation systems has increased the demand for tools capable of understanding natural language instructions and executing complex tasks autonomously. Many existing solutions rely heavily on cloud-based large language models (LLMs), raising concerns regarding privacy, latency, and internet dependency. This paper presents a context-aware task automation system that operates entirely offline using the LLaMA-2 large language model.**

The proposed system interprets natural language commands and converts them into structured task plans decomposed into executable subtasks in JSON format. A dual-context management framework maintains both persistent user context and dynamic task context, enabling adaptive decision-making. The system integrates OmniParser for visual UI parsing and Tesseract OCR for text extraction from real-time screenshots. A feedback-driven execution loop continuously monitors the UI state, validates actions, and replans steps when discrepancies are detected. Experimental evaluation across 16 multi-step desktop tasks demonstrates a task success rate of 92.1%, covering application launching, website navigation, form filling, and message automation.

***Index Terms*— Offline LLM, Task Automation, Context-Aware Systems, UI Parsing, Desktop Automation, LLaMA-2, Human–Computer Interaction.**

I. INTRODUCTION

The increasing integration of artificial intelligence into everyday computing has enhanced the capabilities of automated assistants and intelligent software agents. Large language models (LLMs) such as GPT-4 and LLaMA-2 have shown strong performance in reasoning, planning, and contextual understanding, enabling intelligent systems that can interpret user instructions and autonomously perform complex tasks on computing devices.

Traditional task automation has primarily relied on rule-based scripting and Robotic Process Automation (RPA) tools. These approaches require predefined workflows and manual configuration, limiting flexibility when handling dynamic or unstructured user requests. The advent of LLMs has enabled systems that translate natural language instructions directly into executable actions, fostering a more intuitive and flexible human–computer interaction.

Despite these advancements, most existing LLM-based automation systems depend heavily on cloud-based models, introducing challenges such as latency, privacy concerns, and the necessity for continuous internet connectivity. In privacy-sensitive environments, transmitting user instructions and system data to remote servers is unacceptable. This paper proposes a context-aware desktop task automation framework operating fully offline using LLaMA-2, featuring a dual-context management framework and visual interface parsing via OmniParser.

Grenze ID: 01.GIJET.12.2.519_1

The main contributions of this work are:

- Design and implementation of a completely offline task automation framework powered by a locally deployed LLaMA-2 model, eliminating reliance on cloud-based inference.
- Integration of screenshot-based visual understanding using OmniParser for application-independent GUI interaction.
- A dual-context management framework combining persistent user context and dynamic task context for adaptive and privacy-preserving decision-making.
- A closed-loop, feedback-driven execution strategy that improves robustness and enables recovery from execution errors.

II. Literature Survey

Automation of desktop and mobile tasks has been widely studied, but traditional systems rely on rigid rule-based scripts or DOM-level access, lacking adaptability for dynamic interfaces and natural language instructions. RPA platforms automate repetitive workflows but struggle with unstructured input. Recent LLM advances have enabled more intelligent, context-aware automation agents.

A. LLM-Brained GUI Agents

Surveys on LLM-driven GUI agents highlight the transition from script-based automation to flexible, adaptive systems capable of multi-step task execution and higher-level reasoning across web, mobile, and desktop platforms [3][8].

B. Vision-Based and Multimodal Approaches

Vision Tasker introduces a two-stage approach: UI screenshots are converted into natural language via visual recognition, then an LLM plans next actions step-by-step, reducing reliance on accessibility view hierarchies [13]. Multimodal LLM-powered GUI agents integrating visual and textual perception address key challenges including element localization and long-horizon planning [4][11].

C. LLM-Enhanced RPA and Context-Aware Systems

Recent research integrates LLMs with RPA for text extraction and workflow automation, improving efficiency particularly for OCR tasks and unstructured data [6][10]. Context-aware RPA embeds natural language understanding into automation systems for free-text documents and dynamic forms. However, most existing systems rely on cloud-based models or lack offline execution capabilities, a gap directly addressed by the proposed system.

III. System Architecture

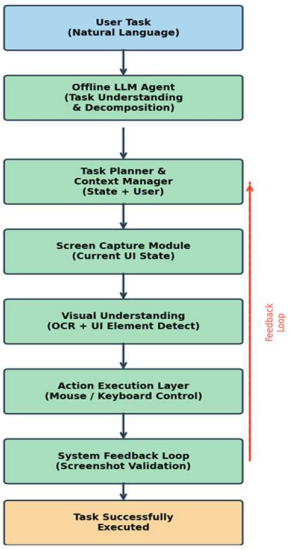

Fig. 1. System architecture of the proposed offline task automation framework

The proposed system employs a closed-loop, agent-based architecture as shown in Fig. 1. The process begins with a natural language task forwarded to offline LLaMA-2, which runs locally, ensuring all sensitive data remains on the user's device. The model interprets the instruction, identifies intent, and decomposes the task into manageable sequential steps.

A task planning and context management module maintains two types of contextual memory: persistent user context storing long-term preferences, workflows, and authentication knowledge; and dynamic task context tracking current progress, completed steps, intermediate goals, and errors.

The system periodically captures screenshots analyzed by Tesseract OCR and OmniParser to provide a structured GUI representation incorporating both semantic and spatial information. are identified, the system applies recovery strategies or safely terminates the process. A secure context file stored locally on the user's device holds sensitive user-specific information and is never transmitted outside the system, dynamically updated as new tasks are completed.

IV. Proposed Methodology

The objective is to enable autonomous execution of complex desktop tasks based on natural language instructions while operating entirely offline. The framework integrates language-based reasoning, visual interface perception, contextual memory management, and a feedback-driven control mechanism.

A. Task Input and Offline Reasoning Engine

The automation process begins with a natural language task accepted via command-line or GUI prompt, forwarded to the offline LLaMA-2 reasoning engine. The engine performs intent recognition, contextual understanding, and step decomposition. It generates a structured interpretation encoded in JSON format for downstream processing, handling diverse instructions without requiring predefined command templates.

B. Task Planning and Context Management

The task planning module decomposes the high-level goal into manageable subtasks. Predefined handlers manage repetitive operations such as launching browsers, reducing computational overhead. The context manager maintains three layers: (1) Persistent User Context—long-term preferences and credentials encrypted locally; (2) Dynamic Task Context—tracks completed steps, errors, and recovery attempts; and (3) Secure Context File—local knowledge repository for sensitive information, restricted to local processes.

C. Visual Perception and Action Execution

Screenshots are captured periodically using platform-agnostic libraries. Tesseract OCR extracts textual content such as button labels, field placeholders, and error messages. OmniParser identifies structural UI elements including buttons, input fields, icons, and menus, outputting element type, position coordinates, and associated text. The action selection module evaluates the GUI analysis, task plan, and context to determine the next operation—mouse clicks, keyboard input, scrolling, or window navigation—executed through PyAutoGUI and native OS APIs.

D. Closed Feedback Loop

After each action, a new screenshot is captured and validated against the expected GUI state. If validation succeeds, the system updates context and proceeds to the next subtask. If discrepancies are detected, recovery mechanisms trigger incremental replanning, adjusted action parameters, or safe termination if errors persist beyond a configurable threshold. This closed-loop design prevents error accumulation and ensures robustness against dynamic interface changes.

V. Algorithm

Algorithm 1 Context-Aware Offline Task Automation with Feedback Loop
Require: Natural language task T
Ensure: Task completed successfully or safely terminated

- Load persistent user context Cu from secure context file
- Initialize dynamic task context Ct
- Decompose T into sub-tasks {S1, S2, ..., Sn} using offline LLaMA-2
- while not all sub-tasks completed do
- Capture screen state St; Extract text via OCR; Parse UI via OmniParser

- Update Ct with current GUI state
- Infer next action at ← π(T, St, Cu, Ct) using LLaMA-2
- if at is valid then
- Execute at via OS interaction layer
- Capture post-action screen St+1; Validate against expected outcome
- if validation succeeds then mark Si complete; update Ct and Cu
- else increment recovery counter; apply recovery/replanning strategy
- end if
- else request replanning of remaining sub-tasks from LLaMA-2
- end if
- if recovery counter exceeds threshold then terminate; break
- end while
- Return overall execution status

VI. IMPLEMENTATION

A. Development Environment

The system was implemented on Windows 10/11 and Ubuntu 22.04. Python 3.11 was used as the primary language. Key libraries include PyTorch for model inference, Open CV for image processing, and PyAutoGUI for GUI automation, and Tesseract OCR for text extraction. LLaMA-2 was deployed locally using the Hugging Face Transformers library with GPU acceleration where available, supporting CPU-only execution for lower-specification machines. The test hardware comprised an Intel i7-12700 CPU, NVIDIA RTX 3060 GPU, and 32 GB RAM.

B. Context Management and Logging

Persistent user context is stored as an AES-encrypted JSON file ensuring sensitive data is protected at rest. Dynamic task context is maintained in-memory and updated after each subtask. The secure context file is read by LLaMA-2 at planning time and written back after successful task execution. A logging module records the full task execution sequence, subtask completion status, recovery attempts, and context file updates, all stored locally without transmitting any telemetry externally.

C. Example Task Execution

Consider the task: "Open Brave, navigate to WhatsApp Web, log in if necessary, send a message to Bhanu saying 'Hi! How are you?', then open Google Calendar, schedule a meeting for tomorrow at 3 PM titled 'Project Update' inviting Bhanu and Akshay, and save a screenshot locally." LLaMA-2 decomposes this into nine sequential subtasks. The secure context file provides contact details and session tokens. OmniParser and OCR identify relevant UI elements at each step. Conditional authentication is handled automatically. The feedback loop validates each action and replans on failure. This demonstrates cross-application automation, offline reasoning, visual adaptation, error recovery, and complete privacy preservation

VII. EXPERIMENTAL EVALUATION

The proposed system was evaluated on 16 complex, multi-step tasks across four categories to measure task success rate, efficiency, robustness, and adaptability under varying interface conditions.

A. Evaluation Metrics and Test Setup

The system was assessed using: Task Completion Rate (%)—tasks completed without manual intervention; Average Steps—mean subtasks per task; Average Execution Time (s)—including recovery steps; and Number of Recoveries—feedback-driven replanning actions per task. Tasks spanned Web Navigation (5), Form Filling (4), File Operations (3), and Application Control (4), including complex multi-step workflows such as WhatsApp Web messaging and Google Calendar scheduling.

B. Results

Table I presents the detailed evaluation results. The system achieves high success rates across all categories. File Operations achieved a perfect 100% success rate with zero recoveries, reflecting the deterministic nature of file system interactions. Form Filling required the most recovery actions due to dynamic form layouts and AJAX-driven UI updates

TABLE I: TASK-LEVEL EVALUATION RESULTS

Task Category	Tasks	Success (%)	Avg. Steps	Avg. Time (s)	Recoveries
Web Navigation	5	92.0	7.4	38.2	0.8
Form Filling	4	87.5	9.1	52.6	1.3
File Operations	3	100.0	5.0	24.7	0.0
App. Control	4	90.0	6.8	31.4	0.6
Overall	16	92.1	7.1	36.8	0.7

C. Visual Results and Feedback Loop Demonstration

Figs. 2–5 show bar charts for task success rate, average steps, average execution time, and recovery actions per category. Figs. 6–10 illustrate the step-by-step WhatsApp Web execution demonstrating the system's ability to detect dynamic UI elements, perform conditional login, identify contacts via OCR and OmniParser, and validate each action through the feedback loop.

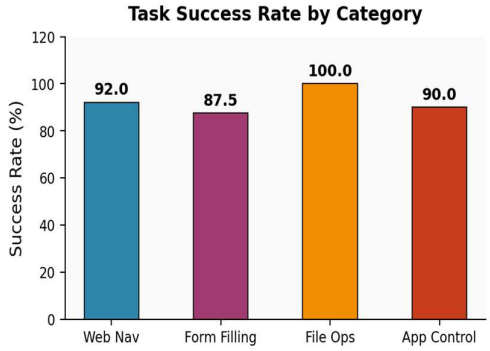

Fig. 2. Task success rate by category (%)

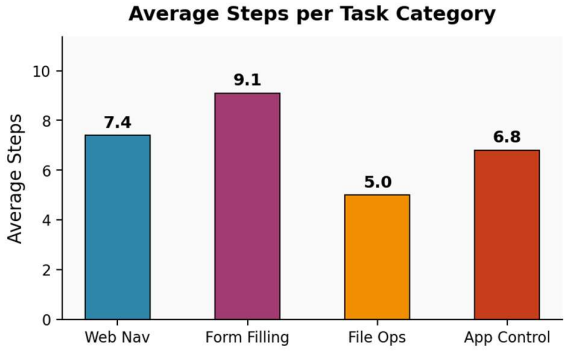

Fig. 3. Average steps per task category

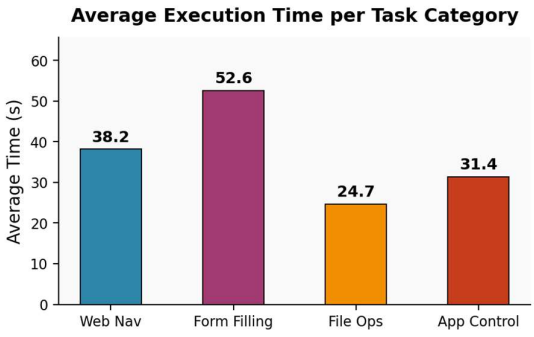

Fig. 4. Average execution time per task category (seconds)

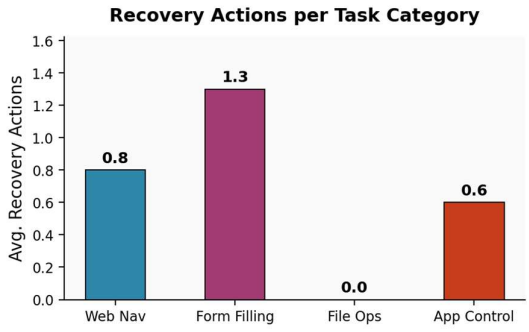

Fig. 5. Recovery actions triggered per task category

VIII. DISCUSSION

The 92.1% overall task success rate confirms that offline LLM reasoning combined with dual-context management and visual feedback-driven execution is sufficient to handle broad real-world desktop automation scenarios. The average execution time of 36.8 seconds per task, including any recovery steps, is comparable to human execution of equivalent multi-step workflows. Privacy is fully preserved—all task instructions, screen content, and credentials are processed and stored entirely on the user's local device with no external transmission.

A. Limitations

OCR errors occasionally affected text recognition on low-contrast or stylized UI elements. Dynamic web interfaces with rapidly changing DOM structures sometimes required multiple recovery attempts. Tasks involving non-standard or highly customized applications may need additional OmniParser configuration for accurate UI element detection. LLaMA-2 inference latency, while acceptable, is higher than cloud-based alternatives in time-critical scenarios.

B. Future Work

Future directions include: integrating multi-modal reasoning combining text, image, and layout features; improving OCR accuracy through domain-adaptive pre-processing; extending support to mobile and virtualized desktop environments; and incorporating predictive planning to anticipate future UI states and reduce recovery actions.

IX. Conclusion

This paper presented a context-aware, offline task automation system leveraging LLaMA-2, OmniParser, Tesseract OCR, and a feedback-driven control loop to reliably perform complex desktop tasks without internet connectivity. Experimental evaluation across 16 diverse multi-step tasks demonstrated a 92.1% task success rate with an average of fewer than one recovery action per task, confirming that the integration of offline language reasoning, visual perception, dual-context management, and closed-loop feedback control enables reliable, privacy-preserving, and flexible desktop task automation that significantly outperforms traditional open-loop approaches.

Acknowledgment

The authors thank the Department of Artificial Intelligence and Data Science, Lakireddy Bali Reddy College of Engineering (Autonomous), Mylavaram, for providing the computational resources and research environment that supported this work.

References

[1] Anthropic. 2023. Claude. https://www.anthropic.com/product.
[2] A. Azim, O. Riva, and S. Nath. 2016. ULink: Enabling User-Defined Deep Linking to App Content. MobiSys '16. ACM, 305–318.
[3] A. Burns, D. Arsan, S. Agrawal, et al. 2022. A Dataset for Interactive Vision Language Navigation with Unknown Command Feasibility. ECCV.
[4] M. Chen, J. Tworek, H. Jun, et al. 2021. Evaluating Large Language Models Trained on Code. arXiv:2107.03374.
[5] W.-L. Chiang, Z. Li, et al. 2023. Vicuna: An Open-Source Chatbot Impressing GPT-4. https://lmsys.org/blog/2023-03-30-vicuna/
[6] A. Chowdhery, S. Narang, J. Devlin, M. Bosma, et al. 2022. PaLM: Scaling Language Modeling with Pathways. arXiv:2204.02311
[7] K. Cobbe, V. Kosaraju, M. Bavarian, et al. 2021. Training Verifiers to Solve Math Word Problems. arXiv:2110.14168.
[8] X. Deng, Y. Gu, B. Zheng, et al. 2023. Mind2Web: Towards a Generalist Agent for the Web. arXiv:2306.06070.
[9] C.-Y. Hsieh, C.-L. Li, C.-K. Yeh, et al. 2023. Distilling Step-by-Step! arXiv:2305.02301.
[10] P. C. Humphreys, D. Raposo, T. Pohlen, et al. 2022. A data-driven approach for learning to control computers. ICML. PMLR, 9466–9482.
[11] T. Kojima, S. S. Gu, M. Reid, et al. 2022. Large Language Models are Zero-Shot Reasoners. NeurIPS 2022, vol. 35, pp. 22199–22213.
[12] J. Lee, H. Kim, S. Kim, et al. 2022. A-Mash: Providing Single-App Illusion for Multi-App Use. MobiCom '22. ACM, 690–702.
[13] Y. Li and Y. Li. 2023. Spotlight: Mobile UI Understanding using Vision-Language Models. ICLR.
[14] T. J.-J. Li, Y. Li, F. Chen, and B. A. Myers. 2017. Programming IoT devices by demonstration using mobile apps. IS-EUD 2017. Springer, 3–17.
[15] T. J.-J. Li and O. Riva. 2018. Kite: Building Conversational Bots from Mobile Apps. MobiSys '18. ACM, 96–109.
[16] Y. Li, J. He, X. Zhou, Y. Zhang, and J. Baldridge. 2020. Mapping Natural Language Instructions to Mobile UI Action Sequences. ACL 2020.
[17] Y. Li, G. Li, L. He, J. Zheng, H. Li, and Z. Guan. 2020. Widget Captioning. EMNLP, pp. 5495–5510.
[18] Y. Li, G. Li, X. Zhou, M. Dehghani, and A. A. Gritsenko. 2022. VUT: Versatile UI Transformer. ICLR.
[19] Y. Li and O. Riva. 2021. Glider: A Reinforcement Learning Approach to Extract UI Scripts. SIGIR '21. ACM, 1420–1430.
[20] Y. Li, H. Wen, W. Wang, et al. 2024. Personal LLM Agents. arXiv:2401.05459.