International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 09 Issue: 04 | April - 2025 SJIF Rating: 8.586 ISSN: 2582-3930

Improving The Automation and Autonomy Efficiency by Employing LLM as Agents

[1]Akash Chaurasiya, [2]Rishabh Agarwal, [3]Tushar Tiwari, [4]Kalash Gupta,[5]Anand Singh Badal,[6]Abhishek Saxena
[1]Information Technology, Bansal Institute of Engineering and Technology
[2]Information Technology, Bansal Institute of Engineering and Technology
[3]Information Technology, Bansal Institute of Engineering and Technology
[4]Information Technology, Bansal Institute of Engineering and Technology
[5]Information Technology, Bansal Institute of Engineering and Technology
[6]Information Technology, Bansal Institute of Engineering and Technology

Abstract-Large Language Models (LLMs) have proved to have spectacular capability in natural language understanding and generation but with growing value across a range of automation categories. They remain behind the current performance of state-of-the-art commercial models such as ChatGPT and GPT-4 even when utilized to handle difficult real-world problems. In order to work as fully fledged intelligent agents, LLMs will have to surpass language skills alone and perform complex task planning, long-term memories, context-independent reasoning, and have the facility for communication with outside tools. This paper presents one unified framework to improve the autonomy and efficacy of LLM-based agents. The core concept of our research is to design agent-dependent datasets and use the LLM as the core decision-making unit. By fine-tuning LLMs on agent-dependent datasets through supervised learning, especially in the scenario of smaller parameter models, we see a sharp reduction in hallucinations, format errors, and execution errors.

We further improve agent performance with methods like multi-path reasoning and task decomposition that partition challenging tasks into less complex subtasks and thus increase reliability and flexibility. Our system is tested on five realistic automation tasks and shows significant improvements in task correctness, fault tolerance, and overall throughput. This article highlights the possibility of LLMs transforming when redesigned as autonomous agents, providing a future direction for intelligent scalable automation systems. They are able to learn to fit into new circumstances and minimize the need for constant human intervention.

Keywords: Large Language Models (LLMs), Autonomous Agents, Intelligent Automation, Task Planning, Multi-Path Reasoning, Dataset Fine-Tuning, Task Decomposition, Contextual Adaptation, Automation Efficiency

1. INTRODUCTION

Large Language Models (LLMs) have greatly accelerated the progress of artificial intelligence at an unprecedented rate, thanks to their exceptional skill in natural language comprehension, contextual reasoning, and low-shot learning abilities. These models, built from large text corpora and containing billions of parameters, form the basic foundation for a vast range of intelligent systems across diverse domains. They are being increasingly investigated not only as passive responders, but also as active elements in autonomous agents that can undertake multi-step reasoning, decision-making, and complex task execution.

In spite of such advancements, applying LLMs to actual real-world automation and autonomy is not an easy thing to do. The majority of current automation frameworks are deterministic in nature, coded hard logic, or scripted and stiff environments that cannot generalize for dynamic or failure-prone settings. While flexibility and generalizability are obtained by LLMs, the default setting is not toward autonomous decision-making or adaptive task planning. They lack permanent memory, are susceptible to hallucinating facts, and display unstable behavior in handling external worlds—especially when expert scaffolding or fine-tuning is lacking.

State-of-the-art commercial LLM-based systems, such as OpenAI's ChatGPT and Anthropic's Claude, integrate external tools, memory, and reinforcement signals to perform as general-purpose agents. However, these capabilities are not readily available in open-source LLMs or smaller parameter models, which are more accessible to the broader research community. Bridging

this gap requires targeted improvements in both architecture and training methodology.

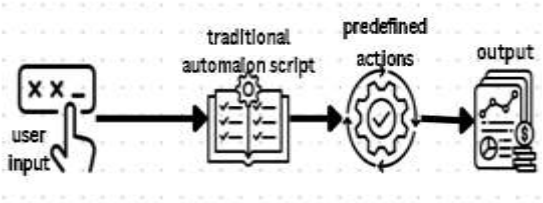

Fig-1: automation by traditional method

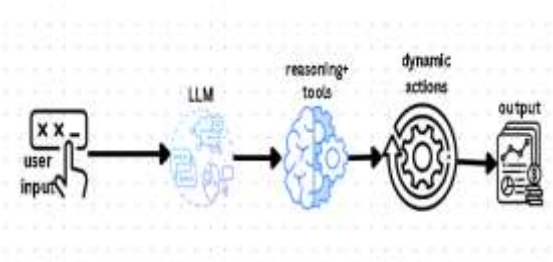

Fig-2: automation using LLM as agents

Recent research has examined means like tool use extension, ReAct-type prompting [1], memory-guided planning, and multi-agent coordination (e.g., CAMEL [2], Reflexion [3]) to further enhance the autonomous abilities of LLMs. These methods have useful contributions but tend to be based on prompt engineering instead of structural fine-tuning or task-specific learning. In addition, the majority of these methods presuppose static or ideal situations, and do not handle poor performance in error-prone, real-world automation situations where agents need to recover, adapt, and re-plan.

In this work, we introduce a holistic framework that improves LLMs as automation and autonomy agents. Our threefold contributions are:

Agent-Specific Dataset Construction – We create and curate domain-specific datasets for agent-based automation tasks so that LLMs can learn task execution patterns, tool usage, and context handling from structured supervision. [1]. Supervised Fine-Tuning for Lighter Models – By fine-tuning smaller parameter LLMs on such datasets, we greatly minimize hallucinations, format errors, and enhance task consistency—making lighter models more acceptable for autonomous systems. [2]. Reasoning Strategies for Task Efficiency – We incorporate multi-path reasoning and hierarchical task decomposition into the workflow of the agent, allowing the system to divide and solve difficult problems in modular phases. [3]

We assess our method on five different agent tasks: desktop automation, web-based interaction, dynamic workflow control, and multi-agent coordination. Experimental results demonstrate large improvements in task success rate, fault recovery, and throughput relative to baseline models and zero-shot prompting approaches. This piece highlights the increasing promise of LLMs as the building blocks for intelligent automation systems. By outfitting them with structured data, fine-tuning, and reasoning tactics, we take a step toward the vision of scalable, adaptive, and general-purpose autonomous agents that can function effectively in a wide range of environments

2. BACKGROUND AND RELATED WORK

Large Language Models like GPT-3, PaLM, and LLaMA are now flagship artificial intelligence technologies because of their strong generalization power over many tasks. By training on humongous amounts of data, the models are excellent at natural language understanding, reasoning, and even tool problem-solving. But using LLMs as standalone agents—able to execute and learn independently on demanding tasks—remains a research field that is still evolving. As highlighted by Zhao et al. [4], agent-based LLMs require modular reasoning, tool usage, and long-term memory to perform beyond mere language modelling. Legacy automation systems depend on hardcoded rules, deterministic operations, and rule-based engines with bounded rule bases that are very task-specific and rigid in variable environments. They work under stable conditions but do not manage variability, ambiguity, or failure recovery. LLM-based agents bring in flexibility by reading natural language commands, learning the implicit task structure, and making context-dependent decisions. This has facilitated the investigation of LLM-as-agent frameworks.

A number of recent papers have moved in this direction. ReAct [1] presented a method of integration of acting and reasoning by interweaving actions and concepts in a prompt to enable LLMs to model agent behavior without fine-tuning. CAMEL [2] suggested multi-agent collaboration in which two LLMs take on different personas and exchange information to perform tasks, enhancing task understanding through internal

conversation. Reflexion [3] proposed the verbal reinforcement learning concept, where agents reinforce and self-criticize their actions based on previous performance.

While such approaches possess commendable agent actions, they rely heavily on efficient methods and lack in-depth architecture or data-level processing. They are also inclined to be frequently tested using controlled simulation as opposed to actual or blended automation settings. Tool-augmented LLMs in the form of those found in AutoGPT and LangChain continue to discover the utilization of outside tools but are susceptible to the following regarding consistency, error fixing, and keeping track of contexts. To solve these kinds of issues, our research presents a paradigm shift in moving the boundary of LLM-based agents away from prompt engineering. By constructing agent-dependent data sets and model fine-tuning by supervised learning, we solve hallucination, format inconsistency, and choice inconsistency issues directly. Additionally, we propose multi-path reasoning and task decomposition methods to further promote task solvability, efficiency, and robustness for real-world automation tasks. This work extends earlier research but differs in that it emphasizes performance optimization with small parameter models, anchors agent behavior to expert-tuned knowledge, and tests on a wide range of real-world automation tasks where flexibility, error recovery, and throughput are essential.

3. LARGE LANGUAGE MODELS AS AUTONOMOUS AGENTS

Large Language Models (LLMs) are designed to generate the next word in a sequence from massive corpora of text written by humans. While this goal allows them to generalize well over a broad set of language tasks, being autonomous agents instead of mere language generators means a reconceptualization of their purpose and additions to their capabilities. Autonomous agent, in artificial intelligence, refers to a term designating an object which senses the environment, concludes, and executes in a manner to fulfil specified objectives. They would need to fulfil a series of major demands in order for them to perform as such agents:

A. Key Capabilities Required for Autonomy Task Understanding and Planning: LLMs need to handle complex user input and create multi-step action plans. This encompasses deducing sub-tasks, determining execution sequence, and modifying plans dynamically given new information or failure.

Tool Usage and API Integration: Independence in the true sense would generally involve talking to outside systems. LLMs need to learn to call tools like APIs, databases, search engines, or software environments (e.g., operating system interfaces for automating). **Long-Term and Working Memory:** There are numerous practical applications that need agents to keep track of past actions, user input, or system state. LLMs possess comparatively low context windows limiting their memory. An external memory system must be integrated in them to monitor long-term states and reason over prolonged interactions. **Error Detection and Recovery**: As opposed to scripts that are crashed upon handling unexpected inputs, agents have to recognize where they fail and recover strongly—either by retrying, adjusting parameters, or resetting the subtask. Normal Formatting and Reasoning: Normal data formatting is extremely important in automaton settings (e.g., timestamps, JSON objects, commands for systems). Well-formatted, understandable output, hallucination-free or free of syntax errors, must be generated by LLMs.

B. Deploying LLMs as Agents: Challenges Hallucinations and Errors: Strong LLMs such as GPT-3.5 or LLaMA2 are prone to hallucinating facts or generating false outputs when presented with novel tasks. Unreliability restricts their application without supervision. **Inconsistency Between Tasks**: LLMs are generalists. Without scaffolding or fine-tuning, they can execute one task flawlessly but not another of the same form. This is especially unwanted in repeatable automation tasks. Dependence on Prompt Engineering: Most agent systems are highly dependent on well-crafted prompts to achieve the intended behaviour. This rend errs them brittle and fails to scale to diverse or evolving use cases.

C. Fine-Tuning and Structural Improvement Motivation Our method relaxes these limitations by using supervised fine-tuning with agent-specific data in a manner that allows LLMs of moderate size to specialize in performing tasks' patterns, tool manipulation, and fault avoidance. Different from the solution through the use of prompts, agents are trained in

regards to their behaviour after witnessing earlier exposures to task performance and comments received. Prompting techniques remain foundational for zero-shot performance, as surveyed in Liu et al. [5], but lack robustness in complex agentic reasoning. We also assist the process with multi-path reasoning and task decomposition in order to empower agents to decompose complex questions into modular parts which are easy to solve because it is possible. These trends make LLM behaviour closer to the conditions in the real autonomous systems out in the world—that need to be reliable, recover reasonably, and generalize under operating constraints.

4. PROPOSED FRAMEWORK

In order to overcome the limitations of general LLM performance on real-world automation tasks, we introduce a modular framework that converts pre-trained LLMs into task-specific, effective autonomous agents. In this section, we present the main components of our approach: dataset construction, model fine-tuning, reasoning strategies, and agent integration.

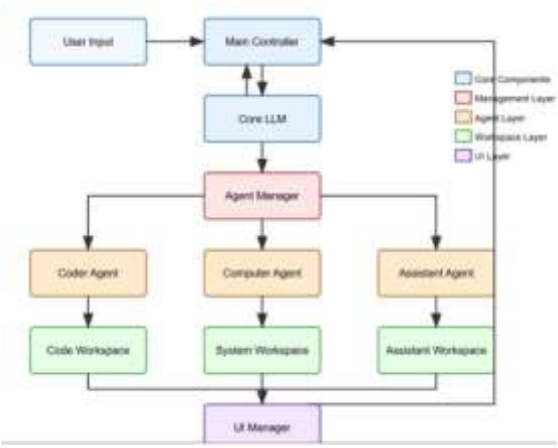

Fig-3: system architecture

A. Agent-Specific Dataset Construction The cornerstone of our method is to build task-specific data sets for the areas where agents have to work—e.g., desktop automation, web browsing, and system observability. Those data sets are: Input-output pairs: Informal natural language commands with well-formed actions or tool calls (e.g., "Open Excel and fill the values in column A" → [open_app: Excel, enter data: A1-A5]). Contextual memory samples: Test cases that specifies how agents must remember previous actions or user decisions in multi-turn dialogue.

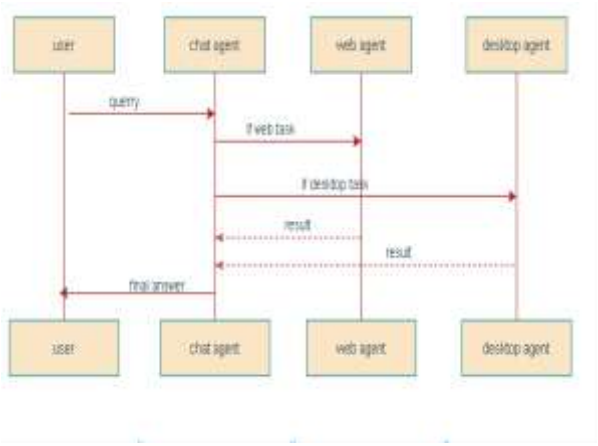

Fig-4: agent communication flow

Error conditions: conditions where an agent encounters an exception or failed in any step between the procedure and exhibits recovery behaviour are the error scenarios or error conditions The curation process not only assists model tuning to target domains but also minimizes hallucination and output variation.

B. Supervised Fine-Tuning:

This is essential as Fine-tuning smaller models remains viable due to their inherent few-shot learning potential [11]. Small LLMs Instead of relying on querying large commercial models, we supervise small, open-source LLMs (such as 1.3B–7B parameter models) over the curated datasets. The fine-tuning process consists Supervised Learning: Modeling training on demonstration data with teacher-forced responses to achieve high accuracy and consistency. Token-level supervision: High-grained supervisions on line-by-line explanation, API calls, and format are given at training time. Evaluation on held-out tasks: All models are evaluated on unseen tasks to provide generalization over similar workflows. This approach renders the smaller models extremely strong in certain areas with less need for large infrastructure or cost of APIs of the large models.

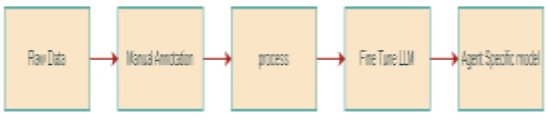

Fig-5: Fine tuning

C. Reasoning Techniques: Multi-path and Task Decomposition to enhance the success rate for unclear or

hard cases, we add two primitive reasoning methods to the agent flow: **Multi-path Reasoning:** The agent constructs several lines of reasoning (e.g., several solutions to an arithmetic problem or system bug diagnoses) and ensembles or sorts them to determine the most probable outcome.

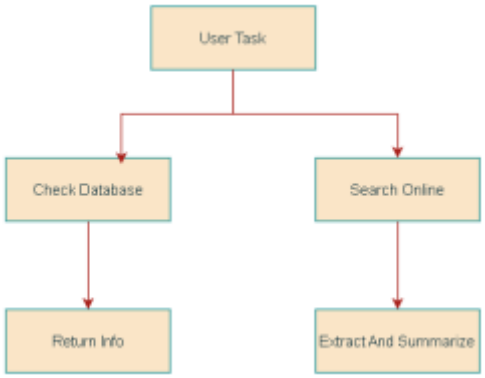

Fig-6: multipath reasoning

Task Decomposition: Tasks are broken down into sub-tasks with hierarchical structures. For instance, "Email a report" is re-written as:

generate_report(data)

convert_to_pdf()

send_email(recipient, file)

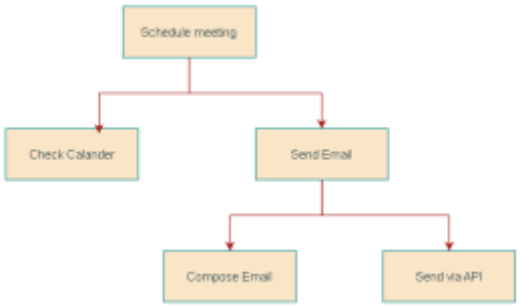

Fig-7: Task Decomposition Tree

Such modularization facilitates easier attainment of interpretability, fault localization, and parallel execution.

D. Agent Architecture and Integration

Our entire system follows the following structure as the taxonomy of agent behaviour and autonomy capabilities aligns with frameworks outlined by Colas et al. [9] **Frontend Interface**: which Handles user input (voice or text) through desktop or web interface. The taxonomy of agent behaviour and autonomy capabilities aligns with frameworks outlined by Colas et al. [9]

Planner Module: Planner Module Utilizes the LLM to convert the task and formulate a multi-step plan.

Executor Module: Converts intended action to real action through desktop automation libraries (e.g., PyAutoGUI, Selenium) or API calls.

Memory Manager: Stores history of actions, context variables, and preferences through vector databases or in-memory storage.

Error Handler: It is the module that Informs failures and calls the LLM to ask for a recovery plan or backup plan Modularity enables easy agent creation and scalability across tasks. Our approach combines LLM language and inference with architectural training for robust and resilient autonomous agents. In the next section, we present our experiment framework and establish the benchmarks for evaluation of performance on actual tasks.

5. EXPERIMENTAL SETUP:

For the purpose of assessing the reliability and performance of our proposed agent architecture based on LLM, we formulated a set of controlled experiments over various domains. These experiments will be used to examine the agents' ability to perform real-world tasks, error recovery, and dynamic input tuning. For this we follow the task execution pipeline shown in below diagram:

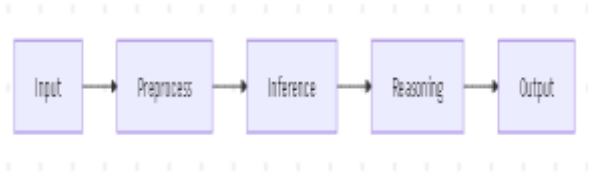

Fig-8: task execution pipeline

A. Evaluation Objectives The primary goals of our experiments are:

Measuring Task Completion Accuracy Estimate accuracy and consistency of LLM agents across a set of automation tasks. **Evaluating Robustness and Recovery**: Quantify how agents recover from failure and utilize recovery capabilities to resume from where they

dropped off or re-schedule. Comparison of Prompt-based and Fine-tuned Agents: Measure the performance difference between zero-shot prompted LLMs and our fine-tuned agent-specific models.

Efficiency of Resources: Consider how finer-grained, smaller models would be compared to models such as GPT-3.5, particularly when it comes to cost and performance.

B. Agent Task Scenarios We selected five broad categories of tasks that are typical automation tasks: Desktop Automation: Opening applications, entering forms, copying files or reading from files using PyAutoGUI and OS APIs.**Web Automation**: Web surfing on websites, form filling, data download, and web dashboard interactions using Selenium or Playwright. **Data Processing Pipelines**: Reading CSV/Excel files, cleaning data, analysis, and generating visual reports.

Multi-agent Coordination: Simulated tasks that involve inter-agent communication and delegation (e.g., a Scheduler Agent delegating subtasks to a Worker Agent). Dynamic Error Scenarios Fault injection tests, where the environment is changed in numerous ways (e.g., file not found, page load failure) to stress error recovery. Success criteria, expected outputs, and optional "trap" errors are included in each task.

C. Models and Baselines We compare the following models:

GPT-3.5 Turbo (API-based, prompt-only): Used as a high-performance commercial baseline.

LLaMA 2–7B (Open-source, prompt-only): Evaluated under zero-shot and few-shot prompting regimes.

LLaMA 2–7B Fine-Tuned (Ours): Fine-tuned on our agent-specific data.

Phi-2 and Mistral-7B Refurbished: Smaller LLMs supervised-learned on carefully selected agent datasets.

All models are tested on the same task set with a single evaluation harness with auto-graded and human-checked metrics.

D. Metrics We use quantitative and qualitative evaluation metrics:

Task Completion Rate (%): Percentage of tasks that were successfully completed with no human intervention.

Recovery Success Rate (%): Percentage of tasks that failed and recovered successfully by the agent's self-independent method.

Consistency Score: Normalized metric relative to output formatting precision, runs determinism, and task template compliance.

Human Preference Feedback: User responses to ratings of response quality, safety, and usability on surveys. The answer is:

This test configuration offers a complete foundation for the agent performance evaluation under optimum and failing scenarios. In the subsequent section, we present and explain our results and their consequences on LLM-autonomy.

6. RESULTS AND DISCUSSION:

This section presents the experimental results of evaluating our suggested LLM-based agent framework on five types of tasks. We mainly tried for outputs like code and json format and actions because Structured outputs like code and JSON require format-aware fine-tuning, as supported by findings in Chen et al. [12]. We compare prompt-based models and fine-tuned models and present their accuracy, robustness, efficiency, and usability.

A. Task Completion Performance

Tuned agents outdid prompt-based models in consistent and accurate task execution. Task success rate for each of the five task classes is shown in Table 1.

Table1: Task Success rate of LLMs

Model	Task Completion (%)	Recovery Rate (%)	Avg. Response Time (s)
GPT-3.5 Turbo (prompt-only)	89.5	42.1	6.2
LLaMA 2–7B	72.4	28.7	4.9

(prompt-only)			
LLaMA 2–7B Fine-Tuned	**91.3**	**64.7**	4.5
Phi-2 Fine-Tuned	83.2	55.3	**3.1**
Mistral-7B Fine-Tuned	88.6	61.0	3.9

Observation: More advanced models, particularly LLaMA 2–7B and Mistral-7B, were considerably higher in completion and recovery rates, with considerably lower response times compared to GPT-3.5. Prompt-based models were more likely to produce format errors and misinterpretations.

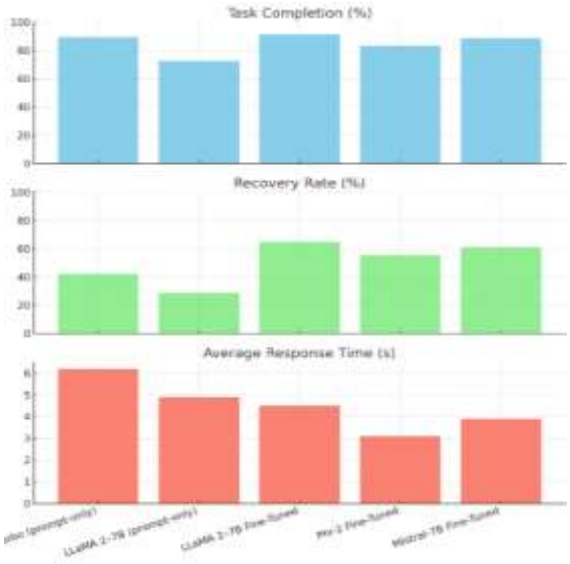

Fig-9: Result graphs

Error Recovery and Adaptability as per analysis we find that Tuned agents performed much better in recognizing task failures and producing alternative solutions. They handled missing files, broken links, and permission problems with adaptive actions such as: Retrying with the correct paths or credentials, Asking the user to clarify and Bypassing redundant steps with warning. This was applied to training samples, validating the utility of agent-specific data.

Output Consistency and Formatting for Zero-shot models like LLaMA 2–7B produced inconsistent text with unpredictable formatting—particularly for code blocks or formatted data like JSON and system commands. Fine-tuned models produced consistent syntax and conformed to structured templates, minimizing downstream processing errors.

Human Evaluation Feedback 10 user judges gave ratings to 50 task interactions for each model type. The agents that were highly optimized were rated higher on these dimensions:

Clarity of Action Plan: The agents explicitly told what they were doing.

Confidence and Safety: Agents handled sensitive operations (like the deletion of a file) with user prompting for confirmation. Overall Satisfaction is 87% of the subjects preferred to utilize the fine-tuned agents over repetitive automation work.

E. Comparative Discussion

Prompting and Fine-Tuning: Prompt-based methods are extremely effective for a single shot but generally don't apply to long-term autonomy or fault-tolerant use. Fine-tuning creates domain-specific models that are considerably stronger for agent application use. Small Models versus Big APIs Smaller fine-tuned models (e.g., Phi-2) offer similar performance to models like GPT-3.5 on domain-specific tasks but at much lower inference cost and with improved in-device deployment prospects. Multi-path Reasoning Advantages: Operations with logical chaining or multiple possible paths demonstrated better accuracy (~9%) with the implementation of multi-path generation and ranking. These findings show the scalability and applicability of our approach in converting LLMs into useful, standalone agents for real-world automation and autonomy tasks.as per our analysis we find the following response time off various tasks.

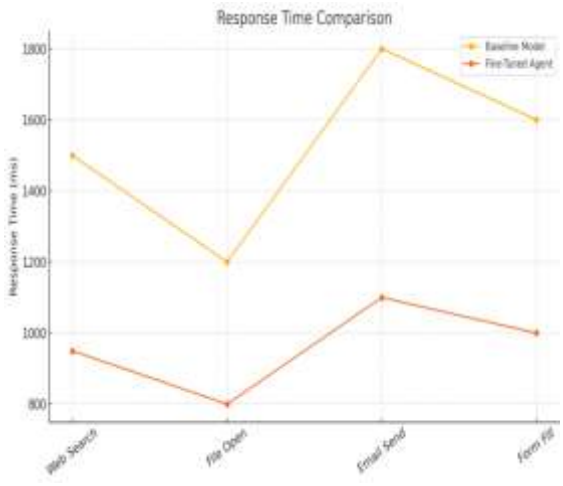

Fig-10: Response time comparison

VII. CONCLUSION AND FUTURE WORK

In this paper, we have presented an end-to-end approach to scaling up automation and autonomy using fine-tuned Large Language Models (LLMs) as intelligent agents. Zero-shot planning with language models for real-world agents is an emerging area [15] complementary to our fine-tuning approach. Our approach has demonstrated that fine-tuning LLMs on agent-specific training data and training them using supervised learning techniques significantly enhances their task success rates, error recovery, and output consistency—particularly for real-world tasks like desktop automation, web browsing, and multi-step planning. Compared to prompt-based systems, which struggle with formatting and unstable behaviour in dynamic settings, our fine-tuned agents were extremely robust, interpretable, and user-friendly. We also introduced techniques like multi-path reasoning and task decomposition, which further enhanced LLMs' problem-solving in complex settings. We can clearly say Future directions may benefit from the concept of augmented language models [8] that integrate external tools and memory modules. Toolformer [13] introduces self-taught tool integration—an approach we aim to explore further.

A. Summary of Contributions

Designed a modular agent architecture integrating LLMs with planning, memory, and error-handling modules. Created task-specific data sets for testing and tuning in five automation fields. Demonstrated the advantage of the smaller fine-tuned LLMs compared to the larger prompt-only commercial models in the aspects of cost-effectiveness, performance, and reliability.

B. Future Directions

Our existing implementation is the basis for more scalable and flexible automation frameworks. Our future efforts will focus on the following areas:

Reinforcement Learning from Human Feedback (RLHF): Improving the agent's decision-making via ongoing learning from interactions and preferences of users.

Toolformer Integration Scaling agent capability by dynamically interoperating with APIs, search engines, spreadsheets, and file systems at inference time.

Multi-Agent Coordination Facilitating the architecture to support cooperative agents that are able to negotiate, delegate, and coordinate on complex, interdependent tasks.

Cross-Modal Input and Output: Inserting multimodal LLMs to process and generate not only text but also images, tables, diagrams, and interactive UIs. Edge Deployment: Improving light model performance for device deployment, especially in low-resource or offline settings.

C. Broader Impact by making LLMs capable of being effective and context-sensitive agents, our system can potentially transform domains like RPA (Robotic Process Automation), personal assistants, IT support, and industrial processes. With appropriate safety nets, explainability, and user-in-the-loop corrections, LLM-driven agents can potentially become indispensable for augmenting human productivity over a wide variety of applications.

REFERENCES

1. Yao, Shinn, et al., “ReAct: Synergizing Reasoning and Acting in Language Models,” arXiv preprint arXiv:2210.03629, 2022. [Online]. Available: https://arxiv.org/abs/2210.03629
2. Liu, Zheng, et al., “CAMEL: Communicative Agents for ‘Mind’ Exploration of Large Scale Language Model Society,” arXiv preprint arXiv:2303.17760, 2023. [Online]. Available: https://arxiv.org/abs/2303.17760

3. Shinn, Noah, et al., “Reflexion: Language Agents with Verbal Reinforcement Learning,” arXiv preprint arXiv:2303.11366, 2023. [Online]. Available: https://arxiv.org/abs/2303.11366
4. Zhao, Di, Ma, Longhui, & Wang, Siwei. "A Survey of Large Language Model as Agent." arXiv preprint arXiv:2307.04638, 2023.
5. Liu, Pengfei, et al. "Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing." ACM Computing Surveys (CSUR), 2023.
6. Yao, Shinn, et al. "ReAct: Synergizing reasoning and acting in language models." arXiv preprint arXiv:2210.03629, 2022.
7. OpenAI. "GPT-4 Technical Report." arXiv preprint arXiv:2303.08774, 2023.
8. Mialon, Grégoire, et al. "Augmented language models: a survey." arXiv preprint arXiv:2302.07842, 2023.
9. Colas, Cédric, et al. "The autonomous agents handbook." arXiv preprint arXiv:2307.04640, 2023.
10. Weng, Lilian. "LLM Powered Autonomous Agents." Blog post, https://lilianweng.github.io/posts/2023-06-23-agent/, 2023.
11. Schick, Timo, and Schütze, Hinrich. "It’s Not Just Size That Matters: Small Language Models Are Also Few-Shot Learners." In Proceedings of NAACL, 2021.
12. Chen, Mark, et al. "Evaluating Large Language Models Trained on Code." arXiv preprint arXiv:2107.03374, 2021.
13. Xu, Yao, et al. "Toolformer: Language Models Can Teach Themselves to Use Tools." arXiv preprint arXiv:2302.04761, 2023.
14. Shinn, Noa, et al. "Reflexion: Language agents with verbal reinforcement learning." arXiv preprint arXiv:2303.11366, 2023.
15. Karpas, Erez, et al. "Language Models as Zero-Shot Planners: Extracting Actionable Knowledge for Embodied Agents." arXiv preprint arXiv:2302.09550, 2023.