iJRASET
International Journal For Research in Applied Science and Engineering Technology



INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 14 **Issue:** XI **Month of publication:** November 2026

DOI: https://doi.org/10.22214/ijraset.2026.85609

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 14 Issue XI Nov 2026- Available at www.ijraset.com

Agentic AI: A Review and Proposed Framework

Ronit Raj Sharma[1], Abhijeet Kumar [2], Dr. Vishal Shrivastava [3], Dr. Akhil Pandey[3]
[1, 2, 3]*Computer Science & Engineering, Arya College of Engineering & I.T. Jaipur, India*

Abstract: ***Chatbots built on large language models (LLMs) are good at producing text, but most of them stop at the reply. They cannot look up an order, run a test or update a record unless someone connects them to tools and tells them how to use those tools safely. Agentic AI is the name given to systems that close this gap: an LLM is placed inside a loop where it makes a plan, calls a tool, reads what came back and decides what to do next. This paper reviews the main building blocks of such systems, namely planning and reasoning, tool use, memory and state, and human oversight, and places them in the recent literature on ReAct-style agents, tool learning, reflection, multi-agent conversation and agent benchmarks. Building on this review, we propose a modular agent pipeline made of a tool registry with permissions, a planner, a permission check, an optional human approval gate, tool execution with observation, result checking and an audit log. We also describe how such a pipeline should be evaluated, using task success, tool errors and injection resistance, and time and cost. The paper is a review with a proposed design and does not report new experiments. It closes with open problems, including reliable planning over long tasks, defence against prompt injection, and fair evaluation of agents that behave differently on every run.***

Keywords: Agentic Ai, Large Language Models, Ai Agents, Tool Use, Planning, Multi-Agent Systems, Human-In-The-Loop, Prompt Injection.

I. INTRODUCTION

Large Language Models built on the transformer architecture [1] can summarise, translate, explain and answer questions in fluent language. Used as a chatbot, though, a model does one thing: it reads a message and writes a reply. If the user asks it to help with a late order, the chatbot can describe what a good answer looks like, but it cannot open the order system, read the return policy or send the reply. A person still has to do the actual work, and the confident tone of the model can hide the fact that it never checked anything. Agentic AI tries to change that. In an agentic system the language model is not asked for a single answer. It is given a goal, a set of approved tools and some memory, and it works in a loop: plan a step, call a tool, look at the result, and adjust. The idea gained attention after work such as ReAct [3], which showed that letting a model alternate between reasoning and acting makes it more useful on tasks that need outside information, and Toolformer [4], which showed that a model can learn when to call simple tools. Since then, agents have been built for web browsing [9], software repair [10] and many other tasks; a broader survey is given in [14].

A typical agentic system has four cooperating parts: a planner, usually an LLM, that decides what to do next; a set of tools that let it act, such as search, files, calculators and business APIs; a memory that keeps track of the task, the past steps and useful results; and a control layer that checks permissions, asks for human approval when needed and records what happened (Fig. 1). The rest of this section introduces these parts, Section 2 reviews representative work, and Section 3 presents a proposed pipeline that combines them.

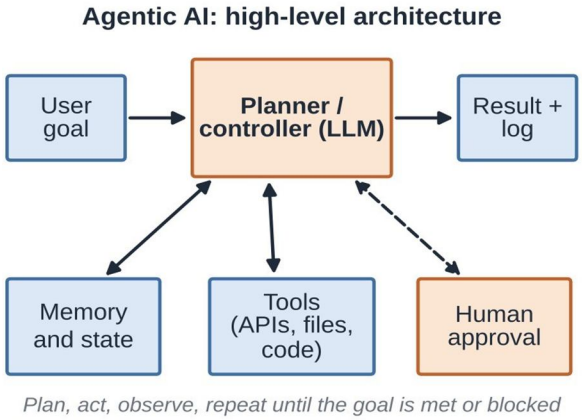


Fig. 1: High-level architecture of an agentic AI system

Why study this carefully? Because an agent that can act can also make mistakes that matter. A chatbot that gets a date wrong wastes a minute. An agent that refunds the wrong order, deletes a file or follows a hidden instruction in a web page can cause real damage. Researchers have shown that applications connected to outside content can be steered by instructions planted in that content [12], and that agents can be tested for this problem in a systematic way [11]. For this reason we treat safety, permissions and human control as part of the design from the start, not as extras added at the end. Recent human-in-the-loop systems take the same view and keep a person involved in planning and in risky actions [13].

A. Planning and Reasoning

Planning is how an agent turns a goal into steps. The simplest approach is to let the model think step by step before answering, which chain-of-thought prompting showed can improve reasoning on harder problems [2]. In an agent, that reasoning is mixed with actions. In a ReAct-style loop the model writes a short thought, picks an action, reads the observation and then writes the next thought [3]. A different approach is plan-then-execute, where the model writes the whole plan first and then carries it out, revising only when something fails. The first approach adapts quickly but can wander. The second is easier to inspect and approve but can break when the situation changes in the middle of a task. Table 1 compares the common patterns. In practice many systems mix them: a short plan up front, then a loop that is allowed to change the plan.

Table 1: Comparison of control patterns used in agentic systems

Pattern	How it works	Strength	Limitation
Single reply (chatbot)	One prompt, one answer, no tools	Simple and cheap	Cannot act or check results
Reason and act loop	Thought, action, observation, repeat	Adapts to tool results	Can wander or loop
Plan then execute	Full plan first, then the steps	Easy to review and approve	Brittle when the plan goes stale
Multi-agent team	A coordinator hands tasks to specialists	Splits complex work	More cost and coordination

B. Tool Use

Tools are what separate an agent from a chatbot. A tool can be a web search, a file reader, a calculator, a code runner, or an API for an order system or a CRM. Each tool needs a name, a short description and clear input rules, because the model chooses tools by reading those descriptions. Toolformer showed that a model can learn when a call to a simple API will help it [4], and most agent frameworks today expose tools through function calling or a similar interface. Good tool design matters more than people expect. A tool that returns a vague error message leaves the agent guessing, and a tool with too much power, for example one that can write to any table, turns a small planning mistake into a large one. Our rule of thumb is to give each agent the smallest set of tools it needs, with the narrowest permissions that still let it finish the job.

C. Memory and State

An agent needs to remember what it is doing. Short-term memory is the running context: the goal, the plan and the last few observations. Longer tasks need more than the context window can hold, so systems write notes, summaries and results to an external store. Generative agents kept their experiences in a memory stream and later reflected on them to plan their days [6], and Reflexion let an agent write down in plain language what went wrong and use those notes in its next attempt [5]. When an agent needs facts from documents, it can call a retrieval tool, which is where Retrieval-Augmented Generation [15] fits in, as one tool among others. Table 2 lists common memory and state options.

Table 2: Overview of memory and state options used in agents

Option	Holds	Typical use	Notable limit
Context window	Goal, plan, recent steps	Short tasks	Fills up quickly
Scratchpad or notes	Intermediate results	Multi-step reasoning	Needs pruning
Vector store (RAG)	Documents and past notes	Looking up facts	Retrieval can miss the answer
State store	Task status, IDs, flags	Resuming a task	Needs a clear schema
Audit log	Every tool call and outcome	Review and debugging	Grows over time

D. Control, Feedback and Human Oversight:

The control layer decides what the agent is allowed to do and when it must stop. It checks every tool call against the permissions of the user and the agent, limits the number of steps and retries, and sends risky actions such as payments, refunds or deletion to a person. Limits matter because agents can get stuck. An agent that retries a failing call forever burns time and money without getting closer to the goal. Fig. 2 shows the usual shape of this trade-off: completion improves quickly over the first few steps and then flattens, while cost keeps rising. The figure is synthetic and only meant to illustrate the idea; the right limit has to be found by testing on real tasks. Finally, the control layer writes an audit log, so that a person can later see what the agent did and why.

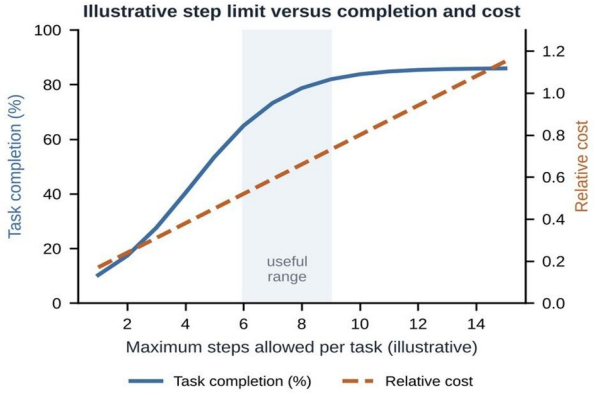


Fig. 2: Illustrative relation between step limit, task completion and cost (synthetic data, for conceptual illustration only)

II. RELATED WORKS

Work on agents has grown quickly over the last few years. This section groups representative approaches by the problem they address. Table 3 gives a short overview before each is described.

Table 3: Summary of representative approaches reviewed in this paper

Approach	Focus	Key idea
ReAct (Yao et al.)	Reasoning with actions	Interleave thoughts and tool actions
Toolformer (Schick et al.)	Learning tool calls	Model learns when to call simple APIs
Reflexion (Shinn et al.)	Learning from feedback	Verbal self-reflection kept as memory
Generative Agents (Park et al.)	Memory and planning	Memory stream, reflection and planning
AutoGen (Wu et al.)	Multi-agent work	Conversing agents with humans and tools
Magentic-UI (Mozannar et al.)	Human oversight	Co-planning and approval of actions

A. *Reasoning and Acting*

Yao et al. proposed ReAct, in which a language model produces reasoning traces and actions in an interleaved way [3]. The reasoning helps the model keep track of and update its plan, while the actions let it pull in information from outside sources such as a search interface. The authors tested the approach on question answering, fact checking and interactive decision-making tasks, and reported fewer made-up facts than with reasoning-only prompting. ReAct is now the template behind many practical agent loops, even when the exact prompt differs.

B. *Learning to Use Tools*

Schick et al. introduced Toolformer, a model trained in a self-supervised way to decide which APIs to call, when to call them and how to use the result in what it writes next [4]. The tools included a calculator, a search engine, a question answering system, a translator and a calendar. The point of the work is that tool use can be learned from only a handful of demonstrations per API instead of being hard-coded.

C. *Reflection and Memory*

Shinn et al. proposed Reflexion, in which an agent that fails a task writes a short verbal reflection on what went wrong and keeps it in an episodic memory for the next attempt, without any change to the model weights [5]. Park et al. built generative agents that store experiences in natural language, condense them into higher-level reflections and retrieve them when planning [6]. Both lines of work suggest that a simple, readable memory can go a long way, and because the memory is readable, a person can inspect it, which helps with debugging.

D. *Multi-Agent Systems*

Wu et al. presented AutoGen, a framework in which several customisable agents talk to each other, and to humans and tools, to finish a task [7]. Splitting the work this way can make each agent's job smaller and its tool set narrower, which fits the least-privilege idea above. The price is extra coordination and more calls to the model. Because of this we follow the common advice to start with one agent and add more only when the task clearly needs it.

E. *Human-in-the-Loop Agents*

Mozannar et al. described Magentic-UI, a research prototype built to study how people and agents work together [13]. It includes interaction mechanisms such as co-planning, where the user can edit the plan of the agent before it starts, and action guards, where risky actions wait for approval. Systems like this move the question from "can the agent do it alone?" to "where should a person stay in control?", which we think is the more useful question for real deployments.

F. *Benchmarks and Evaluation*

Several benchmarks test agents on realistic work. AgentBench evaluates LLMs as agents across a range of interactive environments [8]. WebArena provides realistic websites in which agents must complete everyday tasks, and it showed a large gap between agent and human success when it was released [9]. SWE-bench asks models to resolve real issues from GitHub repositories, which requires reading code, editing it and checking the outcome [10]. Taken together, these benchmarks point to one lesson: scoring only the final answer hides how the agent got there, so tool calls, steps and cost need to be tracked as well.

G. *Security of Agents*

Greshake et al. showed that applications which connect LLMs to external content can be attacked through indirect prompt injection, where instructions hidden in a web page or document are read by the model and treated as commands [12]. Debenedetti et al. built AgentDojo, an environment of realistic tasks with security test cases, for measuring both how useful agents are and how well they resist such attacks [11]. The lesson for design is plain: anything an agent reads from outside the system should be treated as untrusted input, and the permissions of the agent should never depend on trusting it.

III. PROPOSED METHODOLOGY

This section presents a proposed agent pipeline for tasks in which an LLM must use approved tools to finish a goal. We use an online store support desk as the running example. The design follows the parts introduced in Section 1 and is organised so that each stage can be built and tested separately.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

A. Goal and Scope Definition

The pipeline begins with a clear goal and a clear boundary. For the store example, a goal might be "prepare a reply for this customer about a late order". The task context lists what the agent may know, such as the order ID and the customer message, and what the finished result should look like, here a draft reply and a proposed action. Vague goals lead to wandering agents, so the goal is written as a short task with a definition of done.

B. Tool Registry and Permissions

Every tool is listed in a registry with a name, a plain-language description, its input rules and the permission it needs. In the store example the registry holds an order lookup, a return-policy search and a reply drafting tool. Read-only tools are allowed by default. Tools that change something, such as issuing a refund, are marked as needing human approval. The registry is the only place where the agent can learn which tools exist, so adding or removing a tool does not require changing the planner.

C. Planning

The planner, an LLM, receives the goal, the context and the tool descriptions and writes a short plan of small steps, for example: read the order, check the policy, draft the reply. The plan is stored in memory and can be revised after each observation. For higher-risk tasks the plan can be shown to a person before any tool is called, in the style of co-planning [13].

D. Memory and State

A small state record holds the goal, the plan, the step number, the results so far and any errors. Long results are summarised before they are stored, so that the context does not fill up. Everything the agent does is also appended to an audit log, which is kept separate from the memory that the model reads.

E. Acting and Observing

For each step the agent proposes a tool call with inputs. A permission check compares the call with the registry and with the rights of the current user. Calls that pass are executed with a timeout, and the result comes back as an observation. Content that comes from outside, such as a web page or a customer message, is marked as untrusted data and is never allowed to change the permissions or the plan on its own [12].

F. Verification and Reflection

After each observation the agent checks whether the step worked and whether the goal is now met. If a tool failed, it may retry a limited number of times or choose another route. If it is stuck, it writes a short reflection on what went wrong, as in Reflexion [5], and then either changes the plan or stops and asks a person for help. A step limit prevents endless loops.

G. Human Approval Gate

Actions marked as risky, such as payments, refunds, messages sent outside the company or deletion of data, are paused until a person approves them. The approver sees the proposed action, the evidence the agent used and the relevant policy text, so the decision takes seconds rather than minutes. The person can approve, edit or reject, and the choice is logged.

H. Proposed Algorithm

Input: Goal, tool registry, user permissions Output: Result with a supporting action log Process:

1) Define the goal, task context and definition of done.
2) Load the tool registry and the permissions of the current user.
3) Ask the planner for a short plan and store it in memory.
4) Select the next step and propose a tool call with inputs.
5) Check the call against the registry and permissions; reject it if it is not allowed.
6) If the action is marked risky, pause and request human approval.
7) Execute the tool with a timeout and capture the observation.
8) Check the result; retry, revise the plan or reflect if the step failed.
9) Repeat steps 4 to 8 until the goal is met, the step limit is reached or the agent is blocked.
10) Return the result with a short summary and write the full action log.

I. Proposed Flowchart

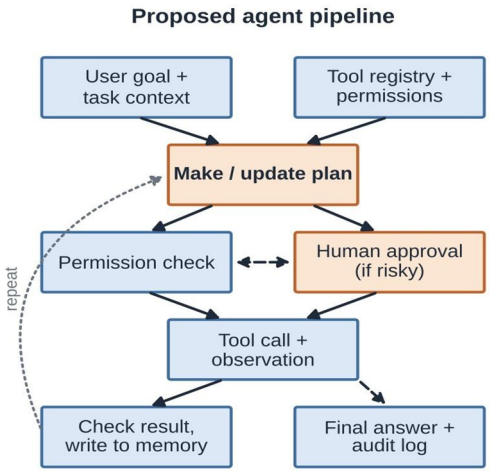


Fig. 3: Proposed agent pipeline flowchart, showing planning, the permission check, the human approval gate and the repeat loop

The pipeline is intentionally modular. The planner model, the tool registry, the permission checker, the memory store and the approval interface can each be replaced without redesigning the rest. A first version can be a single agent with three tools; a coordinator and specialist agents can be added later if the load or the task needs them [7]. Table 4 lists the parameters we would tune first. The ranges are suggested starting points, not tested values.

Table 4: Suggested starting configuration for the proposed pipeline

Parameter	Starting range	Effect
Maximum steps	5 to 15	More steps raise completion but also cost and loop risk
Retries per tool	1 to 3	Recovers from temporary errors; too many waste time
Tool timeout	5 to 30 seconds	Stops a slow tool from blocking the task
Approval rule	Risky actions only	Fewer interruptions while keeping control
Context budget	Summarise long results	Keeps the prompt short and focused

IV. RESULTS AND DISCUSSION

A. Evaluation Methodology

Because this paper presents a review and a proposed design, this section describes how the pipeline should be evaluated once it is built, and does not report experimental numbers. An agent has to be judged on three separate questions: did it finish the task, did it act safely, and was it worth the time and cost (Fig. 4). A weakness in any one of them can ruin the result even if the other two look good.

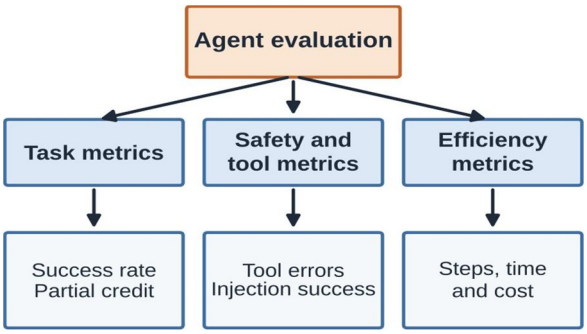


Fig. 4: Proposed evaluation framework, separating task, safety and efficiency metrics

B. Task Metrics

The main task metric is success rate: the share of test tasks that the agent finishes correctly, judged against a clear definition of done. Partial credit can be given when some of the required steps are completed, as long as the scoring rule is fixed in advance. The number of steps taken is recorded as well, since two agents with the same success rate may differ a lot in how directly they reach it.

C. Safety and Tool Metrics

Safety is measured with the tool-call error rate (calls with wrong inputs, the wrong tool or missing permissions), the number of policy violations, and the injection success rate, which is the share of planted malicious instructions that the agent follows, as in AgentDojo [11]. A separate check looks at whether risky actions were always sent for approval and how often the approver had to correct the agent.

D. Efficiency Metrics

Efficiency covers wall-clock time, the number of model calls, the tokens used and the money spent per task. These are compared with the manual time a person would need, because an agent that is slower or more expensive than the person it helps is hard to justify.

E. Comparison Baselines

A fair evaluation compares the pipeline with at least two baselines on the same tasks: a plain chatbot with no tools, and a single agent without the permission check and the approval gate. The first shows what tools add. The second shows what the safety layer costs in speed and what it saves in errors. Because agents can behave differently from run to run, each task should be repeated several times and the results reported with their spread, not as a single number.

F. Error Analysis

Beyond averages, failures should be sorted into types: planning failures (wrong or missing steps), tool failures (the wrong tool, bad inputs or a tool that returned an error), looping failures (repeated attempts without progress) and safety failures (an action taken without permission or approval). Knowing which type dominates tells the team whether to fix the planning prompt, the tool descriptions, the step limits or the approval rules.

G. Discussion

Several trade-offs follow from the design. More autonomy saves time but increases the damage a single mistake can do. More approval gates make the system safer but slower and more tiring for staff. More agents split the work but add cost and make failures harder to trace. We do not think there is one right setting; it depends on how costly a mistake is in the target domain. For a store support desk, drafting replies can be close to automatic, while refunds should stay under staff control. Benchmarks such as WebArena and SWE-bench were built because early agents failed on many realistic tasks [9], [10]. Results keep improving, but we still see human review as a necessary part of the system and not a temporary patch.

V. CONCLUSION AND FUTURE WORK

Agentic AI moves language models from writing about work to doing it. This paper reviewed the main parts of an agent, namely planning, tool use, memory and human oversight, summarised representative work from ReAct to Magentic-UI, and proposed a modular pipeline with a tool registry, permission checks, an approval gate and an audit log. We also set out how to evaluate such a pipeline on task success, safety and cost. The approach has clear limits. Agents depend on the quality of their tools and instructions, they can loop or plan badly, they add cost and delay, and anything they read from outside can be used against them. Future work should build and test the proposed pipeline on a real support-desk task set, study how to plan reliably over long tasks, improve defences against prompt injection, and develop evaluation methods that cope with agents that give different results on each run. Another useful direction is to find out how much human review is enough, so that oversight stays practical for the people who do it.

REFERENCES

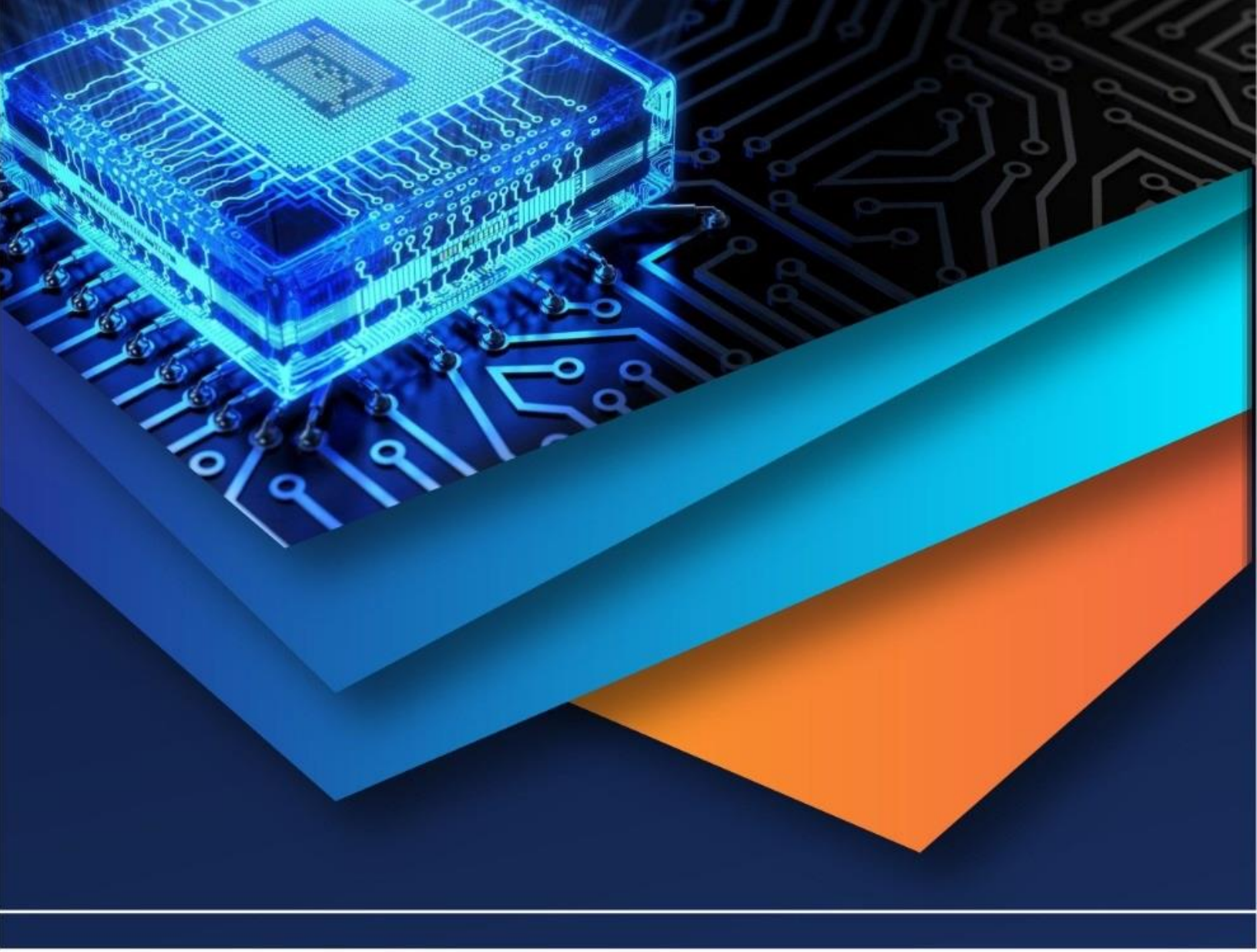
[1] Vaswani, A., et al. "Attention Is All You Need." Advances in Neural Information Processing Systems (NeurIPS), 2017.
[2] Wei, J., et al. "Chain-of-Thought Prompting Elicits Reasoning in Large Language Models." Advances in Neural Information Processing Systems (NeurIPS), 2022.

[3] Yao, S., Zhao, J., Yu, D., Du, N., et al. "ReAct: Synergizing Reasoning and Acting in Language Models." International Conference on Learning Representations (ICLR), 2023.
[4] Schick, T., Dwivedi-Yu, J., Dessì, R., Raileanu, R., et al. "Toolformer: Language Models Can Teach Themselves to Use Tools." Advances in Neural Information Processing Systems (NeurIPS), 2023.
[5] Shinn, N., et al. "Reflexion: Language Agents with Verbal Reinforcement Learning." Advances in Neural Information Processing Systems (NeurIPS), 2023.
[6] Park, J. S., et al. "Generative Agents: Interactive Simulacra of Human Behavior." ACM Symposium on User Interface Software and Technology (UIST), 2023.
[7] Wu, Q., Bansal, G., Zhang, J., Wu, Y., et al. "AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversations." Conference on Language Modeling (COLM), 2024.
[8] Liu, X., Yu, H., Zhang, H., Xu, Y., et al. "AgentBench: Evaluating LLMs as Agents." International Conference on Learning Representations (ICLR), 2024.
[9] Zhou, S., et al. "WebArena: A Realistic Web Environment for Building Autonomous Agents." International Conference on Learning Representations (ICLR), 2024.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

1. Jimenez, C. E., et al. "SWE-bench: Can Language Models Resolve Real-World GitHub Issues?" International Conference on Learning Representations (ICLR), 2024.
2. Debenedetti, E., Zhang, J., Balunović, M., et al. "AgentDojo: A Dynamic Environment to Evaluate Prompt Injection Attacks and Defenses for LLM Agents." Advances in Neural Information Processing Systems (NeurIPS), 2024.
3. Greshake, K., et al. "Not What You've Signed Up For: Compromising Real-World LLM-Integrated Applications with Indirect Prompt Injection." ACM Workshop on Artificial Intelligence and Security (AISec), 2023.
4. Mozannar, H., Bansal, G., Tan, C., Fourney, A., et al. "Magentic-UI: Towards Human-in-the-loop Agentic Systems." Microsoft Research Technical Report MSR-TR-2025-40, 2025.
5. Wang, L., et al. "A Survey on Large Language Model Based Autonomous Agents." Frontiers of Computer Science, 2024.
6. Lewis, P., et al. "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks." Advances in Neural Information Processing Systems (NeurIPS), 2020.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Call : 08813907089 (24*7 Support on Whatsapp)