International Scientific Journal of Engineering and Management (ISJEM) ISSN: 2583-6129
Volume: 05 Issue: 04 | April – 2026 DOI:10.55041/ISJEM06032
An International Scholarly || Multidisciplinary || Open Access || Indexing in all major Database & Metadata

REXIE AI - Commsync MCP using Automation Tools

Aleti Lakshmi Harika, Balaganu Shyam, Kaki Ramesh, Kadimi Bhuvaneswari and Mrs B.Sujata

Department of Information Engineering and Computational Technology, MVGR College of Engineering (A), Vizianagaram, Andhra Pradesh, India

Abstract-- Currently we a living in a AI world where everybody wants ai bots to do their external tasks , but present systems can't directly interact with external tools and perform actions . Rexie AI is a type of agent bot which uses MCP (Model Context Protocol) for connection to LLMs. This MCP works with tools like Gmail, Slack, Calendar, Docs, and SMS in a secure and organized way. LLMs are used for the most part to produce human like text and for understanding it, but they do not perform tasks by themselves. Also they have issues with external systems' interaction which is where MCP comes in to play in terms of intent interpretation and task performance. MCP is a server which integrates many AI agents to work together and solve problems. It supports real world applications via standardized context sharing also it improves efficiency, scale, and total system performance.

Keywords — Model Context Protocol, Context-Aware AI, Multi-Agent Systems, Large Language Models, AI Workflow Automation, Tool-Orchestrated AI

1. INTRODUCTION

As the artificial Intelligence evolves from traditional chatbot systems where it has achieved a significant attention due to its ability to understand natural language and generate human like text to real world task execution [9],[11]. It wants to perform tasks, not just generate content, so it comes with a server called MCP, which integrates with external tools and performs tasks [1][7]. This Agent Bot is a computerized program which acts like a virtual assistant and has become extremely popular [8],[12].

As in the digital world, communication plays a crucial role in almost all our lives. Rexie Bot is integrated with SMS, Gmail, and Slack external tools, which is very helpful in organizations to send messages to multiple persons with no time. This not only reduces the task, saves money, and also the labour. The Rexie Bot is also integrated with the calendar and document tools to schedule meetings and create documents. These applications boost the power of Rexie AI. The transition from single agents xto multi-agent systems. Rexie is a tool-aware AI assistant that can decide which tool to use, when to use, and how to use multiple tools within one conversation [11][16].

II. LITERATURE SURVEY

The Model Context Protocol (MCP) has lately surfaced as a significant frame for enabling structured commerce between Large Language Models (LLMs) and external tools. Singh et al. presented one of the foremost comprehensive checks on MCP, defining it as a formalised protocol that facilitates harmonious communication between LLMs and external systems while conserving contextual durability [1]. Their work highlights how MCP addresses the limitations of fractured API- grounded integrations in conventional AI systems. Pajo delved into the sphere of workflow robotization and demonstrated that MCP enables dynamic, environment- apprehensive task prosecution, unlike traditional rule- grounded systems similar as Zapier and IFTTT [2]. This rigidity is particularly applicable for systems like Rexie, where task prosecution depends on stoner intent rather than predefined workflows. Sanikommu explored the operation of MCP in enhancing LLM performance for observability and analytics, introducing mechanisms similar as environment encoding and applicability scoring [3]. These ways significantly reduce token inefficiency and ameliorate response delicacy, which are critical for real- time conversational systems. Parwani et al. proposed MCP as a scalable frame for environment- apprehensive multi-agent collaboration,

introducing concentrated environment scales and memory buffers to ameliorate agent collaboration [4]. also, Krishnan addressed the issue of disconnected models in multi-agent systems by enabling participated contextual memory and patient logic across agents [5]. These benefactions support the transition from single-agent systems to multi-agent infrastructures, which is a crucial aspect of Rexie. Jonnadula et al. anatomized MCP from an enterprise integration perspective, emphasizing its capability to support interoperability, dynamic tool discovery, and secure unity across distributed systems [6]. still, Ray linked several challenges associated with MCP, including quiescence, security pitfalls, and scalability limitations, which must be considered in real- time executions [7]. In the broader environment of Artificial Intelligence, Russell and Norvig handed foundational generalities of intelligent agents and decision- making systems [8]. Brown et al. demonstrated the effectiveness of LLMs in natural language understanding and generation, forming the base for ultramodern conversational AI systems [9]. Wang et al. explored LLM- powered agents and stressed the significance of integrating LLMs with external tools to enable real- world task prosecution [11]. This aligns directly with the ideal of Rexie to extend AI capabilities beyond textbook generation to action- grounded systems. Wooldridge bandied the principles of multi-agent systems, emphasizing collaboration and communication among agents [12], which is reflected in Rexie's tool unity medium. Chen et al. further demonstrated the rigidity of LLMs in structured tasks similar as law generation [13]. Li et al. examined environment- apprehensive AI systems and stressed the challenges associated with maintaining applicable environment efficiently [14]. Zhou et al. demonstrated the effectiveness of tool-stoked LLMs for real- world task robotization [16]. while Liu et al. emphasized the significance of prompt engineering and environment operation in perfecting LLM performance [17]. Recent advancements by OpenAI and Google Research highlight the growing significance of function calling and agent- grounded infrastructures in ultramodern AI systems [18], [19]. These developments support the necessity of fabrics like MCP for enabling structured and scalable tool commerce. Eventually, Fowler's work on enterprise operation armature supports the modular and service-acquainted design espoused in Rexie, enabling scalability and maintainability [20].

Research Objectives

1. To understand the capability of Generative AI in performing real-world tasks beyond content generation.
2. To analyze how Large Language Models (LLMs) can interpret user intent and support task execution.
3. To study the role of the Model Context Protocol (MCP) in integrating AI systems with external tools.
4. To design and develop an AI-powered assistant that automates tasks such as email communication, scheduling, and document generation.
5. To evaluate the effectiveness of tool integration (Gmail, Slack, Calendar, SMS) in improving productivity and reducing manual effort.
6. To analyze system performance in terms of response time, task execution efficiency, and real-time interaction.
7. To provide a scalable and modular solution for intelligent workflow automation using AI.

Problem Statement

Large Language Models (LLMs) are very good at making natural language, but they have a lot of problems when they are used in the real world. They often have trouble with tasks that require more than one step, can't keep track of context across workflows, and can't reliably use external tools like email, SMS, or file generators. These limits lead to wrong outputs, unnecessary communication, and broken user experiences. The Model Context Protocol (MCP) was created to make systems more reliable and secure, but most of the research so far has been on its theoretical design and security implications, not on how to use it in real life. This means that we can't use MCP to make AI-powered communication systems that are efficient, aware of their surroundings, and can meet real-world needs.

Scope of the study

Our research is into what Generative AI does in terms of task performance and also how it develops agent bots which in turn perform those tasks. Many people are out to do tasks faster, in particular repetitive and time extensive ones like sending out emails to each set of candidates and sending them out separately. We rather put our energy into more productive activities instead of being tied up in simple repeat tasks. We put our effort into better things instead of in to simple repeat tasks. We present our study which developed MCP which we use as a middle man system between the

LLM and external tools It is a element which has the agent bots choose the right tool and carry out tasks very efficiently. This we see play out in the benefit to many B2B companies and educational institutions which report easier and more efficient task execution.

III. METHODOLOGY

The methodology adopted in this paper focuses on system design and implementation rather than statistical modeling. A modular and service-oriented architecture is used to integrate Large Language Models with external tools through MCP. The system includes frontend interaction, backend processing, tool orchestration, real-time communication, and data persistence. This approach enables efficient task automation, scalability, and real-world applicability.

A. System Design Approach

Unlike what we see in simple chatbots, Rexie is an intelligent agent which we have designed with a modular and service oriented architecture. We have put together separate components which we call modules, each with a very defined role. These modules include chat interface, LLM routing logic, tools, an authentication system etc Each module is independent. Also you are able to add or remove tools as you like and in the event that one of them fails it will not bring down the rest.2. Service oriented approach we present each feature as a service which is called out as needed Also we treat every tool and AI model as a service. The bot's ability is to understand what the user is after and which service that is required which it in turn requests.

B. User Interaction Layer

Rexie front end is a chat interface which we have designed around a conversational user experience model very much like what you would see in popular AI chat platforms. This is a choice which we made to present our system in a natural way which does not require technical know how from the user. We put in a centered input bar which also serves to put the user's focus on the conversation at hand, also we display responses in a running chat log.

We use WebSocket technology for real time message streaming which in turn makes responses appear as they come instead of waiting for the full response to load which in turn improves the speed which the user perceives and also increases user engagement. Also we have included a conversation history side bar which allows users to go back to previous sessions and pick up where they left off. We present our users with a choice of which tools to use at any point in time which in turn puts them in the driver's seat as to how tasks are executed. Also we have implemented responsive design, smooth transitions, and we offer both light and dark themes which in turn we have found to improve usability across devices.

We log all of the users' actions that include sending messages, switching between different sessions, or selecting which tools to use and pass those on as structured requests to the backend for processing.

Reference:
WebSocket Communication – RFC 6455 (IETF)

The user uses the bot interface to provide a prompt through the interaction with the chatbot; that prompt is looked at by the LLM which in turn determines the intent and related elements. The identified request is then sent to the MCP server which in that capacity serves as an orchestra for the process. Per the identified intent, the server chooses the proper productivity tool out of Gmail, Slack, SMS, Calendar, or Docs. That tool then does the needed action through its API which in the meantime the associated data and execution logs go into MongoDB for we know and trace what happens.

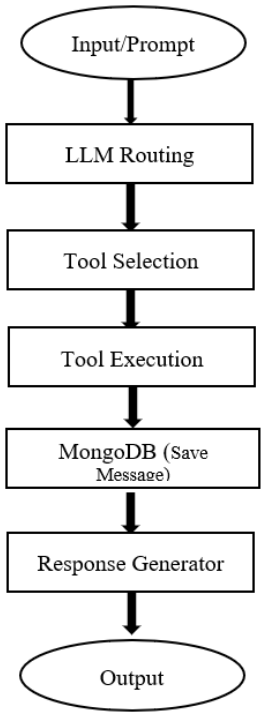

Fig 1. Process Flow Chart

C. Backend Processing and LLM Routing

The backend is designed around an event based request response model which we have put together using

ISSN: 2583-6129
DOI:10.55041/ISJEM06032

FastAPI, a very performant Python web framework for high performance APIs.

Each time a user sends in a message it is done so via a REST endpoint or a WebSocket connection. We include in that which which particular user is that from, what the session details are and also any which tools the user has chosen. This structure of input allows the backend to process the requests accurately and to also maintain the flow of the conversation.

We have put in a dynamic language model routing which goes out to choose the best large language model from the options like Groq or Gemini. The choice is made based on what is available, response time, and also what the user has configured. Once a model is chosen it looks at what the user is trying to do, which external tool if any will be required, which specific tool it should use and what inputs that tool will need. This design what we have put in place is very flexible and does not tie us to a single AI provider.

Reference:
FastAPI Documentation – https://fastapi.tiangolo.com
Large Language Models (LLMs) – Brown et al., 2020

D. Tool Integration Methodology

Rexie integrates a suite of productivity services via a tool-as-a-service model. We have broken each tool out into its own module which we defined by its input variables, what it does, and how it handles errors. This modular design which also happens to be what we did, is that tools may be added, removed or updated without affecting the base system.

We have put in email via Gmail, chat via Slack, calendar management, document and PDF creation, and SMS via Twilio. At run time the language model determines which tool is needed based on what the user is after, we parse out the relevant params from the natural language input and trigger the tool at that point. Once the tool has run we take the results back to the language model which in turn puts together the final reply for the user. We see this as a flexible approach that includes many tools in a single chat, which in turn is able to present complex processes like running a meeting from start to finish in a single interaction.

Reference:
Service-Oriented Architecture – Erl, T. (2005)

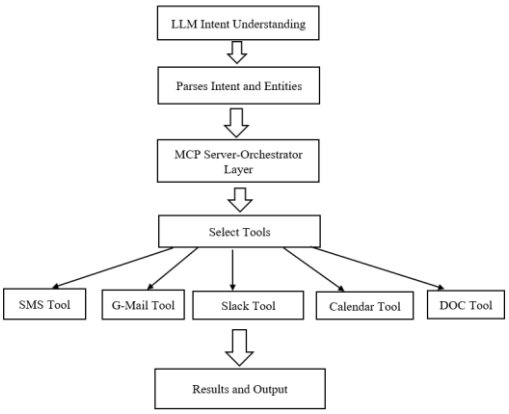

Fig 2. Block diagram

E. Conversation State and Data Persistence

To keep conversation flow seamless Rexie uses a mixed data persistence which includes long term and short term storage. We use MongoDB for the long term storage of conversation history, messages and also tool responses. This in turn allows users to pick up past conversations and to continue the interaction across many sessions. For the short term we use Redis for things like which sessions are active, what the current conversational context is, and the state of WebSockets. What we find is that performance improves with Redis' role in reducing the number of times we hit the database and also in the fast access it provides to data that is used often. As each user interacts with the system we at the same time create or update their conversation record which in turn gives us consistent context.

Reference:
MongoDB Architecture – MongoDB Documentation
Redis In-Memory Data Store – Redis Documentation

F. Real-Time Communication Strategy

Web-sockets in the rexie is used for real-time communication which improves the response time. As soon as the user types in a message, it is processed, and AI responses are out in a token by token fashion as they are generated. We also see that tool execution status is updated in real time which in turn keeps the user informed of what the system is doing at every step. Also we have implemented auto scroll, which keeps the focus on the latest messages thus improving the flow of conversation and the overall user experience. This approach we found to greatly reduce what the user perceives as latency and in turn present a much smoother interaction as compared to the old request-response; models.

G. Security and Authentication Methodology

Security in Rexie is out fit with what the industry uses to keep user data and external service access safe. We use OAuth 2.0 for authentication which in turn gives access to Google services like Gmail, Calendar, and Docs. We secure access tokens and refresh them as needed. Sensitive credentials are managed via environment based configs which in turn which does not put secrets in code. Also we have in place request validation, CORS protection and rate limiting which in turn does the job of prevention for misuse. As for integration of services like Slack and Twilio that is done with the use of scoped access tokens with only required permissions to thus minimize security issues.

Reference:
OAuth 2.0 Framework – RFC 6749

H. Deployment Strategy

Rexie which is very flexible and scalable in its deployment environments. During dev stage the system may be run locally in virtual environments. For consistent and which is also portable deployment we use Docker and Docker Compose to containerize the app. In production, for which we have performance tuned front end deployments, we also go for a back end that is decoupled of databases and cache layers. That which we do is to allow each component to scale out independently based on what is put to them.

Reference:
Docker Documentation – https://docs.docker.com

I. Testing and Evaluation Methodology

We use a large scale of assessment tools and methods in which we conduct functional, system level, and user focused evaluations. In each and every tool we do the functional test to confirm correctness and performance. Also we do end to end chat testing to study on intent detection, tool integration, and response to multi turn interactions.

We evaluate real time performance with the help of stress testing which is done via Web Socket and we simulate many simultaneous users. For the what users experience we do manual evaluation of interface clarity, chat flow and response speed. Also we use metrics like response time, tool performance time, how well it does with many users at once, and database query speed to measure system performance.

IV. EVALUATION

The testing phase involved a comprehensive evaluation focusing on five key areas: Response time, tool performance, scalability during high demand, user experience and total reliability. We did a mix of automated and human based testing which in turn we used to make sure the application is fast, smart and stable for practical use.

Performance study which reported that Rexie puts out average response times of under two seconds for basic AI queries. For requests that involved the execution of external tools, we experienced average response times of 2 to 4 seconds depending on the response time of the third party API. Also we noted that the use of token based streaming which in turn improved the performance of large conversations. For the scale study we found that the system does very well at handling over one hundred concurrent WebSocket connections without any performance issues. Also we saw that the integration of Redis caching did very well in reducing db latency in particular for session and state management ops.

Scalability tests showed that our system is able to handle over one hundred simultaneous WebSocket connections at the same time without issues. We also saw that the Redis cache did very well in reducing database latency which in turn improved performance in the areas of session and state management.

Functional testing was successful in the area of functional evaluation, indicating full functionality of all tools that were integrated. Multi-execution of tools in one conversation was successful, and persistent conversations were successful between different sessions. Authentication using OAuth was secure, and the streaming aspect of each functional test was functioning properly and stable across all functional tests. The methods for error handling that have been implemented were able to properly handle the errors that resulted from API rate limitations and temporary unavailability of service in a graceful manner.

In our manual study of usability we saw that we had great success in the time which it took users to get set up with an app we used as an interface similar to ChatGPT. Also we found the menu based tool selection to be very intuitive and easy for the users. Also that the feature to save conversations did well at breaking up large tasks over time and also enhanced the experience when using dark mode during long sessions. We did see

International Scientific Journal of Engineering and Management (ISJEM)
Volume: 05 Issue: 04 | April – 2026
An International Scholarly || Multidisciplinary || Open Access || Indexing in all major Database & Metadata
ISSN: 2583-6129
DOI:10.55041/ISJEM06032

some small issues come up like delay due to third party API rate limits but they were handled well and we got good info out of those error messages.

The below figure represents the Gmail integration functionality of Rexie. The system successfully sends an email based on the user's instruction. The email contains structured content generated by the AI.

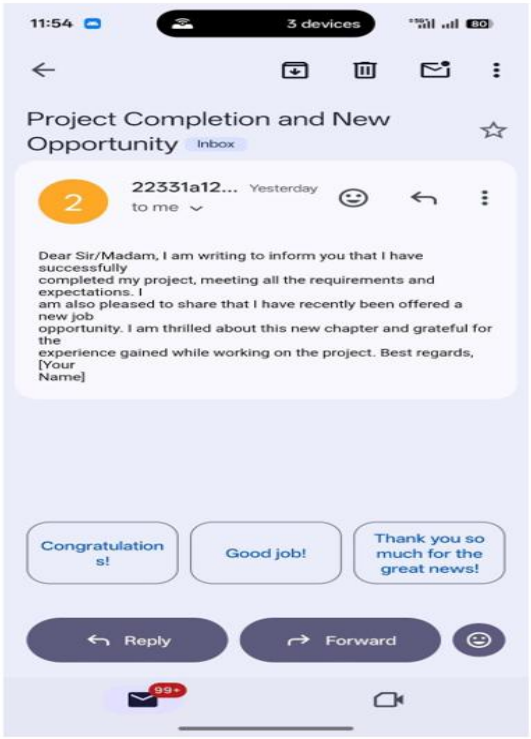

Fig 3. Email Integration output

The below figure shows the Slack integration where messages are sent in a channel. The system posts messages automatically based on user commands. Slack messaging in the Rexie system enables automated communication within team collaboration platforms by integrating with the Slack API. Based on the user's natural language input, the system identifies the intent to send a message and extracts relevant details such as the target channel or user and the message content. The MCP layer selects the Slack tool, and the backend securely communicates with Slack using API tokens to post messages in real time. This allows users to send reminders, updates, or notifications directly to team channels without manually accessing Slack. The integration supports efficient team collaboration by reducing manual effort, enabling instant communication, and allowing task updates to be shared automatically within workflows.

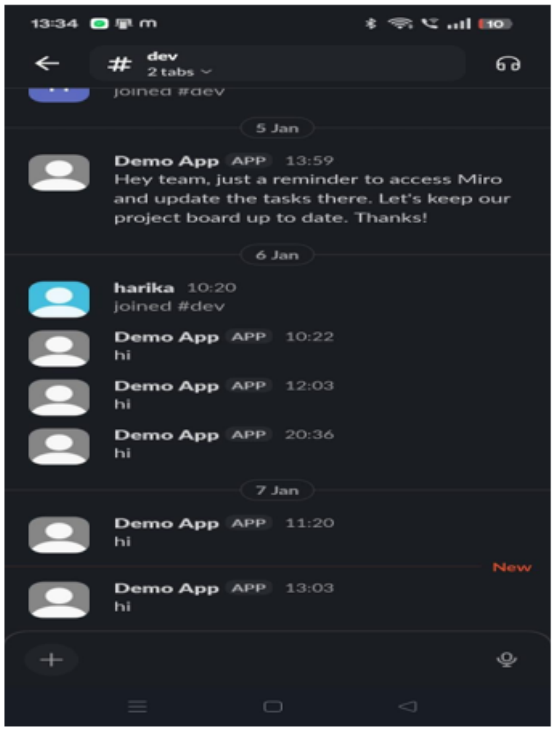

Fig 4. Slack Messaging format

This application has a modular architecture which in turn increases its reliability. Also it is easy for us to add in new features as we go along. Also note that we have included real time data streaming which is a great feature that improves performance. We have enhanced security which we have done via the use of OAuth in addition to other login options. Also it is important to note that application performance is a function of third party service performance which in turn we have no control over. We integrate the APIs into our application which in turn brings together different services. The initial OAuth based auth setup is a do it yourself project and also note that for the SMS and Slack tools you will have to put in extra time to set up their account details which are required for integration.

Rexie has reported to have put together a great success in the development of real time, integrated (multi platform), and very diverse (AI) solutions which in turn report to improve a user's productivity with regard to use of said productivity tools. What we see is the project's results are that which were set out at the start from a functionality and performance point of view which also happen to be very much seen to in a real world productivity setting. Also introduced is a modular architecture designed for growth which contains for instance; features of tool integration, role based access control, and a report of data based on analytics.

VI. CONCLUSION

This study we present to you Rexie AI Bot which is a multi tool AI assistant we designed to put together external services into one conversation interface. We have put together a modular service oriented architecture which in turn we backed with asynchronous communication to support scale and response. We integrated Large Language Model, WebSocket based streaming and independent tool modules which in turn made Rexie to perform tasks across platforms like email, messaging, scheduling, document generation, and SMS sending. In the eval we did it was proven that the system does in fact meet performance and usability requirements which includes low response latency, stable real time streaming and reliable multi task execution. This in turn proves that Rexie is a go to for real world productivity settings and also that it is a flexible solution which we may expand to meet future needs.

Future with regard to Rexie we may see the development of better integration of tools that in turn will include more enterprise and personal productivity software. Also to that we may also see the addition of access control and collaboration features for multiple users which in turn will make it more of a business solution. Also we will work on model routing and caching optimization which in turn will reduce operational expenses and improve consistency. Also to that we may look at adding in analytics that will help to determine user behavior.

VII. REFERENCES

[1] A. Singh, A. Ehtesham, S. Kumar, and T. T. Khoei, "A survey of the Model Context Protocol (MCP)," Preprints.org, 2025, doi: 10.20944/preprints202504.0245.v1.

[2] F. A. Pajo, "Model Context Protocol servers: A novel paradigm for AI-driven workflow automation," Zenodo, 2025, doi: 10.5281/zenodo.17210632.

[3] N. R. Sanikommu, "Model Context Protocol: Enhancing LLM performance for observability and analytics," European Journal of Computer Science and Information Technology, vol. 13, no. 2, pp. 91–112, 2025, doi: 10.37745/ejcsit.2013/vol13n29112126.

[4] K. Parwani, S. Das, and D. K. Vijay, "Model Context Protocol (MCP): A scalable framework for context-aware multi-agent coordination," The Chitransh Academic & Research Journal, 2025, doi: 10.5281/zenodo.17210632.

[5] N. Krishnan, "Advancing multi-agent systems through Model Context Protocol," arXiv preprint arXiv:2504.21030, 2025.

[6] V. Jonnadula et al., "Transforming enterprise AI integration architecture with Model Context Protocol," 2025.

[7] P. P. Ray, "A survey on Model Context Protocol: Architecture, state-of-the-art, challenges and future directions," 2025.

[8] S. Russell and P. Norvig, Artificial Intelligence: A Modern Approach, 4th ed. Upper Saddle River, NJ, USA: Pearson, 2021.

[9] J. Brown et al., "Language models are few-shot learners," in Proc. Adv. Neural Inf. Process. Syst. (NeurIPS), 2020, pp. 1877–1901.

[10] D. Silver et al., "Mastering the game of Go with deep neural networks and tree search," Nature, vol. 529, no. 7587, pp. 484–489, 2016.

[11] L. Wang et al., "LLM-powered agents: Architecture, challenges, and applications," IEEE Intelligent Systems, vol. 39, no. 1, pp. 30–42, 2024.

[12] A. Wooldridge, An Introduction to MultiAgent Systems, 2nd ed. Hoboken, NJ, USA: Wiley, 2009.

[13] M. Chen et al., "Evaluating large language models trained on code," arXiv preprint arXiv:2107.03374, 2021.

[14] S. Li, Y. Zhang, and Q. Yang, "Context-aware AI systems: Design principles and challenges," IEEE Transactions on Knowledge and Data Engineering, vol. 35, no. 4, pp. 3210–3223, 2023.

[15] R. Sutton and A. Barto, Reinforcement Learning: An Introduction, 2nd ed. Cambridge, MA, USA: MIT Press, 2018.

[16] H. Zhou et al., "Tool-augmented large language models for real-world task automation," arXiv preprint arXiv:2307.16789, 2023.

[17] Y. Liu et al., "Prompt engineering and context management for large language models," ACM Computing Surveys, vol. 56, no. 3, pp. 1–36, 2024.

[18] OpenAI, "Function calling and tool use in large language models," 2024. [Online]. Available: https://platform.openai.com/docs

[19] Google, "Large language models and agent-based architectures," Google Research, 2024. [Online]. Available: https://research.google

[20] M. Fowler, Patterns of Enterprise Application Architecture. Boston, MA, USA: Addison-Wesley, 2002.