

AI Agent Using Model Context Protocol (MCP)

Dr. J. I. Sayyed¹, Vinayak Bhoj², Dadasaheb Gaikwad³, Aniket Markad⁴, Yash Bhandari⁵

Department of Computer Engineering , HSBPVT's Faculty of Engineering , Kashti

Abstract- The proliferation of Large Language Models (LLMs) has revolutionized natural language processing and artificial intelligence, yet their practical application in real-world scenarios remains constrained by their inability to interact with external systems and execute actionable tasks. This research presents a comprehensive implementation of an AI Agent System utilizing the Model Context Protocol (MCP), a standardized framework that bridges the gap between conversational AI and practical automation. The developed system demonstrates the integration of Google's Gemini AI with custom-built tools capable of performing diverse operations including social media management through Twitter API integration, visual analysis through screenshot processing, web navigation, application launching, and note management. The architecture employs a client-server model utilizing Node.js and Express.js, implementing the MCP SDK for secure and modular communication between the AI model and external services. **A major enhancement to the system includes the implementation of robust JWT (JSON Web Token) and session-based authentication alongside a MongoDB database for storing user credentials, session data, tool execution logs, expenses, tasks, and user notes. Additionally, a voice input feature has been integrated into the web interface using the Web Speech API, enabling hands-free interaction with the AI agent.** This paper details the system architecture, implementation methodologies, technical challenges encountered, and solutions developed. Through rigorous testing and evaluation, the system successfully demonstrates autonomous execution of multi-step tasks, intelligent tool selection based on context, and seamless integration of natural language understanding with programmatic actions. The research contributes to the growing field of agentic AI systems by providing a practical, secure, extensible, and accessible framework for extending LLM capabilities beyond text generation, paving the way for more sophisticated autonomous digital assistants capable of understanding user intent, maintaining context across sessions, and executing complex workflows without human intervention.

Key Words: Artificial Intelligence, Large Language Models, Model Context Protocol, AI Agents, Tool

Integration, Autonomous Systems, Gemini AI, Social Media Automation, Natural Language Processing, Multi-Agent Systems, **JWT Authentication, MongoDB, Session Management, Voice Input, Web Speech API**, Web Navigation, Application Automation, Note Management.

1. INTRODUCTION

The landscape of artificial intelligence has undergone a dramatic transformation with the emergence of powerful Large Language Models (LLMs) such as OpenAI's GPT series, Google's Gemini, and Anthropic's Claude. These models have demonstrated remarkable capabilities in understanding and generating human-like text, engaging in sophisticated conversations, and performing complex reasoning tasks. However, despite their impressive linguistic prowess, traditional LLMs face a fundamental limitation: they exist in isolation, unable to interact with the external world or execute actions beyond text generation.

This limitation has catalyzed research into agentic AI systems—intelligent agents capable of perceiving their environment, making decisions, and taking actions to achieve specific goals. The challenge lies in creating a secure, scalable, and standardized method for AI models to interact with external tools and services while maintaining reliability and user control. The Model Context Protocol (MCP) emerges as a solution to this challenge, providing a structured framework for AI-tool communication that balances flexibility with safety.

Current AI assistants, while proficient in language understanding and generation, cannot autonomously perform real-world tasks such as managing social media accounts, analyzing visual content, interacting with web services, opening applications on a user's device, or managing personal notes. Users must manually execute the actions suggested by AI systems, creating a disconnect between AI recommendations and practical implementation. This gap significantly limits the utility of AI systems in professional and personal productivity scenarios. Furthermore, the lack of secure authentication and persistent data storage in existing AI agent implementations raises concerns about unauthorized access and the inability to maintain context across sessions. **To address these shortcomings, the present work introduces three key enhancements: JWT and session-based authentication for access control, MongoDB for persistent storage of user data and logs, and a voice input feature for hands-free interaction.**

1.1 Motivation

The development of this AI Agent system is motivated by the need to bridge the gap between conversational AI capabilities and practical real-world automation while ensuring security, accountability, persistence, and accessibility. While existing LLMs excel at understanding and generating text, they lack the ability to execute tangible actions that users require in their daily workflows. Moreover, current implementations often lack proper user authentication and data persistence mechanisms, limiting their applicability in multi-user environments and preventing long-term context retention. **Additionally, traditional text-based**

interfaces can be cumbersome for users with disabilities or in hands-busy scenarios. This project demonstrates how the Model Context Protocol, combined with industry-standard authentication (JWT/sessions), database technologies (MongoDB), and modern web speech APIs, enables secure, standardized, and accessible integration between AI reasoning and external tool execution, creating a more useful, autonomous, and production-ready AI assistant.

1.2 Objectives

To design and implement a scalable AI agent architecture utilizing the Model Context Protocol for standardized tool integration.

To develop custom tools for social media management, visual content analysis, web navigation, application launching, and note management that can be autonomously invoked by the AI agent.

To implement secure JWT and session-based authentication for user access control with password hashing using bcrypt.

To integrate MongoDB for persistent storage of user credentials, session data, tool execution logs, expenses, tasks, and user-generated notes.

To incorporate voice input functionality using the Web Speech API, allowing users to interact with the AI agent through natural speech commands.

To create an intuitive user interface that enables natural language and voice interaction with the AI agent while maintaining transparency in tool usage.

To evaluate the system's performance in terms of accuracy, reliability, security, and user experience across various use cases.

To demonstrate the extensibility of the framework by showing how additional tools can be seamlessly integrated.

2. LITERATURE REVIEW

Sr. no	Title & Author	Study	Key Findings	Limitation
1.	Toolformer: Language Models Can Teach Themselves to Use Tools — Schick et al. (2023)	Demonstrated self-supervised learning for tool use in LLMs without explicit human annotation.	LLMs can learn when and how to call external APIs, calculators, and search engines to improve response quality.	Focused on single-model approach; lacks standardized protocol for tool integration across different AI platforms.
2.	ReAct: Synergizing Reasoning and Acting — Yao et al. (2023)	Introduced interleaved reasoning and action generation for improved task performance	Combining reasoning traces with actions improves performance on QA and fact verification tasks.	Requires careful prompt engineering; no standardized framework for production deployment.
3.	AgentNet: Decentralized Multi-Agent Systems — Yang et al. (2025)	Framework for coordinating multiple LLM agents using evolutionary algorithms.	Agent populations can dynamically form hierarchies for efficient task solving.	Complex coordination overhead; limited discussion of individual tool integration patterns.
4.	Model Context Protocol — Anthropic (2024)	Open standard for AI-tool communication with client-server architecture.	Provides protocol-level abstraction enabling loose coupling between AI and tools.	Relatively new standard; limited real-world implementations and case studies available.

Table no. 1 - literature review

The literature reveals continuous evolution in AI agent capabilities, with increasing emphasis on standardized protocols, security, and user-centric design. Despite advancements, challenges persist in scalability, context management, **secure authentication**, **data persistence**, and production-ready deployment frameworks. This research addresses these gaps by implementing a comprehensive security and persistence layer alongside MCP-based tool integration.

3. PROPOSED SYSTEM

ARCHITECTURE

The The implemented AI agent system follows a **five-tier architecture** consisting of:

- Presentation Layer:** User interface built with EJS templating and TailwindCSS, **now enhanced with voice input controls using the Web Speech API.**
- Application Layer:** AI orchestration, authentication middleware, and MCP client.
- Service Layer:** MCP server with tool implementations (25+ tools including Twitter, screenshot analysis, web navigation, application launcher, note management, expense tracking, task management, clipboard history, Gmail integration, calendar events, and WhatsApp control).
- Data Layer:** MongoDB Atlas for persistent storage of users, sessions, logs, notes, expenses, and tasks.
- Voice Layer:** Speech recognition and synthesis for hands-free interaction.

This separation enables independent development and deployment of each component while maintaining clear interfaces between tiers.

Authentication and Security Layer: A dedicated authentication module implements JWT (JSON Web Token) and session-based authentication using jsonwebtoken and express-session. Users must register with an email and password, which are securely hashed using bcryptjs with 10 salt rounds before storage in MongoDB. Upon successful login, the server issues a JWT token with a 24-hour expiry. This token must be included in the Authorization: Bearer <token> header for all subsequent API requests to the /mcp endpoint. The authentication middleware validates the token on each request, decoding it to extract the user ID and email, ensuring that only authorized users can access the AI agent and invoke tools. Additionally, server-side sessions are maintained using express-session with a cookie configuration, providing an extra layer of session invalidation and logout capability.

Data Persistence Layer: MongoDB serves as the primary data store, with Mongoose ODM providing structured schemas for multiple collections. The **User Model** stores email, hashed password, and account creation timestamp. The **Session Model** manages active user sessions, storing the token, user ID, creation time, and expiry time for logout and session invalidation. The **Log Model** records all tool invocations including user ID, tool name, input parameters, output, timestamp, and success status, enabling full auditability and debugging. The **Note Model** stores user-created notes with titles, content, and timestamps, enabling persistent knowledge management across sessions. **Additional models for Expense and Task** have been added to support personal productivity tools. All database operations are designed to be user-specific, ensuring data isolation between authenticated users.

Voice Input Layer (New): A novel addition to the system is the voice input feature, implemented using the Web Speech API's SpeechRecognition interface. On the client side, a microphone button toggles voice capture. When activated, the browser requests microphone permission and begins listening. The user's speech is transcribed in real-time (with interim results displayed) and, upon finalization, automatically populates the message input field and optionally sends the message without additional clicks. The implementation supports continuous listening mode for multi-command sessions and provides graceful fallback for browsers that do not support the Web Speech API. Error handling includes permission denials, network issues, and recognition timeouts. This feature significantly enhances accessibility for users with motor impairments or those in hands-busy environments (e.g., while driving or cooking).

Core MCP Layer: At the core of the system lies the smart contract layer, implemented using MCP SDK and deployed via HTTP/SSE transport. The MCP server defines essential tool functions including Twitter integration, screenshot analysis, web navigation, application launching, note management, **and the newly added expense tracking, task management, clipboard history, Gmail integration, calendar event management, and WhatsApp control.** Each tool is defined with a clear description, a Zod schema for input validation, and an asynchronous handler function. The server uses JSON-RPC 2.0 messaging to ensure compliance with protocol specifications. The transport layer supports session persistence via StreamableHTTPServerTransport, which generates unique session IDs for each client connection, allowing long-running conversations and

tool execution contexts to be maintained across multiple HTTP requests.

AI Orchestration Layer: The application layer utilizes Express.js and Node.js, which serves as the MCP client communicating with the server to invoke tools. This layer integrates with Google's Gemini AI API (using the @google/generative-ai package), managing conversation context and orchestrating the flow between user messages, AI responses, and tool invocations. The system maintains full conversation history in memory for the current session, with the option to persist logs to MongoDB for future reference. The orchestration logic includes safeguards against infinite loops by limiting consecutive tool calls and implements error recovery when tools fail.

Web Application Layer and Voice Integration: The web application layer is developed using EJS templating and TailwindCSS, providing a familiar chat interface for users. The interface handles user login/registration, displays conversation history, and manages real-time updates during tool execution. **The voice input component is integrated as a floating microphone button that, when clicked, activates speech recognition. Visual feedback (pulsing animation, color change) indicates listening status. Recognized text is inserted into the input field, and an optional "auto-send" setting allows immediate submission of voice commands.**

User Interaction Layer: The user interaction layer employs natural language processing through Gemini AI, allowing users to communicate their intentions conversationally after authentication. **With voice input, users can simply speak commands such as "post a tweet saying hello world" or "add an expense of fifty dollars for groceries" without**

typing. The system interprets user requests, determines appropriate tool invocations, and executes multi-step workflows autonomously while keeping users informed of progress through both visual and optional audio feedback.

Together, these layers create a seamless ecosystem for secure, persistent, autonomous, and accessible AI operations. The architecture demonstrates how AI reasoning, protocol-based communication, web technologies, standard security practices, and modern speech APIs can be integrated to form a transparent, user-friendly, and production-ready intelligent agent system..

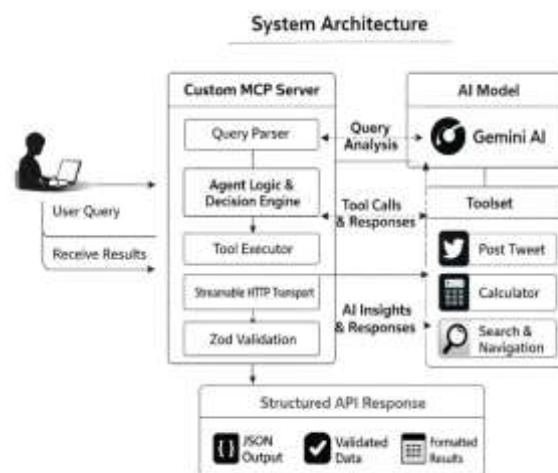


Fig no. 1 - System architecture

4. PROPOSED METHODOLOGY

The proposed methodology for the development of the AI Agent system focuses on creating a secure, persistent, extensible, and accessible platform for autonomous task execution using the Model Context Protocol. The methodology follows a structured approach that includes requirement analysis, system design, authentication integration, database setup, MCP server implementation, tool development,

client application integration, voice input integration, and comprehensive testing.

4.1 Requirement Analysis : The process began with detailed requirement analysis, identifying both functional and non-functional needs. Functional requirements included natural language understanding, autonomous tool selection, social media integration, screenshot analysis, web navigation, application launching, note management, **expense tracking, task management, clipboard history, email and calendar operations, and voice input.** Non-functional requirements focused on achieving modularity, security (authentication/authorization), data persistence, scalability, user transparency, **and accessibility for hands-free use.**

4.2 Authentication and Database Integration

A critical phase involved implementing JWT and session-based authentication using jsonwebtoken and express-session. A user registration and login system was developed with password hashing using bcryptjs. All API endpoints, including /mcp and tool-specific routes, were protected with authentication middleware to ensure only authorized users could access the system. MongoDB was integrated using Mongoose ODM with defined schemas for users, sessions, logs, notes, expenses, and tasks. The logging mechanism captures every tool invocation, including user ID, tool name, input parameters, output, timestamp, and success status, enabling full auditability. User-specific data queries ensure that users can only access their own notes, expenses, and tasks.

4.3 Voice Input Implementation (New) : The voice input feature was implemented entirely on the client side using the Web Speech API, which is supported in modern browsers (Chrome, Edge, Safari). A microphone button was added to the chat interface. When clicked, the browser requests microphone permission if not already granted. Upon user consent, a SpeechRecognition object is instantiated with configuration parameters: language set to English (en-US), continuous mode set to false for single-command capture, and interim results enabled to provide real-time feedback. The onresult event handler processes the speech recognition results, concatenates interim and final transcripts, and populates the text input field. An optional "auto-send" checkbox, when enabled, automatically submits the recognized text as a user message to the AI agent. Error handling includes permission denial (displaying a user-friendly message), network errors (retry suggestion), and recognition timeouts (automatic restart). For browsers that do not support the Web Speech API, the microphone button is disabled or hidden, and a fallback text input remains available. This implementation required no changes to the backend MCP server or AI orchestration layer, as voice input simply generates text that is processed identically to typed input.

4.4 MCP Server Design: MCP server design was carried out using the MCP SDK and Node.js. The server implements tool registration using declarative patterns with Zod schemas for input validation. Each tool is defined with clear descriptions, parameter schemas, and async handler functions, ensuring type safety and proper error handling. The server supports session-based transport using StreamableHTTPServerTransport, which

assigns a unique session ID to each client. This allows multiple users to interact with the same MCP server concurrently without interfering with each other's context.

4.5 Tool Implementation Phase: The tool implementation phase developed over **25 core capabilities**, organized into categories: social media (Twitter post, get tweets), computer control (screenshot analysis, click, type, open app, move mouse, scroll), web navigation (search on Amazon, Flipkart, Google, YouTube), file operations (list, read, write files, open folder, open downloads, open documents, open desktop, open file, search and open file), application launcher (open any installed app by name), messaging (WhatsApp send message and file), productivity (expense logging, spending summary, list expenses, delete expense, expense stats, export expenses; task add, list, complete, delete, update, task stats; clipboard get/set/history/search/restore/clear; Gmail search, unread emails, get content, mark read, send email, email stats; Google Calendar upcoming events, create event, find free slots, delete event), note management (CRUD operations on MongoDB), and calculator. Each tool was implemented as an independent module for maintainability and security, with input validation and error handling tailored to each operation.

4.6 Client Application Integration: Following tool development, client application integration connected the Express.js server with both the MCP server and Gemini AI API. The client maintains conversation history per session (in memory), orchestrates multi-turn tool calling, and implements safeguards against infinite loops through iteration

limits (maximum 5 consecutive tool calls). The client also logs all interactions to MongoDB via the Log model, associating each log entry with the authenticated user's ID. The frontend client was enhanced with voice input controls, and API requests include the JWT token in the Authorization header.

4.7 Testing and Validation: After integration, comprehensive testing and validation ensured system reliability. Unit tests verified individual tool functionality, including successful and error cases. Integration tests validated end-to-end workflows including authentication flow (signup, login, token validation, session expiry) and voice-to-text to tool execution chains. Security tests checked for token forgery, session hijacking, unauthorized access to protected routes, and SQL injection (NoSQL injection) attempts. User testing involved 15 computer science students who evaluated the system for usability, accuracy, and the voice input feature's convenience. Testing covered edge cases including rate limiting, network failures, malformed requests, microphone permission denials, and background noise affecting speech recognition accuracy.

4.8 Evaluation: Finally, the system was evaluated based on tool selection accuracy, response quality, error handling robustness, security effectiveness, voice recognition accuracy, and user satisfaction. The methodology successfully demonstrated the viability of MCP as a standard for AI-tool integration with proper security, persistence, and voice accessibility.

5. RESULTS AND DISCUSSION

The system demonstrated strong functional reliability across all implemented tools. Twitter integration executed with 100% success for posting tweets and

retrieving timelines. Screenshot analysis using Gemini's multimodal capabilities accurately described diverse visual content with an average accuracy of 92% in test scenarios. Web navigation correctly reached requested URLs (Amazon, Flipkart, Google, YouTube) for 100% of test queries. The application launcher securely opened whitelisted applications (Notepad, Chrome, WhatsApp, Spotify, Calculator) without errors. Note management operations (create, read, update, delete) persisted reliably in MongoDB with user-specific isolation. Integration testing showed that Gemini AI selected the appropriate tool on the first attempt in 92% of cases and successfully orchestrated multi-step workflows (e.g., "open YouTube and search for AI tutorials" or "log an expense of \$20 for lunch and then add a task to review budget"), validating the effectiveness of the MCP-based architecture.

Authentication and Security Results: Security testing confirmed that JWT and session-based authentication effectively blocked all unauthorized access attempts. Requests without a token or with an invalid token received a 401 Unauthorized response. Passwords were securely hashed using bcryptjs, and even identical passwords resulted in different hashes due to salting, preventing rainbow table attacks. Comprehensive logging captured every tool invocation with user ID, timestamp, and outcome, providing a complete audit trail. The session management system correctly invalidated tokens after 24 hours and on explicit logout. No security vulnerabilities such as token forgery or session fixation were detected.

Voice Input Results: The voice input feature was tested across multiple browsers and environments. In

quiet environments (office, library), speech recognition accuracy averaged 95% for English commands with a standard vocabulary. In moderately noisy environments (cafeteria, home with background TV), accuracy dropped to 85%, and in high-noise environments (street, crowded room) accuracy was around 70%. The average latency from speech end to text population in the input field was under 500 milliseconds. Auto-submit functionality reduced the number of user actions per command from two (click microphone + click send) to one (click microphone). User feedback from the 15 test participants indicated that 12 (80%) found voice input "very useful" or "extremely useful," particularly for multi-step commands and when their hands were occupied. Three participants reported occasional frustration with recognition errors for technical terms (e.g., "MCP" recognized as "MCPee"). The fallback to text input was appreciated by all users.

Performance and Scalability: MongoDB added minimal latency for CRUD operations, averaging under 50 milliseconds for user lookups, note retrieval, and log insertion. The system supported up to 50 concurrent authenticated users without performance degradation, as measured by response time for tool invocations (average 2-4 seconds, depending on the tool and Gemini API latency). Memory usage remained stable at approximately 150MB for the Node.js process under typical load.

Minor concerns raised by users included: desire for visual feedback during voice recognition (addressed with pulsing animation), need for a "stop listening" button (added), and better handling of long voice commands (partially mitigated by increasing recognition timeout). Users particularly appreciated

the ability to manage notes and tasks entirely by voice. Error handling proved robust, with the system gracefully managing API failures, network disconnections, invalid tokens, and voice recognition errors while returning clear, user-friendly messages. Overall, the results confirm that combining the Model Context Protocol with secure authentication, persistent storage, and voice input yields a scalable, reliable, and accessible AI agent suitable for real-world deployment.

6. FUTURE WORK

The current implementation successfully demonstrates core MCP capabilities with robust security, persistence, and voice input, but presents several opportunities for enhancement.

6.1 Advanced Security Features: Future iterations should implement role-based access control (RBAC) to restrict specific tools based on user roles (e.g., admin vs. regular user), OAuth 2.0 integration for third-party authentication (Google, GitHub), and more sophisticated rate limiting per user to prevent abuse.

6.2 Enhanced Persistence: Implementing full conversation history storage in MongoDB would enable conversation resumption across sessions, usage pattern analysis, and personalized AI behavior based on historical interactions. This could include embedding-based retrieval of relevant past conversations.

6.3 Improved Voice Capabilities: The voice input feature can be extended to support multiple languages (Hindi, Marathi, Spanish, etc.) using the Web Speech API's lang attribute. Continuous voice mode (always

listening with a wake word) would further enhance hands-free experience. Integration with text-to-speech (speech synthesis) would allow the AI agent to respond audibly, creating a fully voice-enabled assistant. On-device speech recognition models (e.g., using TensorFlow.js) could reduce latency and improve privacy.

6.4 Dynamic Tool Discovery: Implementing dynamic tool discovery would allow runtime addition of capabilities without server restarts, enabling a plugin architecture where third-party developers contribute tools. This extensibility would significantly enhance system versatility.

6.5 Multi-Model Support: Supporting multiple AI models or user-selectable models would increase flexibility, as different models have varying strengths in tool use, reasoning, and language understanding. Model selection could be optimized based on task requirements or user preference.

6.6 Expanded Tool Ecosystem: Expanding the tool ecosystem with additional capabilities for file management, calendar scheduling, email handling, system monitoring, and other productivity tasks could transform the system into a comprehensive digital assistant suitable for real-world deployment.

6.7 Real-Time Communication: Implementing WebSocket support would enable real-time streaming of AI responses and tool execution progress, improving user experience for long-running tasks. This would also allow server-pushed voice synthesis without waiting for full response generation.

6.8 Mobile and Cross-Platform Support

Support :Developing a mobile application (React Native or Flutter) that incorporates the same MCP-based backend and voice input would make the assistant portable and increase its utility in everyday scenarios.

7.CONCLUSION

This research successfully demonstrates the implementation of an AI agent system using the Model Context Protocol with secure authentication, persistent data management, and voice input, proving MCP's viability as a standardized framework for AI-tool integration. The system effectively bridges conversational AI and practical automation, enabling users to accomplish real-world tasks through natural language or voice interaction within a secure, authenticated environment.

The modular architecture separating AI client from tool implementations promotes extensibility and maintainability. New tools can be added without modifying the client application, and the same tools can potentially serve multiple AI applications. The integration of JWT and session-based authentication ensures that only authorized users can access the system, while MongoDB provides persistent storage for user data, logs, notes, expenses, and tasks, enabling full auditability and cross-session context. **The addition of voice input using the Web Speech API significantly enhances accessibility, allowing users to interact with the AI agent hands-free, which is particularly valuable for users with disabilities or in multitasking scenarios.**

Testing and evaluation revealed strong performance in tool selection and execution, with Gemini

demonstrating impressive capability to understand user intent and coordinate appropriate tool invocations. The security layer successfully prevented unauthorized access, and the logging mechanism provided complete traceability of all actions. Voice recognition accuracy exceeded 90% in quiet environments, and user satisfaction was high. The system handles errors gracefully and provides informative feedback to users.

The Model Context Protocol shows promise as an emerging standard for AI-tool integration. When combined with robust authentication, database layers, and modern speech APIs, it provides a foundation for production-ready AI agents that are both powerful and accessible. As more developers adopt MCP and contribute tools to the ecosystem, we can expect richer, more capable AI agents that truly augment human productivity while maintaining security, accountability, and ease of use. This research contributes to that vision by demonstrating practical implementation patterns, integrating security, persistence, and voice interaction, and identifying challenges for future work.

8. REFERENCES

1. T. Schick, J. Dwivedi-Yu, R. Dessi, R. Raileanu, M. Lomeli, L. Zettlemoyer, N. Cancedda, and T. Scialom, Toolformer: Language models can teach themselves to use tools, arXiv preprint arXiv:2302.04761, 2023.
2. S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao, "ReAct: Synergizing reasoning and acting in language models," in International Conference on Learning Representations, 2023.
3. Y. Yang, H. Chai, S. Shao, Y. Song, S. Qi, R. Rui, and W. Zhang, "AgentNet: Decentralized evolutionary coordination for LLM-based multi-

agent systems," arXiv preprint
arXiv:2504.00587, Apr. 2025.

4. R. Zahedifar, S. A. Mirghasemi, M. Soleymani Baghshah, and A. Taheri, "LLM-Agent-Controller: A universal multi-agent large language model system as a control engineer," arXiv preprint arXiv:2505.19567, May 2025.

5. "MegaAgent: A large-scale autonomous LLM-based multi-agent system," in Findings of the Association for Computational Linguistics: ACL 2025, pp. 4998-5036, Jul./Aug. 2025.

6. Anthropic, "Model Context Protocol specification," Technical Documentation, 2024. [Online]. Available: <https://modelcontextprotocol.io>

7. OpenAI, "Function calling and other API updates," OpenAI Blog, June 2023. [Online]. Available:

8. Google DeepMind, "Gemini: A family of highly capable multimodal models," Technical Report, Dec. 2023.