

DEVELOPMENT OF A MULTI-AGENT FRAMEWORK BASED ON MODEL CONTEXT PROTOCOL

A PROJECT REPORT

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE AWARD OF THE DEGREE
OF

MASTER OF TECHNOLOGY
IN
ARTIFICIAL INTELLIGENCE

Submitted by

PRASHANT KUMAR SINGH (24/AFI/19)

Under the supervision of

PROF. DR. RAHUL KATARYA



COMPUTER SCIENCE AND ENGINEERING
DELHI TECHNOLOGICAL UNIVERSITY
(Formerly Delhi College of Engineering)
Bawana Road, Delhi 110042

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
DELHI TECHNOLOGICAL UNIVERSITY
(Formerly Delhi College of Engineering)
Bawana Road, Delhi-110042

CANDIDATE’S DECLARATION

We, Prashant Kumar Singh, Roll No’s – 24/AFI/19 students of M.Tech (Artificial Intelligence), hereby declare that the project Dissertation titled “Development of a Multi-Agent Framework Based on Model Context Protocol” which is submitted by us to the Computer Science and Engineering, Delhi Technological University, Delhi in partial fulfillment of the requirement for the award of degree of Master of Technology, is original and not copied from any source without proper citation. This work has not previously formed the basis for the award of any Degree, Diploma Associateship, Fellowship or other similar title or recognition.

Place: Delhi

Prashant Kumar Singh

Date: dd.mm.yy

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
DELHI TECHNOLOGICAL UNIVERSITY
(Formerly Delhi College of Engineering)
Bawana Road, Delhi-110042

CERTIFICATE

I hereby certify that the Project Dissertation titled “Development of a Multi- Agent Framework Based on Model Context Protocol” which is submitted by Prashant Kumar Sing, Roll No’s – 24/AFI/19, Computer Science and Engineering ,Delhi Technological University, Delhi in partial fulfilment of the requirement for the award of the degree of Master of Technology, is a record of the project work carried out by the students under my supervision.

Place: Delhi

Prof. Dr Rahul Katarya

Date: 06/2026 **SUPERVISOR**

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
DELHI TECHNOLOGICAL UNIVERSITY
(Formerly Delhi College of Engineering)
Bawana Road, Delhi-110042

ACKNOWLEDGEMENT

We wish to express our sincerest gratitude to Prof. Dr Rahul Katarya for his continuous guidance and mentorship that he provided us during the project. He showed us the path to achieve our targets by explaining all the tasks to be done and explained to us the importance of this project as well as its industrial relevance. He was always ready to help us and clear our doubts regarding any hurdles in this project. Without his constant support and motivation, this project would not have been successful.

Place: Delhi

Prashant Kumar Singh

Date: /06/2026

Abstract

The aim of this project report is to create a practical, extensible and modular AI assistant that can execute real-world productivity tasks, using the Development of a Multi-Agent Framework Based on Model Context Protocol. In a traditional chatbot setup, you'd have one big LLM responsible for all the interactions, resulting in context pollution, poor tool selection, limited scalability, and lack of maintainability when you integrate various domains like email, calendar, web search, weather, memory, file handling, and voice interaction.

The proposed framework tackles these problems by means of a multi-agent architecture, driven by a supervisor. Each user requests is analysed by a central supervisor agent and the most appropriate specialist agent is used. These dedicated agents include calendar management, e-mail automation, web searching, weather retrieval, general assistance, memory operations and desktop file interaction. All agents employ domain-specific instructions and tools, making them more reliable and reducing the need to load unnecessary context.

One of the key areas of this work is on the extension mechanism in the form of Model Context Protocol (MCP) for external capability servers. One important aspect of this work is on Model Context Protocol (MCP) as an extension mechanism for connecting external capability servers. The framework enables services like Gmail retrieval or access to files on the desktop to be made available via a uniform interface, allowing the framework to be extended without altering the basic flow. The system also operates an email subsystem that has a human in the loop, where emails are drafted, reviewed, re-written, if necessary, and only sent upon explicit consent.

The framework has been designed to interact with text and voice using Streamlit or command-line interface. It also supports short-term memory via LangGraph checkpointing and long-term memory via SQLite FTS5.

In conclusion, the project showcases the feasibility of assembling a practical and scal-

able framework of an AI assistant by integrating multiple agents, multimodal interaction, memory management, tool-augmented reasoning, and extensibility via MCP.

Contents

Candidate's Declaration	i
Certificate	ii
Acknowledgement	iii
Abstract	v
Content	viii
List of Tables	ix
List of Figures	x
1 INTRODUCTION	xi
1.1 Background and Motivation	xi
1.2 Problem Statement	xi
1.3 Project Objectives	xii
1.4 Project Scope	xii
2 Literature Review and Technical Background	xiii
2.1 Overview	xiii
2.2 Large Language Models	xiii
2.3 Limitations of Monolithic Chatbots	xiii
2.4 Multi-Agent Systems	1
2.5 LangGraph and Tool Integration	1
2.6 Model Context Protocol	1
2.7 Memory and Voice Interaction	1
2.8 Requirements Analysis and Design Objectives	2
2.8.1 Functional requirements	2
2.8.2 Non-Functional Requirements	2
2.9 Design Objectives	2
2.10 System Constraints	4
2.11 System Architecture	4
2.12 Overview	4
2.13 Architectural Layers	4
2.13.1 User Interface Layer	4
2.13.2 Voice Interaction Layer	4
2.13.3 Supervisor Layer	5
2.13.4 Specialist Agent Layer	5
2.13.5 Tool Integration Layer	5

2.13.6	MCP Extension Layer	5
2.13.7	Memory and Storage Layer	6
2.14	Architecture Flow	6
2.15	Model Context Protocol	6
2.16	Benefits of the Proposed Architecture	7
2.17	Summary	7
3	Methodology	8
3.1	Introduction	8
3.2	Supervisor Agent	8
3.3	Calendar Agent	9
3.4	Email Agent	9
3.5	Weather Agent	10
3.6	Search Agent	10
3.7	Memory and General Agent	11
3.8	Tool Integration	12
3.9	Agent and Tool Mapping	12
4	Implementation Details	13
4.1	Introduction	13
4.2	Technology Stack	13
4.3	Environment Configuration	13
4.4	LLM Configuration	14
5	Testing, Results, and Discussion	16
5.1	Introduction	16
5.2	Testing Strategy	16
5.3	Supervisor Routing Validation	17
5.4	Agent-Level Testing	17
5.4.1	Calendar Agent Testing	17
5.4.2	Email Agent Testing	18
5.4.3	Weather Agent Testing	18
5.4.4	Web Search Agent Testing	19
5.4.5	General and Memory Agent Testing	19
5.5	MCP Server Validation	19
5.6	Voice Interaction Testing	20
5.7	User Interface Validation	20
5.8	Screenshots and Visual Observations	21
5.8.1	Main Agent Interface	21
5.8.2	Developer Console Interface	21
5.9	Limitations Observed During Testing	21
5.10	Summary	22
5.11	Summary	23
6	Limitations and Future Scope	25
6.1	Introduction	25
6.2	Current Limitations	25
6.2.1	Single-User Design	25
6.2.2	Limited Security Controls	25

6.2.3	Dependency on External APIs	25
6.2.4	Limited Long-Term Memory Intelligence	26
6.2.5	Voice Recognition Limitations	26
6.2.6	Limited Evaluation Metrics	26
6.2.7	Basic MCP Integration	26
6.3	Future Scope	26
7	Conclusion	28
7.1	Project Contribution	29
7.2	Final Outcome	29

List of Tables

2.1	Functional requirements of the proposed assistant	2
2.2	Non-functional requirements and design responses	3
3.1	Agent and Tool Mapping	12
4.1	Technology Stack Used in the Proposed Framework	14
5.1	Supervisor Routing Validation	17

List of Figures

4.1	Environment configuration using	14
5.1	Main Agent Interface	21
5.2	Developer Console Interface	22

Chapter 1

INTRODUCTION

1.1 Background and Motivation

AI, more specifically large language models, has been responsible for great changes in software experience. Language models such as GPT, Gemini, and transformers are capable of not just text generation, but question answering, summarization, coding, reasoning, and much more. The above advancements have led to the development of smart assistants in fields like productivity, communication, data collection, decision-making, and many others.

In the traditional sense, AI assistants have been designed using one method, whereby everything is handled by the same model or prompt. An AI assistant would fail to handle multiple tasks like email automation, scheduling, web search, weather information, memory tasks, file handling, voice commands, and many others. Some of the challenges associated with such an assistant include overloading of contexts, misuse of tools, among others.

Such a multi-agent system architecture can cope with these problems. To overcome these mentioned problems, the use of modular multi-agent system architecture will be implemented in this project. It is worth mentioning that the architecture being proposed includes a super agent. The idea here is that the tasks will be delegated to different agents which will increase the reliability, scalability and security of the application. “A smart assistant that understands your intent and connects you with the right expert who can help you reach your goal.” agent[3, 4, 12].

As seen from the above, extensibility of the architecture will be improved by implementing external tools such as Gmail, desktop files etc., as interoperability with them will be achieved using MCP. It is important to add that the aim of this particular project is to develop a voice-activated intelligent agent capable of performing actions in the real world.

1.2 Problem Statement

The problem arises when the single model tries to perform such functions as detecting user intention, finding right tools, handling memories, connecting to external sources and creating answers. Adding further tasks and instructions causes pollution of context, incorrect task distribution, errors, hallucinations and inability to scale.

The idea of the present project is to solve the mentioned problems by means of developing multi-agent modular system of personal AI assistant using Model Context Protocol. The main goal of the supervisor agent would be to detect the intention of the user and

give right tasks to relevant agents, such as email, calendar, weather, internet, memory agents, etc.

1.3 Project Objectives

The main goal of this project is to create a multi-agent framework based on Model Context Protocol (MCP) that can complete real-world productivity tasks through intelligent coordination and tool integration. The specific objectives of the project are as follows:

- To develop a supervisor-based multi-agent architecture for directing user requests to the right specialist agents.
- To use dedicated agents for tasks such as calendar management, email automation, weather retrieval, web search, general assistance, memory operations, and file interaction.
- To include Model Context Protocol for connecting external capability servers, such as Gmail retrieval and desktop file access.
- To create a human-in-the-loop email automation process where emails are written, reviewed, regenerated if needed, and sent only after user approval.

1.4 Project Scope

This project focuses on designing and implementing a single-user AI framework for productivity and automation assistance. The system is built around modular agents, tool integration, memory handling, extensibility through Model Context Protocol (MCP), and text-based as well as voice-based user interaction.

The framework includes features such as calendar operations, email drafting and sending, email retrieval from Gmail, web search, weather information retrieval, desktop file access, and memory-based personalization. Additionally, it provides a user interface developed with Streamlit and a command-line interface for direct interaction.

This architecture was not created or optimized to be implemented in any real-life business setting that requires numerous users as well as proper security implementation. Rather, the architecture will serve only as a theoretical experiment with regard to the creation of an intelligent assistant via multi agent systems and MCP.

Chapter 2

Literature Review and Technical Background

2.1 Overview

Intelligent assistants have evolved very fast due to development in areas such as large language models, multi-agent architecture, tool reasoning, and integration using standard protocols. Whereas initial solutions in chatbots focused on the capability of conducting a conversation, today's intelligent assistants must manage a series of tasks that include emailing, calendar management, internet browsing, weather data lookup, file manipulation, preferences, and voice communication [1, 2].

As noted, some of these tasks cannot be accomplished through the utilization of chatbot architecture alone. There should be good collaboration between modules within the system. This chapter will focus on the technical aspects of large language models, multi-agent systems, LangGraph orchestration, tool integration, Model Context Protocol, memory management, and voice communication.

2.2 Large Language Models

Large Language Models are examples of such systems as GPT, Gemini, and other systems built on top of transformers, which have been thoroughly trained on vast text datasets. Such models are able to comprehend natural language prompts and provide appropriate responses. The use cases for Large Language Models include question answering, text summarization, generation of content, reasoning, and programming [1, 2].

LLM cannot act in real-time, however. It cannot send emails, check calendars, access local storage, get weather information, and do other tasks without using additional services. This is why modern AI assistants incorporate such models together with tool calling, memory, and workflows.

2.3 Limitations of Monolithic Chatbots

Monolithic chatbots' issues are in using either a single model or single large prompt that will take care of the needs of all kinds of users. While this might sound like a smart solution, as more and more functionality gets included in it, it becomes less reliable because the model will get confused by all of these prompts.

Some of the biggest problems that can arise in building monolithic bots include too many contextues, improper usage of software tools, lack of specialization, and the challenge of maintenance. For instance, it is impossible to fine-tune the same prompt to

compose letters, get weather forecasts, browse the Internet, and schedule meetings. In addition, there are some problems connected to such issues as emailing.

2.4 Multi-Agent Systems

Nevertheless, in multi-agent systems, the way to solve this problem is to divide these tasks between specialized agents who will cope with particular task fields. Thus, there are agents responsible for dealing with calendar issues and scheduling events, an agent able to write and send e-mail messages, an agent providing information on the weather, and an agent-memory who stores the user's preferences.

The function of the supervisory agent is the function of a controller within the multi-agent system. He accepts user's requests, analyzes them, and sends the information received to the appropriate specialist-agent. Besides increased modularity, when separate modules may be created and tested independently, we also acquire more reliability due to the fact that each agent receives information concerning his field only.

2.5 LangGraph and Tool Integration

LangGraph is a tool that was used for creating graph based AI workflows. In the current model, supervisor and each of the various specialist agents will be depicted as a node of a graph, and conditional The edges are used to choose from them the one which receives the request from the user.

The agents can perform real actions in the system via the use of tools. More than simply produce text answers. These tools will be included within the following set: the framework: Gmail, Calendar, web search, weather, memory storage, desktop file access. The agent then executes a tool to retrieve the requested information, retrieves the information from the tool, and returns it to the user. Produces the final response the results.

2.6 Model Context Protocol

Model Context Protocol (MCP) is a way to integrate AI frameworks to external components. Tools, services, and data sources. By giving a standardised communication interface, MCP works with external services: Different capabilities can be connected to the system. through independent servers.

In this project, MCP is used to improve the assistant's capabilities by adding to it. Incorporation of services like Gmail services, desktop file handling. Through this The framework is very extensible and modular, which means that it can be easily extended and modified. Can be easily expanded for additional services in the future.

2.7 Memory and Voice Interaction

How efficient and effective the assistant is will be possible due to the memory. Of recalling interaction from previous time periods. The assistant uses short-term memory to make sure there is continuity of conversation in one conversation, The system can retain user preferences and key information in long-term memory. future interactions.

Voice interaction makes the system more user friendly. The user’s voice control. will be translated to text using speech recognition. When processed, the output of the assistant can also be voiced via a text-to-speech function.

2.8 Requirements Analysis and Design Objectives

2.8.1 Functional requirements

Table 2.1: Functional requirements of the proposed assistant

Functional re-quirements	Functional Requirement	Implemented Through
User Interaction	Accept natural-language user re-quests from a text interface.	CLI main loop and Streamlit UI
Voice Interac-tion	Accept microphone input and con-vert speech to text.	voice.py microphone capture + Whisper transcription
Supervisor-Based Routing	Route user intents to specialist agents.	Supervisor agent + Lang-Graph conditional edges
Calendar Man-agement	Create, search, reschedule, and delete calendar events.	Calendar agent and Google Calendar tools
Weather Re-trieval	Retrieve current weather for a loca-tion.	Weather agent and Open-Weather tool
Web Search	Search the web for current informa-tion.	Search agent and DuckDuckGo-based search tool
Email Automa-tion	Draft and send multi-recipient email messages.	Email agent and SMTP dis-patch tool
Desktop File Ac-cess	Read and write selected desktop text files.	Desktop tools with path re-strictions

2.8.2 Non-Functional Requirements

Non-functional requirements define the quality attributes of the system.

2.9 Design Objectives

The main design The following are the design goals of the project:

1. To create a modular multi-agent architecture where each agent handles a specific domain-related task.

Table 2.2: Non-functional requirements and design responses

Quality attribute	Expectation	Design Response
Maintainability	Capabilities should be independently modifiable.	Specialist agents and tool modules are separated by file and function.
Reliability	Tool failures should not crash the application loop.	Most tools return informative error strings; CLI wraps invocation in exception handling.
Extensibility	New domains should be addable with minimal disruption.	Supervisor route choices and graph nodes can be expanded.
Usability	The assistant should support text and voice interaction.	Streamlit UI, CLI mode selection, microphone input, and TTS response path.
Statefulness	Conversations should maintain turn continuity.	MemorySaver checkpoints indexed by thread ID.
Personalisation	Useful long-lived preferences should be retrievable.	SQLite FTS5 store accessed via <code>save_memory/search_memory</code> .
Safety	Consequential email sending should be controlled.	Dedicated email workflow pauses before approval and supports feedback loops.
Portability	Configuration should not be hard-coded in code where avoidable.	Environment variables and settings module control API keys and SMTP configuration.

2. To build a supervisor agent that can identify user intent and route the request to the appropriate specialist agent.
3. To integrate the Model Context Protocol (MCP) for connecting the system with external tools and services.
4. To reduce context pollution by providing specialist agents only the required information and tool usage instructions.
5. To automate email safely using a human-in-the-loop approval mechanism.
6. To provide short-term and long-term memory for maintaining context and personalization.
7. To offer communication facilities through both text and voice interaction.
8. To design the framework in a way that makes future addition of tools, agents, and services easier.

2.10 System Constraints

The system is built as a one user prototype system. It needs to be able to reach external APIs and services like Large Language Models, Gmail, Google Calendar, weather APIs and more. MCP servers. API keys and authentication information, therefore, should be supplied for Ensuring proper operation of the system. It is also assumed that there is a trusted local environment that is available in the system. Enterprise The advanced authentication and multi-user support are not available at this stage.

The current version mainly focuses on demonstrating the working of a A framework for modular multi-agent assistant framework with tool integration, memory support and user. Interactions are capabilities that are present but can be accomplished in subsequent versions of the architecture.

2.11 System Architecture

2.12 Overview

The proposed system uses a modular multi-agent structure based on the Model Context Protocol. Its main goal is to break down the responsibilities of an AI assistant into separate components, allowing each task to be handled by a specialized agent[3, 4, 12]. Instead of relying on a single chatbot to manage all user requests, the framework employs a supervisor agent. This agent identifies the user’s intent and directs the request to the appropriate specialist agent[3, 4].

The architecture is intended to support real-world productivity tasks such as email automation, calendar management, weather retrieval, web search, memory handling, desktop file access, Gmail retrieval, and voice interaction[5, 6, 11]. Each functional area is built as an independent module, which improves scalability, reliability, and maintainability.

2.13 Architectural Layers

The system is made up of layers, and these layers fulfill certain functions within the system. workflow.

2.13.1 User Interface Layer

The supervisor layer is in charge of intent classification and task routing. Each user action takes the following steps: goes through the supervisor agent. Supervisor reviews the request and decides Who is to be responsible for it as an expert agent? On this calendar there are requests that go to the There are two different agents: the calendar agent for calendar-based requests and the email agent for e-mail-based requests.

2.13.2 Voice Interaction Layer

The voice layer facilitates speech-based communication. It captures sound from the microphone, converts spoken words into text, and sends the transcribed text to the multi-agent

workflow. After the assistant generates a response, the system can convert the text response into speech using text-to-speech technology. This makes the assistant more natural and convenient to use.

2.13.3 Supervisor Layer

There are specific agents specializing in different types of tasks in the specialist agent layer. Each agent has their own set of instructions and apparatus. The principle specialist agents are:

2.13.4 Specialist Agent Layer

The specialist agent layer includes dedicated agents for various task domains. Each agent has its own instructions and tools. The main specialist agents are:

- Calendar Agent
- Email Agent
- Weather Agent
- Web Search Agent
- General Assistant Agent
- Memory Agent
- Desktop File Agent

This configuration improves the accuracy of tasks for each agent as they only work on what they specialize in. domain.

2.13.5 Tool Integration Layer

The tool integration layer can be used to connect the agents to external services and APIs. These tools enable the assistant to carry out tasks other than just simple text generation. Examples include While not everything on this list is necessarily a function of the device, such things as sending emails, getting the weather, searching the web, and accessing Google Calendar all are. The ability to read out computers, and store memory.

2.13.6 MCP Extension Layer

The Model Context Protocol layer is a standard method to connect external capability servers with the assistant. MCP enables access to services like Gmail retrieval and desktop services, among others. file access can be added as separate modules. This then gives the architecture a degree of extensibility, Capable of integration of new tools without re-design of the whole system.

2.13.7 Memory and Storage Layer

Short or long-term memory is stored in the memory layer. Short-term memory Provides the context to the conversation in a given session and long-term memory stores user For future use, preferences, facts and important information. This enables the assistant to Make more context-specific and personalized responses[7, 18].

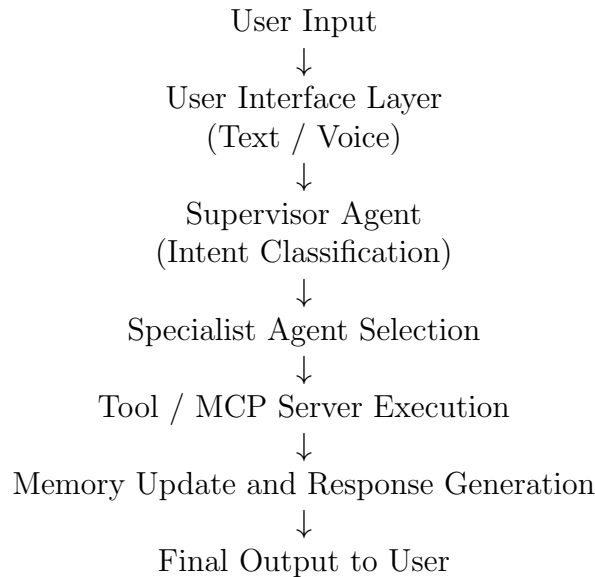
User issues a request via text or voice then the workflow begins. If voice input is used, the voice layer turns speech into text. The request is made to the supervisor agent.

The supervisor agent is then able to determine the user's purpose and pass it on to the most appropriate other agent. appropriate specialist agent. The agent chosen evaluates the task and decides if a tool is needed. When needed, to do an external action, the agent calls the external tool or MCP server.

When the agent receives the result of the tool, he(or she) prepares a final response, relays it to the user interface. In this case, if the user responds, "I need to book a meeting tomorrow at 5 PM", the supervisor will execute the command, "Book a meeting 5 PM tomorrow". Sends the request to the calendar agent. The calendar agent processes the date and time, Calls the calendar tool, makes the event and sends the user a confirmation message.

2.14 Architecture Flow

The overall system flow can be depicted as shown below:



2.15 Model Context Protocol

The Model Context Protocol (MCP) is used to implement a simple and standard protocol To read and write data from and to model contexts. It is undertaken in an important way Creating more extensible and modular framework. Traditional systems have to have custom integration logic for each external service.

This Complicates and restricts future growth of the system. However, MCP is a solution to this issue since: Bringing capabilities exposed from outside to a common

interface. Services like accessing files on the desktop and retrieving Gmail are incorporated in this project using MCP.

These services are independent capability servers that are accessible by The assistant to be used when needed. This design will achieve the following: This will allow the system to be modular, reusable, easier to expand.

2.16 Benefits of the Proposed Architecture

The introduced design offers the following advantages:

1. **Modularity:** Each agent and each tool is created separately and independently from other agents and tools.
2. **Scalability:** New tools and agents can be added without modifying the whole system architecture.
3. **Reliability:** Specialist agents reduce tool confusion and improve task-specific performance.
4. **Maintainability:** Error handling and updates can be managed on a per-module basis.
5. **Extensibility:** MCP provides external services that can be plugged in through well-defined capability server interfaces.
6. **Security:** For sensitive operations such as sending emails, human review can be required before final execution.
7. **Personalisation:** The assistant can remember important user information and user preferences.

2.17 Summary

The overall system architecture is a multi-agent architecture with a supervisor. It is an agent and is enhanced with the Model Context Protocol. The building is split into two parts. into various levels – user interaction, supervisor routing, specialist agents, tools The areas of integration, memory management and access to external services.

The application is more reliable and scalable if it's broken into separate layers, and maintainable. Multi-agent coordination with extensibility based on MCP is proposed, and it is expected to lay a solid foundation on which to develop intelligent AI assistants. Able to engage in practical productivity activities.

Chapter 3

Methodology

3.1 Introduction

The requirement analysis is one of the very important factors in designing any software system. It defines the basic functions, actions and needs to be met by the software. In the present project, requirement analysis will be used to design an AI with multi-agents. The Model Context Protocol was used to implement a system. The system is aimed at supporting the assistant for conducting real-life productivity tasks, Including email management, calendar management, web search and weather information. retrieval, memory management, desktop file access and voice recognition.

The main requirement of the assistant is that it should have a modular and expandable design. The system does not bundle all the features within a single monolithic chatbot, but rather integrates them into distinct modules. Rather than having everything in one large chatbot, they are packaged into separate modules.

employs a variety of specific agents to various productive activities.

Supervisor Agent → Specialist Agent → Tool Execution → Final Response

The specialist agents include the calendar agent, email agent, weather agent, search agent, and memory/general agent.

3.2 Supervisor Agent

The supervisor agent is the central decision-making component of the system. Every user query first reaches the supervisor agent. Its role is to understand the intent of the user and decide which specialist agent should handle the request.

For example, if the user asks, “Schedule a meeting tomorrow at 3 PM,” the supervisor routes the request to the calendar agent. If the user asks, “What is the weather in Delhi?”, the request is routed to the weather agent. Similarly, email-related queries are routed to the email agent, search-related queries to the search agent, and memory or file-related queries to the general agent.

The supervisor agent improves the efficiency of the system by ensuring that only the relevant agent and tools are activated for a particular task. This avoids unnecessary processing and keeps the workflow organized. ...necessary processing and keeps the workflow organized.

The routing decision may contain values such as:

- calendar

- email
- weather
- search
- general
- FINISH

The FINISH option is used when no specialist agent is required or when the response can be completed directly.

3.3 Calendar Agent

The calendar agent is responsible for handling calendar-related tasks. It interacts with calendar tools connected to the Google Calendar API. The main functions of the calendar agent include creating events, checking availability, searching existing events, deleting events, and rescheduling meetings.

The Calendar Agent is useful for productivity-related tasks that require calendar management. It lets the user schedule, search, delete and create calendar events with natural language commands.

For instance, a user may type in “book a meeting for Rahul at 10am tomorrow.” The agent transforms the natural language request to a structured calendar operation and She plans the meeting for the appropriate time [7].

The main tools used by the Calendar Agent are:

- Schedule meeting
- Check availability
- Search events
- Delete event
- Reschedule event

When all required details are available, the agent directly calls the scheduling tool to create the meeting. For deleting or rescheduling an event, the agent first searches the calendar and retrieves the event ID. This prevents accidental changes to the wrong calendar entry.

3.4 Email Agent

The Email Agent is responsible for email-related tasks such as drafting, reviewing, regenerating, and sending emails. Since sending an email is a sensitive operation, the system follows a human-in-the-loop workflow.

In this workflow, the email is not sent automatically. First, the agent drafts the email and presents it to the user for review. The email is sent only after the user gives explicit approval [?].

Example requests handled by the Email Agent include:

- Write an email to my professor.
- Write an email to these recipients.
- Use a formal tone for the email.
- Send the email only after I approve.

The major tool used by the Email Agent is:

- Send multiple emails

The Email Agent can also use memory tools to remember user preferences, such as the preferred tone, writing style, or name format. This design improves safety and user control by ensuring that irreversible actions are not performed without confirmation.

3.5 Weather Agent

The weather agent handles weather-related queries. It is connected to a weather retrieval tool that fetches live weather information from an external API. The agent is instructed not to guess weather information from the LLM’s training data[5, 6].

Typical user queries include:

- What is the weather in Delhi?
- Tell me the temperature in Mumbai.
- Will it rain in Noida today?

The main tool used by the weather agent is:

- `get_weather`

The weather agent demonstrates the importance of tool-grounded responses. Since weather changes continuously, relying only on model knowledge would produce outdated or incorrect results. Therefore, the agent always retrieves live information before generating the final answer.

3.6 Search Agent

The search agent is responsible for retrieving current information from the web. It is used when the user asks for recent updates, research topics, coding resources, product comparisons, or information that may not be available in the model’s internal knowledge[7, 6].

Example queries include:

- Search latest research on multi-agent systems.
- Find information about Model Context Protocol.
- Compare LangGraph and AutoGen.

- Search recent AI tools for productivity.

The main tool used by the search agent is:

- `web_search`

The search agent improves the assistant’s ability to answer real-time and research-oriented questions. It reduces hallucination by grounding the response in external search results.

3.7 Memory and General Agent

The memory/general agent handles general conversation, memory operations, and desktop file interaction. It is responsible for storing useful user information and retrieving it when needed. It also acts as a fallback agent for queries that do not belong to calendar, email, weather, or [7, 18].

The memory system is divided into two parts:

- **Short-term memory:** Maintains current conversation context.
- **Long-term memory:** Stores persistent facts and preferences.

The main tools used by the memory/general agent are:

- `save_memory`
- `search_memory`
- `list_desktop_files`
- `read_desktop_file`
- `write_desktop_file`

Example memory-related queries include:

- Remember that I prefer formal emails.
- What email style do I prefer?
- Save this note for later.

Example file-related queries include:

- List files on my desktop.
- Read this text file.
- Write this content into a file.

This agent improves personalization because the assistant can remember useful user preferences and reuse them in future conversations.

3.8 Tool Integration

Tools are external functions that allow agents to perform real-world actions. Without tools, the assistant would only generate text. Tool integration enables the system to send emails, create calendar events, retrieve weather data, search the web, store memory, and interact with local files[5, 6, 10].

Each tool is connected to the relevant specialist agent. This ensures that tools are not exposed unnecessarily to every agent.

The tool execution flow is:



This flow ensures that every action is handled through a controlled and traceable process.

3.9 Agent and Tool Mapping

Table 3.1: Agent and Tool Mapping

Agent	Responsibility	Tools
Supervisor Agent	Routes user request to correct agent	Routing logic
Calendar Agent	Manages events and schedules	Calendar tools
Email Agent	Drafts and sends emails after approval	Gmail/email tools
Weather Agent	Retrieves live weather information	Weather API tool
Search Agent	Performs web search and research	Web search tool
General/Memory Agent	Handles memory and file operations	Memory and desktop tools

Chapter 4

Implementation Details

4.1 Introduction

This chapter explains how the Multi-Agent Framework Based on Model Context Protocol is set up. The design of the framework is a modular AI assistant, with each of the crucial functions, Organized in modular parts such as agents, tools, workflow, memory and user interface, voice pipeline, and MCP servers. This division renders the system more comprehensible, Testable, maintainable and extensible.

The implementation is done in a layered manner. Users can interact with the system via a website through Streamlit, or by using a developed API. The system can be interacted by the user through a website via Streamlit or through a built API. Using a command line interface. The user request is further sent to the LangGraph workflow where the supervisor agent Identifies the intent and routes to appropriate specialist agents. The Then selected agent will get various tools, like calendar tools, Gmail tools, weather, etc. Memory tools, External MCP-based capability servers, tools, or Web search tools.

4.2 Technology Stack

This is a system that is developed in python code and various libraries are used to orchestrate AI, build interfaces, and integrate external services. This technology stack Supports: Modular agent design, multimodal interaction, the use of tools, and memory management.

The choice of a stack is based on quick prototyping and further improvements. LangGraph offers graph-based workflow control flow, LangChain delivers tool integration support. Streamlit provides an interactive UI, and MCP facilitates linking of external Provision services in a stereotyped manner.

4.3 Environment Configuration

This is a system that is developed in python code and various libraries are used to orchestrate AI, build interfaces, and integrate external services. This technology stack Supports: Modular agent design, multimodal interaction, the use of tools, and memory management.

The choice of a stack is based on quick prototyping and further improvements. LangGraph offers graph-based workflow control flow, LangChain delivers tool integration support. Streamlit provides an interactive UI, and MCP facilitates linking of external Provision services in a stereotyped manner.

Table 4.1: Technology Stack Used in the Proposed Framework

Component	Technology / Library	Purpose
Programming Language	Python	Core implementation
Agent Orchestration	LangGraph	Multi-agent workflow and routing
Prompt and Tool Binding	LangChain	Agent prompt templates and tool integration
Language Model	Gemini / OpenAI	Reasoning and response generation
Web Interface	Streamlit	User-friendly graphical interface
Voice Input	Whisper STT	Speech-to-text transcription
Voice Output	OpenAI TTS	Text-to-speech response
Memory Storage	SQLite FTS5	Long-term memory storage
Calendar Service	Google Calendar API	Event creation and management
Email Service	Gmail / SMTP	Email drafting and sending
Web Retrieval	Search API / Requests	Real-time information retrieval
MCP Integration	FastMCP	External capability servers

For security reasons, actual API keys and passwords are not shown in the report. A screenshot of the environment configuration can be added while hiding sensitive values.

For security reasons, actual API keys and passwords should be concealed before adding the screenshot to the report.

4.4 LLM Configuration

All agents use the same LLM configuration, which is stored in a separate utility file. This prevents repeated model initialization and makes it easier to switch the model if required [?, ?].

A low temperature value is used to obtain more consistent responses and reliable tool

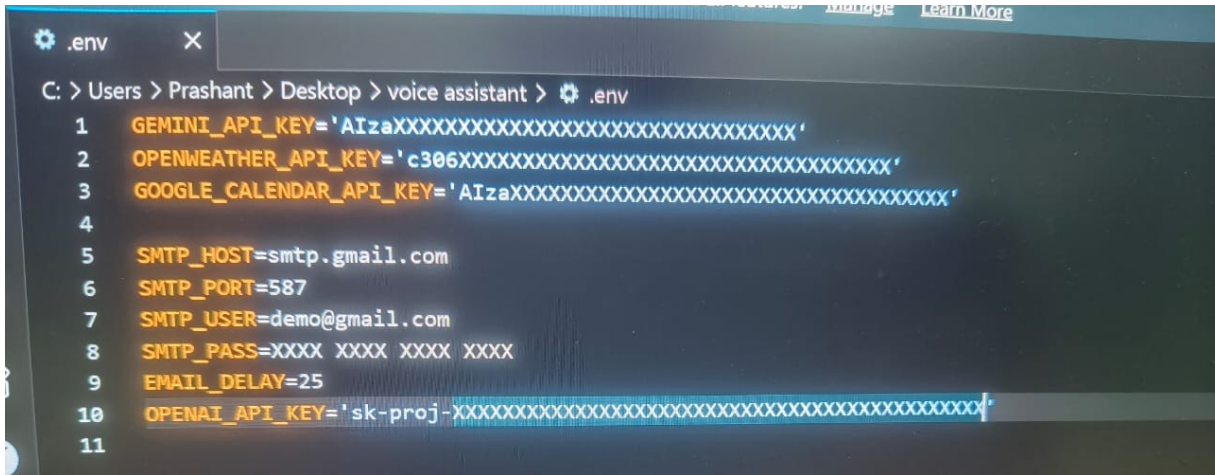


Figure 4.1: Environment configuration using

calls. This is important for routing and tool execution, because unpredictable model behavior may lead to selecting the wrong agent or passing incorrect arguments to a tool.

Chapter 5

Testing, Results, and Discussion

5.1 Introduction

This chapter presents the testing strategy, validation process, observations, screenshots, and result analysis for the development of a Multi-Agent Framework based on the Model Context Protocol (MCP). Testing is important for this project because the system performs more than simple text generation [?, ?].

The framework performs real-world productivity tasks such as processing user requests, using external tools, accessing memory, retrieving Gmail messages, interacting with files, checking weather information, managing calendar tasks, and supporting voice interaction.

The main goal of testing is to verify whether the framework can correctly understand user requests, route them to the appropriate agent, use tools safely, maintain conversation context, and generate suitable responses. Since the system follows a modular multi-agent architecture, each component is tested individually as well as within the complete workflow.

The main goal of testing is to confirm whether the framework can accurately understand user intent, route requests to the right specialist agent, execute tools safely, maintain conversation context, and generate meaningful responses. Since the system operates with a modular multi-agent architecture, each component is tested individually and as part of the complete workflow.

5.2 Testing Strategy

The testing strategy is divided into three main stages:

1. **Unit Testing:** In unit testing, individual tools and functions are tested separately. For example, desktop file retrieval, weather search, memory listing, memory saving, and email tool functions can be tested independently.
2. **Integration Testing:** Integration testing verifies the interaction between agents and their respective tools. For example, the weather agent should call the weather tool, the calendar agent should use calendar tools, and the email agent should create a draft before sending an email.
3. **End-to-End Testing:** End-to-end testing checks the complete flow of a user request. The user query is first received by the supervisor agent, then routed to a specialist agent, and finally processed using the required tool or response generation method.

This testing structure ensures that both the internal logic and the overall system behavior are properly validated.

5.3 Supervisor Routing Validation

The results indicate that common domain may be recognized by the supervisor agent. Forward to the appropriate agent for productivity-related questions This confirms that Using intent classification and routing, the mechanism performs well for tasks like calendar, management, emails automation, weather data, and memory operations.

Table 5.1: Supervisor Routing Validation

User Query	Expected Agent
Schedule a meeting tomorrow at 3 PM	Calendar Agent
What is the weather in Delhi?	Weather Agent
Draft an email to my professor	Email Agent
Search the web for latest AI agent frame-works	Search Agent
Remember that I prefer formal emails	General/Memory Agent
List files on my desktop	General Agent

The results show that the supervisor agent can recognize the domain of common productivity-related queries and forward them to the correct agent. This confirms that the intent classification and routing mechanism works effectively for tasks such as calendar management, email automation, weather checking, and memory operations.

5.4 Agent-Level Testing

5.4.1 Calendar Agent Testing

The Calendar Agent was tested for event creation, event search, availability checking, event cancellation, and event rescheduling. The purpose of testing was to verify whether the agent can correctly understand calendar-related requests and select the appropriate calendar tool. Some important test scenarios were:

Some important test scenarios were:

- Creating a new meeting.
- Checking availability on a particular date.
- Searching for an existing event.
- Rescheduling an event after retrieving the event ID.
- Deleting an event after confirming the correct event.

The results show that the Calendar Agent can handle both single-step and multi-step calendar tasks. Scheduling is performed directly when all required information is available, while deletion and rescheduling first require an event search to avoid modifying the wrong calendar entry.

5.4.2 Email Agent Testing

The Email Agent was tested for email drafting, review, regeneration, approval, and sending. Since sending an email is a sensitive task, the framework uses a human-in-the-loop approval workflow [?, ?].

The Email Agent follows these steps:

1. Understand the user’s email request.
2. Identify the recipient, subject, and purpose of the message.
3. Draft the email.
4. Present the draft to user for approval.
5. Regenerate the email if the user requests changes.
6. Only send the email when user says "yes". The test results indicate that the Email Agent does not send emails when there is no user confirmation. This approval step adds to the safety, minimizes accidental actions and provides authority for the prior step. the user better control over sensitive communication tasks.

The testing results show that the Email Agent does not send emails without user confirmation. This approval step improves safety, reduces accidental actions, and gives the user better control over sensitive communication tasks.

5.4.3 Weather Agent Testing

The Weather Agent was tested using various location-based weather queries. The main objective of this testing was to ensure that the agent does not generate weather information from the language model’s internal knowledge, but instead uses the weather tool to retrieve updated information.

Example queries included:

- What is the weather in Delhi today?
- Check the weather in Mumbai.
- Tell me the temperature in Bengaluru.
- Is it raining in Gurgaon?
- What is the weather in Delhi?
- Tell me the current temperature in Mumbai.
- Will it rain in Noida today?

The results show that the Weather Agent correctly identifies weather-related requests and calls the weather retrieval tool to obtain updated information. The agent successfully identifies the location mentioned in the query and retrieves weather data from an external source. This ensures that the response is based on real-time external data rather than outdated training information, thereby improving the accuracy and reliability of weather responses.

5.4.4 Web Search Agent Testing

The Web Search Agent was tested for retrieving current information, answering research-related queries, assisting with coding-related searches, and supporting product comparison tasks. Since large language models may not always contain the latest information, the Web Search Agent provides real-time information retrieval support.

Example queries included:

- Checking the latest changes in LangGraph.
- Identifying recent research papers in the field of multi-agent systems.
- Comparing MCP-based tool integration with API-based tool integration.

The Web Search Agent fetches relevant information from the web and generates a summarized response for the user. This helps the assistant answer queries that require fresh, updated, or external knowledge.

5.4.5 General and Memory Agent Testing

The General Agent was tested for memory storage, memory retrieval, and desktop file interaction. It is highly reliable. It is very reliable. In the memory storage test, the General Agent was tested to see how many items it could retain in its memory, while it was tested on memory retrieval in the second test and the desktop file in the third test.

interaction. This testing was designed to test and confirm that the system could store user Identify and remember their preferences and use them when interacting in the future.

This could be the case if the user mentions that they prefer formal email writing. Records this preference in long term memory. Later, when the user asks : What email style: When the assistant issues the command do I prefer he or she tries to recall

the previous preference and answers appropriately. The memory test validates the systems capacity to personalize it with the help of: A continuous storage and retrieval system. It was also found that the desktop file operation, which included listing, could be tested.

5.5 MCP Server Validation

Testing of the connection between the assistant and external capability servers was performed using the Model Context Protocol (MCP). The project includes MCP-based support for Gmail retrieval and desktop file access.

The following Gmail MCP server functions were validated:

- Searching Gmail messages using a query.
- Retrieving sender information.
- Retrieving subject lines.
- Extracting message snippets.
- Returning structured results to the assistant.

The Desktop MCP server was tested for the following functions:

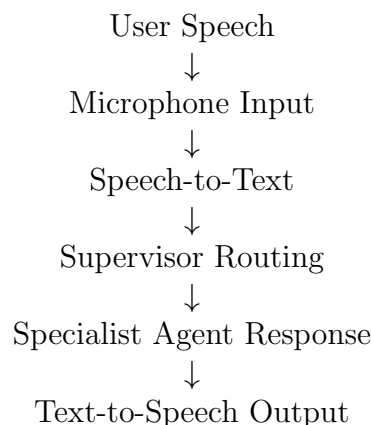
- Listing files from the desktop.
- Viewing files available on the desktop.
- Sending file contents to the assistant.
- Exposing tools in a modular format.

The validation shows that MCP separates external service logic from the main assistant workflow. This improves modularity and makes it easier to extend the framework with additional capability servers in the future.

5.6 Voice Interaction Testing

The voice pipeline was put to the test to see how good a microphone is in terms of input, how well someone can talk to the computer, and how well the computer can listen or how accurately it converts speech to text. and text-to-speech output. The user can utter a command, and the system transforms it into. Uploads Text to the multi- agent workflow and reads it back.

The usability, responsiveness and flow of interaction of the Streamlit interface has been tested. The interface has two large tabs:



5.7 User Interface Validation

The usability, responsiveness and flow of interaction of the Streamlit interface has been tested. The interface has two large tabs:

1. Agent Tab
2. Developer Console Tab

The Agent Tab enables the user to do voice or text communication with the assistant. text. It features an uncluttered dark user interface, an animated assistant orb, and a ‘Talk To Your contact.It has a simple dark color user interface, an animated assistant orb and ‘Talk To Your contact;, an animated assistant orb. Agent” button.

Options for debugging and session information are available in the Developer Console Tab. It displays The current thread ID, recent actions and a memory reset button. These features provide It is not just user-friendly, but also developer-friendly, with both aspects of the interaction.

5.8 Screenshots and Visual Observations

Take screenshots and make visual observations. The screenshots that are part of the report depict the working interface of the project and show the responses of the system when interacted.

5.8.1 Main Agent Interface

Figure 5.1 shows the assistant in an idle state with an orb animation and voice interaction button. The text “Ready when you are” indicates that the assistant is ready to accept user input. This interface is designed to present the system as an interactive AI assistant

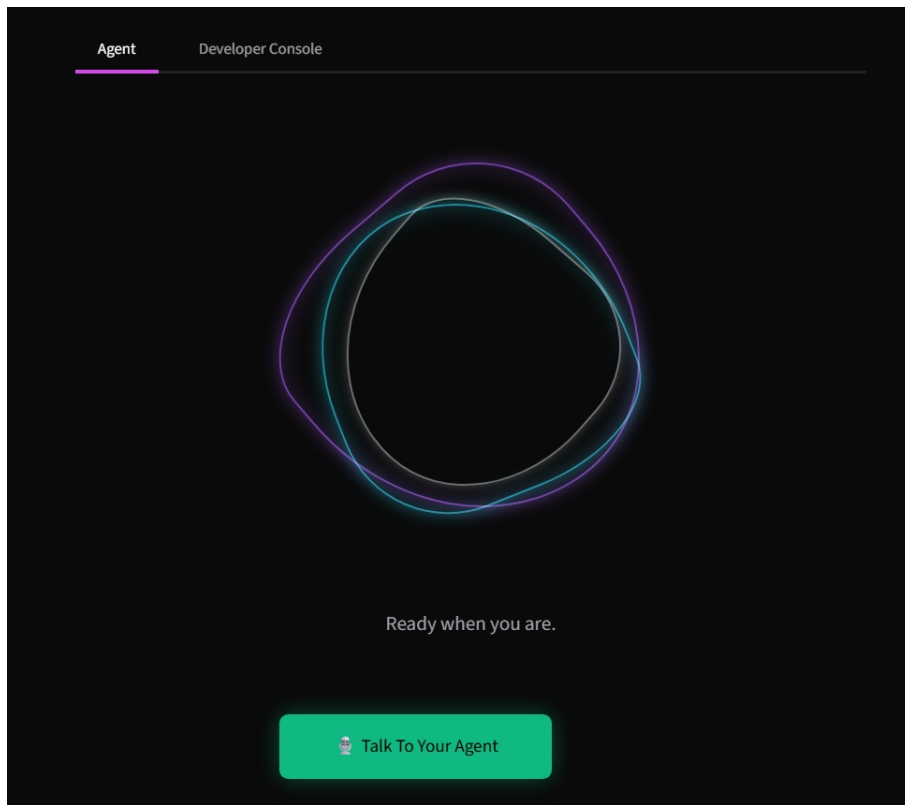


Figure 5.1: Main Agent Interface

rather than just a chatbot.

5.8.2 Developer Console Interface

Figure 5.2 shows the developer console with the active thread ID and memory reset option. The thread ID is useful for debugging LangGraph memory sessions because every conversation session is linked with a unique ID. The reset memory button allows the user to clear the current memory and start a new session.

5.9 Limitations Observed During Testing

Personalization is improved through the memory system, which enhances the overall user experience. Short-term memory is used to maintain the flow of the current conversation,

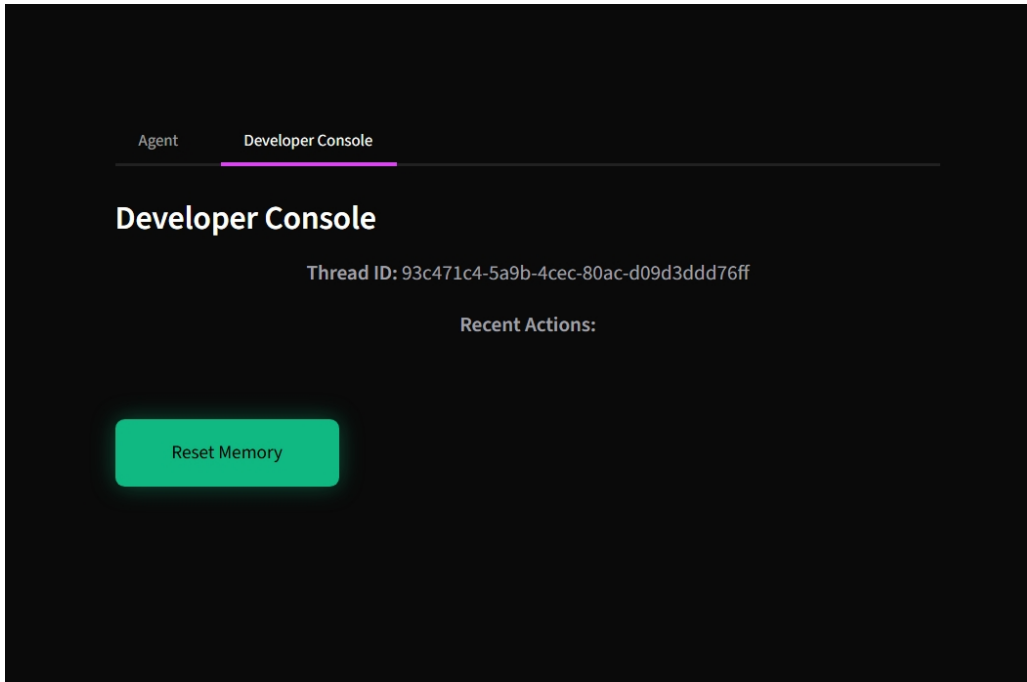


Figure 5.2: Developer Console Interface

while long-term memory stores useful facts and user preferences.

However, the memory system can be improved further by adding semantic search and conflict resolution. These features would help the assistant retrieve more accurate information and handle inconsistent stored memories.

The system is also more accessible and practical because of voice interaction and the Streamlit user interface. The interface is easy of use, easy to read and can be demonstrated. The quality of the voice interaction is dependent on the quality of the microphone, however, and background noise.

The following restrictions were noted during testing:

1. Voice input may not be accurate in noisy environments.
2. The ability to retrieve information from long-term memory depends on the quality of stored information and the precision of user queries.
3. The use of stricter state-level validation can improve the email approval workflow.
4. File access from the desktop should be handled with more robust sandboxing mechanisms for production use.
5. External APIs such as Gmail, Google Calendar, weather services, and web search require valid authentication credentials and a stable internet connection.
6. The existing system is designed for a single-user environment and is not optimized for multiple users.

5.10 Summary

This chapter presented the testing strategy, validation process, observations, screenshots, and result analysis of the proposed multi-agent framework. The system was tested at

unit, integration, and end-to-end levels.

The results show that the framework can route user requests, call suitable tools, maintain memory, support voice interaction, retrieve external information, and provide a usable interface.

The testing confirms that supervisor-driven routing, specialist agents, tool integration, MCP servers, memory systems, and human-in-the-loop workflows together provide a strong foundation for a practical AI assistant framework.

Personalization is enhanced through the memory system, which improves the overall user experience. Short-term memory is used to maintain conversation continuity, while long-term memory stores useful facts and user preferences.

However, the memory system can be further improved by adding semantic search and conflict resolution. These features would help the assistant retrieve more accurate information and handle inconsistent memories more effectively.

The system is also more accessible and practical due to voice interaction and the Streamlit-based user interface. The interface is simple, clear, and suitable for demonstration purposes. However, voice accuracy depends on microphone quality and background noise.

The following limitations were observed during testing:

1. Voice input may not be very accurate in noisy environments.
2. Long-term memory retrieval depends on the quality of stored information and the clarity of user queries.
3. Email approval workflow can be improved further through stricter state-level validation.
4. Desktop file access should be restricted using stronger sandboxing for production use.
5. External APIs such as Gmail, Calendar, weather, and web search require valid credentials and a stable internet connection.
6. The current framework is designed for a single-user environment and is not optimized for multiple users.

5.11 Summary

This chapter introduced the testing approach, validation procedure, observations, screenshots, The proposed multi-agent framework is simulated using simulation tools and the results analyzed. The system was evaluated on a test set of at unit, integration and end-to-end levels.

The framework is able to route a request made by the user, call appropriate functions and send the results back. gathers information from outside and supports voice interaction, maintain memory, and use tools. offers a usable interface.

The validation verifies that supervisor driven routing, specialist agents, tools integration, MCP servers, memory systems and human in the loop workflows form a complete system for providing. Excellent for creating a robust AI assistant system.

Through the memory system, the overall Personalisation is enhanced. user experience. Long-term memory is employed to recall information from the past, long-term memory holds information and preferences that are useful.

Semantic search can be incorporated into the memory system to enhance it further. and conflict resolution. These features would aid the assistant in getting more precise. process information and remember inconsistently better.

Voice interaction and the systems accessibility and usability are also enhanced: Streamlit-based user interface. The interface is easy, clear and can be presented. Voice accuracy is subject to the audio quality of voice input devices and background noise. noise.

The following restrictions were noted during testing:

1. The accuracy of voice input may decrease in noisy environments.
2. Long-term memory retrieval depends on the quality of stored information and the precision of user queries.
3. Further state-level validation can improve the email approval workflow.
4. Desktop access should be restricted using stronger sandboxing mechanisms for production use.
5. External APIs such as Gmail, Google Calendar, weather services, and web search require valid authentication credentials and an active internet connection.
6. The current framework is optimized for a single-user environment and is not designed for multi-user usage.

Chapter 6

Limitations and Future Scope

6.1 Introduction

The Multi-Agent Framework based on Model Context Protocol (MCP) is effective in terms about modular agent orchestration, tool integration, memory management, voice interaction and MCP-based extensibility. But there are some drawbacks in the present implementation.

The majority of these constraints are linked to scalability, security, memory performance, reliance on external APIs, voice recognition, and production-ready. This chapter discusses these constraints, as well as future improvements to make the framework more reliable, Secure, scalable and practical application.

6.2 Current Limitations

6.2.1 Single-User Design

The current system is a single user AI assistant system. It features multiple support. It has no multiple support. User accounts, role-based access control, or individual user profiles.

All memory, configuration and tools are targeted towards one trusted user environment. This limitation is relevant because in real usage, multiple components could be required for use and separation of the components may be necessary. Different user credentials, permissions and memory.

6.2.2 Limited Security Controls

This framework is compatible with sensitive services like Gmail, Google Calendar, desktop, etc. files. While email sending must be approved by human, the system lacks the relevant advanced security features like email authentication, permission management, action auditing etc. or strict sandboxing.

File access and calendar operations should also be more robustly validated and have better permission controls for production use.

6.2.3 Dependency on External APIs

The system relies on external APIs and services like providers of LLM models, Gmail, Google APIs for Calendar, the Weather forecast, Web search, Whisper and Text to speech

services. When the services become unavailable, reach rate limits or when they are mis-configured, There may be a problem with the corresponding agent.

This dependency also calls for greater operating cost and requires A reliable wireless connection.

6.2.4 Limited Long-Term Memory Intelligence

The long-term memory system stores useful facts, preferences, using storage in the form of a sqlite database. and project details. But the present method of memory retrieval has some restriction, that is, from simple to complex semantic memory systems.

If a user presents a query in a different manner to that stored in the memory, retrieval may not be true at all times. The system is also not currently equipped with memory conflict: It may be the resolution, memory ranking or automatic memory cleanup.

6.2.5 Voice Recognition Limitations

The quality of the accuracy of the voice pipeline is dependent on the quality of the microphone, background noise, and what you are announcing. Available: pronunciation, and network availability. The module of speech to text works well in noisy conditions. The module speech to text works well for noisy environments. may misinterpret speech which can impact routing and response.

The current system is also lacking of sophisticated features to allow speaker identification or wake-word detection.

6.2.6 Limited Evaluation Metrics

Current testing for routing, tools, memory, end to end workflow execution. However, the assessment is primarily functional and not comprehensive, such as a large-scale. quantitative metrics.

The accuracy rate of routing, the success rate of calling the tools, the latency, cost and the user satisfaction score can be part of the evaluation for the future. These would give a more real understanding of the performance of the system.

6.2.7 Basic MCP Integration

This project illustrates simple integration of Gmail, Yahoo, other external services with the help of MCP. Accessing or retrieving files on desktop or from a retrieval. The MCP layer is still in its infancy, however.

The current implementation does not support advanced authentication, dynamic Whereas, there are many ways to create a discovery server, load tools based on permissions, or load tools dynamically. Future versions can enhance the flexibility and production readiness of MCP use.

6.3 Future Scope

- **Multi-user support:** can be expanded for multiple users with Separate profiles, memory, Gmail, calendar and permissions. This will make it Perfect for groups, classes, and groups.

- **Stronger security:** Future versions can include authentication, and role-based access. encrypted credentials, audit logs and control. The user should be involved with sensitive actions. confirmation before execution.
- **Advanced memory system:** The memory system can be improved using vector You can use these databases like FAISS, ChromaDB, or Pinecone. This will assist you to get relevant Memories which are created not just on the basis of keywords but also on the basis of the meaning.
- **Improved voice interaction:** Voice operation can be enhanced by the use of wake-word Detection, noise cancelling and real-time speech recognition. This will make the More user friendly (with hands-free operation).

Chapter 7

Conclusion

The aim of this project was to develop a Multi-Agent Framework based on Model Context Protocol. The goal was to design an AI assistant that is modular, scalable, and easy to use. that can actually do tasks and be productive in real life.

The framework tackles the shortcomings of the conventional monolithic chatbot systems, Context, instructions, and tools in one model to respond to all user requests. Such Context overload, a lack of tool selection, low level of specialization, and can be a failure of systems. Problems with future expansion.

The proposed scheme will be based on multi-agent system based on supervisor architecture. The supervisor agent recognizes the users purpose and forwards it to the most appropriate expert. agent. Each agent has a different responsibility like Calendar Management, etc. There are several advantages to using the Auto -Everything features in your email. The Auto -Everything features in email have several benefits. handling, assistance and general assistance.

This distinction in tasks reduces the systems unreliability, complexity and resistance to maintenance and scalability. This project also includes integration with external services through the Model Context Protocol (MCP). For example, Gmail and file access on Desktops. So, how can you integrate MCP The system can be expanded through external capabilities without having to make significant changes to the main system. workflow.

The other key aspect of the system is the human-in-the-loop email automation. workflow. This process involves composing, revising, and rewording emails as needed. They Are sent only when explicitly consented to by the user. This gives an extra security as well. Always place sensitive tasks on the top of the stack (layer).

The system implements the following LSM memory management: short-term memory: LangGraph checkpointing long-term memory: SQLite type based on. The memory that is stored for long-term use. This aids in the continuity of the conversation and fosters support for the conversation. personalized user experiences. The framework also offers support specifically for interactions in both texts and voice with a Streamlit CLI and GUI.

This enhances accessibility and usability. In the end, the project shows that orchestration with multiple agents and MCP-based tools can be used to develop an application that easily handles an increasing number of agents. Voice interaction, memory management and integration can be integrated together to create a The functional AI assistant framework. The system is not just a text generation system, it's a system that goes beyond it. anticipating user needs, choosing the appropriate agent, employing tools, remembering information, performing real-world actions.

7.1 Project Contribution

This project succeeded in achieving the following outcomes:

1. A modular multi-agent approach for productivity and automation tasks.
2. A supervisor-based routing mechanism to select appropriate specialist agents.
3. Specialized agents for handling calendar management, email automation, weather retrieval, web searching, memory operations, and file operations.
4. Support for Model Context Protocol (MCP) to connect external capability servers.
5. A secure email automation process with human-in-the-loop review.
6. Support for both short-term and long-term memory for effective context management.
7. Text-based and voice-based interaction using Streamlit and command-line interface.
8. An extensible architecture that allows easy addition of new agents and tools.

7.2 Final Outcome

The aim of this project is to create a working prototype of an AI assistant system. It shows the application of using multi-agent systems, and Model Context Protocol. (MCP) to automate real-life productivity tasks.

The system is modular, expandable and can carry out different productivity tasks based on intelligent routing and tools usage. At the present time, the implementation is Created for use in a single user academic setting. It is, however, a solid base for improvement in the future, e.g., multi-users, etc. The advantages include support, greater security, better memory, more MCP servers, and better evaluation. methodologies, tracking capabilities and cloud deployment.

Therefore, the overall goal of the project – to develop a multi-agent framework based on Model Context Protocol has been achieved.

Bibliography

- [1] OpenAI, “GPT-4 Technical Report,” *arXiv preprint arXiv:2303.08774*, 2023.
- [2] Gemini Team, “Gemini: A Family of Highly Capable Multimodal Models,” *arXiv preprint arXiv:2312.11805*, 2023.
- [3] Q. Wu, G. Bansal, J. Zhang, Y. Wu, B. Li, E. Zhu, L. Jiang, X. Zhang, S. Zhang, A. H. Awadallah, R. W. White, D. Burger, and C. Wang, “AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation Framework,” *arXiv preprint arXiv:2308.08155*, 2023.
- [4] G. Li, H. A. K. Hammoud, H. Itani, D. Khizbullin, and B. Ghanem, “CAMEL: Communicative Agents for Mind Exploration of Large Language Model Society,” *arXiv preprint arXiv:2303.17760*, 2023.
- [5] 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.
- [6] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao, “ReAct: Synergizing Reasoning and Acting in Language Models,” *International Conference on Learning Representations*, 2023.
- [7] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Kuttler, M. Lewis, W. Yih, T. Rocktaschel, S. Riedel, and D. Kiela, “Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks,” *Advances in Neural Information Processing Systems*, 2020.
- [8] A. Radford, J. W. Kim, T. Xu, G. Brockman, C. McLeavey, and I. Sutskever, “Robust Speech Recognition via Large-Scale Weak Supervision,” *arXiv preprint arXiv:2212.04356*, 2022.
- [9] LangChain, LangGraph StateGraph Documentation,” Available: <https://reference.langchain.com/python/langgraph/graph/state/StateGraph>. Accessed: May 2026.
- [10] LangChain, LangChain Tools Documentation, Available:<https://docs.langchain.com/oss/python/langchain/tools>. Accessed: May 2026.
- [11] LangChain, LangChain Tool Calling Documentation,” Available:<https://docs.langchain.com/oss/python/langchain/frontend/tool-calling>. Accessed: May 2026.
- [12] Anthropic, Model Context Protocol Documentation,” Available:<https://modelcontextprotocol.io/>. Accessed: May 2026.

- [13] OpenAI, OpenAI Audio API Documentation, Available: <https://platform.openai.com/docs/guides/audio>. Accessed: May 2026.
- [14] Google Developers, Google Calendar API Overview, Available: <https://developers.google.com/calendar/api/guides/overview>. Accessed: May 2026.
- [15] Google Developers, Gmail API Documentation, Available: <https://developers.google.com/gmail/api>. Accessed: May 2026.
- [16] Google AI for Developers, Gemini API Documentation, Available: <https://ai.google.dev/gemini-api/docs>. Accessed: May 2026.
- [17] Streamlit, Streamlit API Reference: `st.set_page_config`, Available: https://docs.streamlit.io/develop/api-reference/configuration/st.set_page_config. Accessed: May 2026.
- [18] SQLite, SQLite FTS5 Extension Documentation, Available: <https://www.sqlite.org/fts5.html>. Accessed: May 2026.
- [19] Pydantic, Pydantic Settings Documentation, Available: <https://docs.pydantic.dev/latest/concepts/>. Accessed: May 2026.
- [20] Python Software Foundation, Python 3 Documentation, Available: <https://docs.python.org/3/>. Accessed: May 2026.

Bibliography