MCP and custom tools enabled Personal Assistant

A major project report submitted in partial fulfillment of the requirement for the award of degree of

Bachelor of Technology

in

Computer Science & Engineering

Submitted by

Meghna Mankotia (221030146)

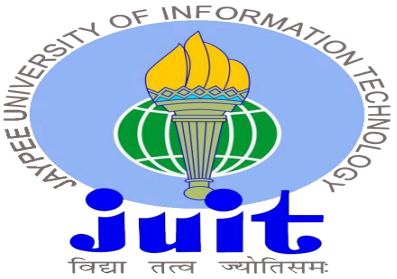
Nikhilesh Sharma (221030003)

Shashvat (221030117)

Yuvraj Saini (221030233)

Under the guidance & supervision of

Prof. Dr. Pradeep Kumar Gupta



Department of Computer Science & Engineering and Information Technology

Jaypee University of Information Technology, Waknaghat, Solan - 173234 (India)

December 2025

Candidate's Declaration

We hereby declare that the work presented in this major project report entitled **'Project Title'**, submitted in partial fulfillment of the requirements for the award of the degree of **Bachelor of Technology** in **Computer Science & Engineering**, in the Department of Computer Science & Engineering and Information Technology, Jaypee University of Information Technology, Waknaghat, is an authentic record of our own work carried out during the period from July 2025 to December 2025 under the supervision of **Supervisor Name**.

We further declare that the matter embodied in this report has not been submitted for the award of any other degree or diploma at any other university or institution.

Name & Sign: Meghna Mankotia

Roll No.: 221030146

Date:

Name & Sign: Nikhilesh Sharma

Roll No.: 221030003

Date:

Name & Sign: Shashvat

Roll No.: 221030117

Date:

Name & Sign: Yuvraj Saini

Roll No.: 221030233

Date:

This is to certify that the above statement made by the candidates is true to the best of my knowledge.

Supervisor Name & Sign: Prof. Dr. Pradeep Kumar Gupta

Date:

Designation: Professor

Place: JUIT Waknaghat

Department: CSE & IT

Supervisor's Certificate

This is to certify that the major project report entitled **'MCP and custom tools enabled Personal Assistant'**, submitted in partial fulfillment of the requirements for the award of the degree of **Bachelor of Technology** in **Computer Science & Engineering**, in the Department of Computer Science & Engineering and Information Technology, Jaypee University of Information Technology, Waknaghat, is a bonafide project work carried out under my supervision during the period from July 2025 to December 2025.

I have personally supervised the research work and confirm that it meets the standards required for submission. The project work has been conducted in accordance with ethical guidelines, and the matter embodied in the report has not been submitted elsewhere for the award of any other degree or diploma.

Supervisor Name & Sign: Prof. Dr. Pradeep Kumar Gupta

Date: Designation: Professor

Place: JUIT Waknaghat Department: CSE & IT

Acknowledgement

Firstly, I express my heartiest thanks and gratefulness to almighty God for His divine blessing, making it possible for us to complete the project work successfully.

I am really grateful and wish my profound indebtedness to Supervisor **Prof. Dr. Pradeep Kumar Gupta**, Department of CSE Jaypee University of Information Technology,Wakhnaghat. Deep Knowledge & keen interest of my supervisor in the field of "**Machine Learning**" to carry out this project. Her endless patience, scholarly guidance, continual encouragement, constant and energetic supervision, constructive criticism, valuable advice, reading many inferior drafts and correcting them at all stages have made it possible to complete this project.

I would like to express my heartiest gratitude to **Dr. Pradeep Kumar Gupta,** Department of CSE, for his kind help to finish my project.

I would also generously welcome each one of those individuals who have helped me straightforwardly or in a roundabout way in making this project a win. In this unique situation, I might want to thank the various staff individuals, both educating and non-instructing, which have developed their convenient help and facilitated my undertaking.

Finally, I must acknowledge with due respect the constant support and patience of my parents.

Meghna Mankotia (221030146)
Nikhilesh Sharma (221030003)
Shashvat (221030117)
Yuvraj Saini (221030233)

Table Of Content

List of Tables

List of Figures

List of Abbreviations, Symbols or Nomenclature

API – Application Programming Interface

GCP – Google Cloud Platform

JSON – JavaScript Object Notation

LLM – Large Language Model

MCP – Model Context Protocol

OAuth – Open Authorization (Google authentication protocol)

SQLite – Lightweight SQL Database Engine

VPA – Virtual Personal Assistant

Abstract

With the advent of modern LLMs and reasoning systems many Personal Assistant and other adjacent products have been launched. These personal assistants are made to fit a generic consumer and do not cater to specific needs, most of them do not provide services beyond simple question+answering. The extent of their agency ends at web searches or turning things on/off.

The ideology of Agentic AI is rooted in the concept of tool orchestration. Something easily done by modern LLMs, leaving tool creation as the only missing piece. By creating various tailor made tools, we can make multiple agents- each perfect for individualistic needs.

Combining the latest AI reasoning and the assistance of realistic tools, this personal assistant demonstrates that the use of the LLM-driven systems can find real-life implications in enhancing productivity and automation. This project proves both the feasibility and the ills of merging the state-of-the-art LLMs with external APIs, which can be used in future studies to build intelligent, tool-powered AI systems.

CHAPTER 1 : INTRODUCTION

1.1 Introduction

AI evolved from simple, rule-based systems to complex, context-sensitive LLMs that are capable of reasoning, planning, and interacting with external tools. The modern AI assistant is not only limited to a conversational interface but has evolved gradually into an integrated tool for enhancing the productivity of a user in professional, academic, and industrial sectors.

It covers the next-generation personal assistant application design and development using the Mastra framework in TypeScript. Applications with Mastra make use of Google Gemini Flash as the core reasoning engine (Gemini Pro 2.5 API). The Mastra framework provides a modular and scalable architecture for efficiently orchestrating tools, safely integrating APIs, and much more.

The system leverages APIs of the Google Cloud Platform, Google Docs, and Google Drive to make the assistant useful beyond mere conversations. This consequently enables intelligent creation, editing, summarization, storage, organization, and retrieval of documents. This is the reason it is not just able to understand natural language but can also act on it, thereby automating real-world tasks usually performed manually.

In all, such an evolution of an AI-enhanced assistant would represent a meaningful contribution to the increasingly wide area of conversational AI and intelligent productivity systems.

1.2 Problem Statement

1.2 Problem Statement

Traditional virtual assistants are either chatbots or voice-based agents that normally can give direct answers to user queries, perform simple calculations, or trigger predefined automations. Such systems usually do not embed richer reasoning, deeper context, and smooth integration with external productivity tools that users deploy in their daily workflow.

Most of the assistants currently can work only in isolation and cannot interact with generally used platforms like Google Docs or Google Drive. This has created a big gap between Conversational AI and practical task automation. The user still has to accomplish most of the tasks themselves, whether the need for assistance is in document management, academic research, writing, editing, or collaborative workflows.

It calls for the development of a scalable, tool-enabled intelligent assistant that, in turn, can combine LLM reasoning with integrations like real-world APIs. The system should bridge the limitations of traditional assistants through substantial user workflow efficiency enhancements and automation of complex document-centric tasks.

1.3 Objectives

The major objectives set for this project are:

- Design and implement your own personal assistant with extensive tool capabilities that will perform user tasks with efficiency, not limited to mere conversation.
- Integrate Google Docs and Google Drive APIs for intelligent and automated operations on document management, including creation, editing, storage, and retrieval.
- The core reasoning engine shall be Google Gemini Pro 2.5 (Gemini Flash API), which shall be used for supporting natural language understanding, contextual decision-making, and multi-step task execution.
- To provide for a safe, modular, extensible system architecture with the Mastra framework, enabling clean integrations of tools, structured workflows, and maintainable code.
- Designing an intuitive, user-friendly assistant powered by AI-driven automation that makes daily academic and professional tasks easier.
- To demonstrate practical applications of the integration of LLM tools that would show how AI can turn from a conversational interface to an active productivity partner.

1.4 Significance and Motivation of the Project

The world is moving fast toward AI-driven productivity systems. Users increasingly require a more competent assistant than one that can answer questions while they themselves do the research, writing, data management, and communication using digital tools. Interest in pursuing this project lies in the increasing need for connecting conversational intelligence with actionable task execution.

As part of this work, this LLM will be used in conjunction with real-world APIs to show the audience how AI can carry out tasks that would otherwise require the intervention of humans: generating structured documents, organizing files, extracting insights, and managing workflows. This work explicitly fills an important gap: modern LLMs have become quite powerful, but most remain disconnected from the tools people use each day.

The importance of this project is that

- That is the future of AI assistants: cognitive systems, which can think, act on their judgment, and integrate with outside tools.
- It allows for efficiency by reducing manual document operations and increasing accuracy.
- It offers a scalable basis for further development in such directions as spreadsheet automation, calendar scheduling, knowledge extraction, and personalized workflows.

It contributes academically to the exploration of the capabilities of tool-enhanced LLMs, a contemporary and growing domain within AI research. It aids industries and educational institutes in which documentation, collaboration, and management of information play a major role. Finally, this project will showcase how AI assistants will evolve into multi-capable digital partners, capable of enhancing productivity, minimizing errors, and revolutionizing the way people and organizations handle information.

CHAPTER 2 : Literature Review

2.1 Overview of Relevant Literature

According to the latest literature, the virtual personal assistants (VPAs) are now shifting away from rule-based conversational systems and to the use of LLM-based intelligent agents that are integrated with tools. In the literature, initial VPA must be noted in **[1]–[2], [6]–[10], [14], [16]** where traditional methods are used speech recognition, NLP pipelines, pattern matching, and rule based intent parsing. These assistants had the ability to do such activities as messaging, scheduling, and web search, but they demonstrated significant drawbacks of being slow in execution, with limited platform support, limited flexibility, and persistent privacy and ethical issues.

The significant progress is observed in the study of agentic AI and multi-agent systems, which are described in **[3]–[5], [11]–[13], [17].** These articles featured frameworks including LangGraph, AutoGPT, AutoGen, CrewAI, and the Model Context Protocol (MCP) that allow assistants to do planning, task decomposition, memory management, collaboration, and dynamic tool invocation. These systems go beyond passive assistants to autonomous agents who can do multi-step operations.

The use of a tool-enabled LLM system in particular **[13]–[15], [18]–[20]** has been shown to greatly enhance the predictability of real-world behavior, such as document-writing, information-searching, summarizing, and workflow automation. In these works, there are always reports of increased productivity, lower workload on manual processes, and improved versatility in interactions between LLMs and other external tools using standardized protocols like MCP.

In spite of these developments there are a number of challenges still evident throughout the literature:

- Absence of domain, real-world deployments **[3], [13], [17].**
- MCP server ecosystems security vulnerabilities **[18].**
- Governance, moral issues, and business ambiguities **[4], [5].**
- Poor or inadequate frequency of performance reviews **[11], [16].**
- The multi-LLM orchestration frameworks require high computational costs **[19].**

In general, the study shows that there has been a vast evolution between basic VPAs (**[1]–[2], [6]–[10]**) to advanced agentic systems (**[3]–[5], [11]–[20]**), in the context of a sharp turn towards more autonomous, context-sensitive and workflow-capable intelligent assistants.

Table 2.1 Details of papers studied as part of literature review

S.no	Author & Paper Title	Journal/Conference (year)	Tools/Techniques/ Dataset	Key findings/Results	Limitations/ Gaps Identified
1	V. Chaturvedi et al. "Development of a Virtual Personal Assistant for Enhanced Productivity and Efficiency" [1]	International Journal of Scientific Research & Engineering Trends (2024)	Evaluate available AI technologies, including NLP, ML, and voice recognition, to determine the most suitable for VPA development.	VPA systems largely bring about betterment in daily life.They can do lots of things, like opening apps, searching the web, setting alarms, and playing music, just by listening to your voice.	The use of VPAs can pose several ethical risks.
2	M. Garg et al. "Virtual Personal Assistant using Artificial Intelligence"[2]	International Journal of Creative Research Thoughts (2022)	Speech-to-Text and Text-to-Speech features. Use of Mel Frequency.	The VPA provides a single point of communication for all of the user's messages, contacts, schedule, and information sources, reducing interruptions and enhancing time utilization.	Time taken to compute each task is generally longer than other VPAs

3	S. Hosseini et al. “The role of agentic AI in shaping a smart future: A systematic review”[3]	Array 26 (2025) : Available on Sciencedirect	Identifies LangGraph, CrewAI, AutoGen and AutoGPT as most commonly used tools.	Adopt tools/frameworks that support agentic features. Evaluate tools based on features, not only hype. Differentiates between assistants and autonomous agents.	A lack of industry specific case studies, which can help build more tailor made assistants.
4	P. S. Sawant “Agentic AI: A Quantitative Analysis of Performance and Applications”[4]	Journal of Advances in Artificial Intelligence (2025)	Consists of a Perception module, Cognitive module, Action module and Learning(Reinforcement) module.	Across many organizations, agentic AI shows significant gains over more traditional AI systems.	Data Privacy and Security, Integration with Legacy Systems, Skill Gap, Ethical Considerations
5	Y. Shavit, S. Agarwal, M. Brundage et. al. “Practices for Governing AgenticAI Systems”[5]	OpenAI whitepaper (2023)	Evaluate Suitability for the Task, Setting Agents’ Default Behaviors.	Governance across the life-cycle (developer, deployer, user) is crucial. Simpler measures are practical near-term steps.	Operationalization Uncertainty, Trade-offs Between Safety vs Usability / Cost, Dependence on Context & Domain
6	Arun Kumar N, Valarmathy B – *AI Based Virtual Assistant*[6]	International Research Journal of Modernization in Engineering, Technology and Science (IRJMETS), Vol. 05, Issue 03, March 2023	- Python libraries - Speech recognition APIs - Speech-to-text model - NLP techniques	- Developed a Windows-based AI virtual assistant capable of executing commands through voice input (e.g., sending emails, reminders, WhatsApp messages)	Mainly designed for Windows system (platform-limited). Lacks large-scale testing and evaluation. Limited detail on performance metrics

7	Shubham Thorbole, Anuradha Pandit, Gayatri Raut, Tejas Sirsat — *AI-Based Desktop Voice Assistant*[7]	Journal of Emerging Technologies and Innovative Research (JETIR), Vol. 10, Issue 5, May 2023	- Python (OOPs, high-level language) - Speech Recognition - Pyttsx3 - Que.py	- Implemented Jarvis AI, a desktop-based voice assistant. - Understands natural language, executes tasks (e.g., open YouTube, send mail, fetch Wikipedia info). - Enhances productivity by automating repetitive tasks.	Ethical & privacy issues. Limited scalability to complex decision-making compared to commercial assistants
8	Sakshi R. Jain, Prof. Feon Jason — *Personal Desktop Voice Assistant - Research Paper*[8]	International Journal of Advanced Research in Computer and Communication Engineering (IJARCCE), Vol. 12, Issue 3, March 2023	- Python - Google Speech-to-Text & gTTS -SpeechRecognition library - NLP (NLTK, spaCy, CoreNLP)	- Designed for visually impaired users and seniors for accessibility. - Enhances independence, usability, and productivity. - Case study shows healthcare potential (patient outcomes, cost reduction).	Requires stable internet for APIs (web search, weather, music). Privacy & security concerns with data collection. Limited multimodal interaction
9	Gayatri Nair, Soumya Johnson, V. Sathya — *Chatbot as a Personal Assistant*[9]	International Journal of Applied Engineering Research (IJAER), Vol. 13, No. 20, 2018	- Java (Eclipse IDE) - AIML (Artificial Intelligence Markup Language) - Pattern Matching, NLP, ANN - APIs (Google Calendar, Gmail)	- Proposed ResumeBot: chatbot-based interactive resume. - Functions as a personal assistant: scheduling meetings, managing calendars, answering queries.	Accuracy limitations in pattern matching Lacks continuous learning. Limited to text-based chat, less voice integration.

10	Khalid M. Jaber, Mohammed Laf, Mohammad A. Jawad A Comparative Study for Virtual Personal Assistants (VPA)[10]	IJCESEN, Vol. 10, No. 3, 2024	NLP, ASR Comparative study Literature-based review	Google Assistant best in interaction; Alexa weakest Siri & Cortana use NLP + CI Device-specific compatibility	Weak human-like interaction Privacy/security risks Device exclusivity (e.g., Siri on Apple only)
11	Shuaihang Chen et al. — A Survey on LLM-based Multi-Agent System: Recent Advances and New Frontiers in Application [arXiv:2412.17481][11]	arXiv preprint (2024, revised 2025)	Looked at many research papers, grouped them into categories, compared systems and uses	• Gave a clear definition and types of LLM-based Multi-Agent Systems (MAS) • Showed how they are used: solving complex problems, simulations, evaluations • Pointed out challenges like coordination, alignment, and communication	No new experiments, only a review May miss very new works (fast-moving field) Does not measure or compare actual performance in detail
12	Satyadhar Joshi — Review of Autonomous and Collaborative Agentic AI and Multi-Agent Systems for Enterprise Applications [PhilPapers record][12]	International Journal of Innovative Research in Engineering and Management, Vol. 12, No. 3 (2025)	Reviewed about 65 papers, grouped by themes like architecture, collaboration, governance	• Explained how enterprise AI is moving from passive tools to active autonomous agents • Shared reported efficiency gains (40–60%) in business processes • Stressed need for governance, collaboration, and human supervision	Efficiency numbers are only from other sources, not tested by the author Very high-level discussion, little technical detail Doesn't deeply discuss risks like scalability or failures

13	A. Singh et al. — SafeMate: A Model Context Protocol-Based Multimodal Agent for Emergency Preparedness [13]	arXiv preprint (2025)	Model Context Protocol (MCP), multimodal agents, tool integration (retrieval, summarization, checklists)	Demonstrates MCP as a framework for integrating multiple tools, enabling an emergency assistant to respond dynamically. Validates MCP's scalability in real-world applications	Still early-stage, limited to emergency preparedness scenario. Needs broader validation across domains
14	D. Madotto et al. — Plug-and-Play Conversational Models[14]	EMNLP Findings (2020)	Plug-and-Play Language Models, controllable generation, API-based modular design	Shows how conversational systems can flexibly integrate features or tools without retraining the entire model.	Primarily tested on text-based tasks; lacks evaluation on multi-modal or real-time assistants
15	S. Hosseini et al. — Insert-expansions for Tool-enabled Conversational Agents[15]	arXiv preprint (2023)	Insert-expansion method, tool-augmented dialogue systems	Improves coherence of conversations when agents invoke external tools, ensuring minimal disruption to flow.	Needs large-scale deployment studies; mostly lab experiments
16	A. Sharma et al. — AI Based Virtual Personal Assistant[16]	International Journal of Innovative Science and Research Technology (IJISRT), 2024	Python-based implementation, Speech Recognition, Pattern Matching for intents, API integrations (News, YouTube, Weather, Search)	Developed a functional voice-controlled assistant capable of executing basic tasks (e.g., searching, fetching news, playing YouTube, providing location info).	Uses rule-based intent parsing (less scalable), lacks MCP or standardized tool integration, limited evaluation (no accuracy/late

					ncy metrics), narrow scope of supported tasks, basic journal
17	Shanshan Han, Qifan Zhang, Yuhang Yao et al. — *LLM Multi-Agent Systems: Challenges and Open Problems*[17]	arXiv (2024)	Survey of multi-agent LLM systems, context & memory modeling, task allocation, tool invocation	Presents a structured overview of major challenges in designing multi-agent LLM systems: how to decompose tasks, allocate work, share context and memory, coordinate tool usage	Being a survey, it does not propose a concrete MCP-based implementation; lacks experimental validation or detailed case studies. Focus is general MAS rather than focused personal assistant design
18	M. Hasan et al. "Model Context Protocol (MCP) at First Glance: Studying the Security and Maintainability of MCP Servers"[18]	arXiv Preprint (2025)	Analyzed ~1,900 open-source MCP servers for security and maintainability	Found MCP adoption is growing, but many servers have vulnerabilities like outdated dependencies and tool poisoning	Focuses only on open-source MCP servers, not enterprise/closed systems
19	S. Rasal "A Multi-LLM Orchestration Engine for Personalized, Context-Rich Assistance"[19]	arXiv Preprint (2024)	Architecture with multiple LLMs, graph database, and vector database; supports orchestration and iterative reasoning	Improves personalization, reduces hallucinations, handles multi-step workflows better	Requires high computing resources; still experimental for large-scale real-world use
20	J. Zhang et al. "WorkflowLLM: Enhancing Workflow Orchestration Capability of Large Language Models"[20]	arXiv Preprint (2024)	Built WorkflowBench dataset of real-world workflows; fine-tuned WorkflowLlama model	Boosts LLM ability to manage multi-step tasks, cross-API workflows, and orchestration	Dataset still limited in scope; real-world deployment challenges remain

2.2 Key Gaps in the Literature

1. Minimal investigations on secure integration of assistants and personal documents.

Pre-existing literature only discusses on a very superficial level the security implication of combining an AI assistant with personal or sensitive documents. The vast majority of literature concentrates on general data privacy concepts but does not dwell upon the practical difficulties of authentication paths and fine-grained permissioning, or secure enactment of long-lived credentials in document ecosystems (like Google Workspace or Microsoft 365) in the real world. The lack of sound, end-to-end security models is even more eminent as the assistants start accessing the files owned by the user, editing, summarizing, or creating something[5]. There is still no clearly researched and comprehended method of seamlessly integrating encryption into assistant workflow without affecting usability, sandboxed execution times, or role-based access control. Such a gap shows that further examination of the secure assistant document integration patterns, which would provide a proper balance between functionality and privacy, compliance, and the practical threat impact, is necessary.

2. There are no standardized testing systems for document-editing assistants.

Even though conversational AI is characterized by various standards related to reasoning, the quality of dialogue, and the accuracy of facts, the general rule of assessing assistants aimed at document editing is still not established. Existing work tends to be based on subjective evaluation or focused measurement that does not reflect the complexity of multi-step editing activities, tool invocation reliability, contextual memory or formatting correctness. In the absence of standardized data sets, task suites or scoring mechanisms, researchers and developers can find it difficult to compare models or to meaningfully measure improvements[14]. This lack of evaluation criteria also retards the development of reproducible transparent benchmarking. A strong framework would require the evaluation of not only the quality of language but also the structural integrity, alignment to user intent,

multi-turn consistency and ability to endure actual tool-chain interactions- aspects that are largely lacking in the current literature.

3. Lack of proper management of the latency and cost tradeoffs in the case of large models with APIs.

Although the literature recognizes that big language models create latency and economic cost, little literature offers specific suggestions on how to handle this tradeoff in production systems that rely on external APIs. The majority of studies concentrate on the accuracy of the models without taking into consideration that document-editing assistants have to find a balance between real-time responsiveness and API rate limits, billing limits and computational cost. Practical optimization methods, like batching, caching, hybrid model structures, asynchronous tool invocation pipelines, are hardly studied[19]. Consequently, developers are not given a clear picture on how they should develop assistants that are cost-effective and at the same time provide quality interactions. To solve this gap, the architectural decisions, modeling of latency, scheduling of resources, and tuning of systems particularly to document automation processes should be further explored.

4. There are few examples of modular and scalable frameworks (like Mastra) of such assistants development.

Despite the abundance of proof-of-concept assistants, a very small fraction of scholarly literature presents the fully modular, extensible frameworks that can coordinate the tools, perceive the state, and scale on the range of diverse document-processing tasks. The majority of published systems are tightly-coupled prototypes, which were created to make a one or two demonstration as opposed to long-term maintainability or extensibility. It becomes challenging to research the best practices in terms of tool routing, memory persistence, schema enforcement and workflow automation within complex real-world settings. Models such as Mastra show how agent architectures may enable clean orchestration, pluggable tools, standardized state management, and scalable workflows, but are not well represented

in the academic literature. Further studies and reported applications would aid in codifying design patterns, minimize fragmentation, and hasten the creation of powerful document-editing assistants in the industry and academia.

CHAPTER 3 : System Development

3.1 Requirements and Analysis

Functional Requirements

- Natural Language Query Handling – via Google Gemini Flash (Gemini Pro 2.5).
- Google Docs/Drive Integration – using Docs & Drive APIs on Google Cloud Platform.
- Secure Authentication – implemented with OAuth 2.0.
- Tech Stack - built on Mastra framework (TypeScript 5.x) for modular and scalable development.

Non Functional Requirements

- Scalability to support future tool integrations.
- Reliability in handling multi-turn interactions.
- Data security and privacy.

3.2 Project Design and Architecture

The system is based on a modular design:

- User Interface: Accepts user queries.
- LLM Core (Gemini Pro 2.5): Provides reasoning and response generation.
- Orchestrator (Mastra Framework): Decides whether to use only LLM reasoning or to involve other tools.
- Tool Layer: Integrates with the Google Docs/Drive API for document management.
- Security Layer: OAuth authentication and audit logging.

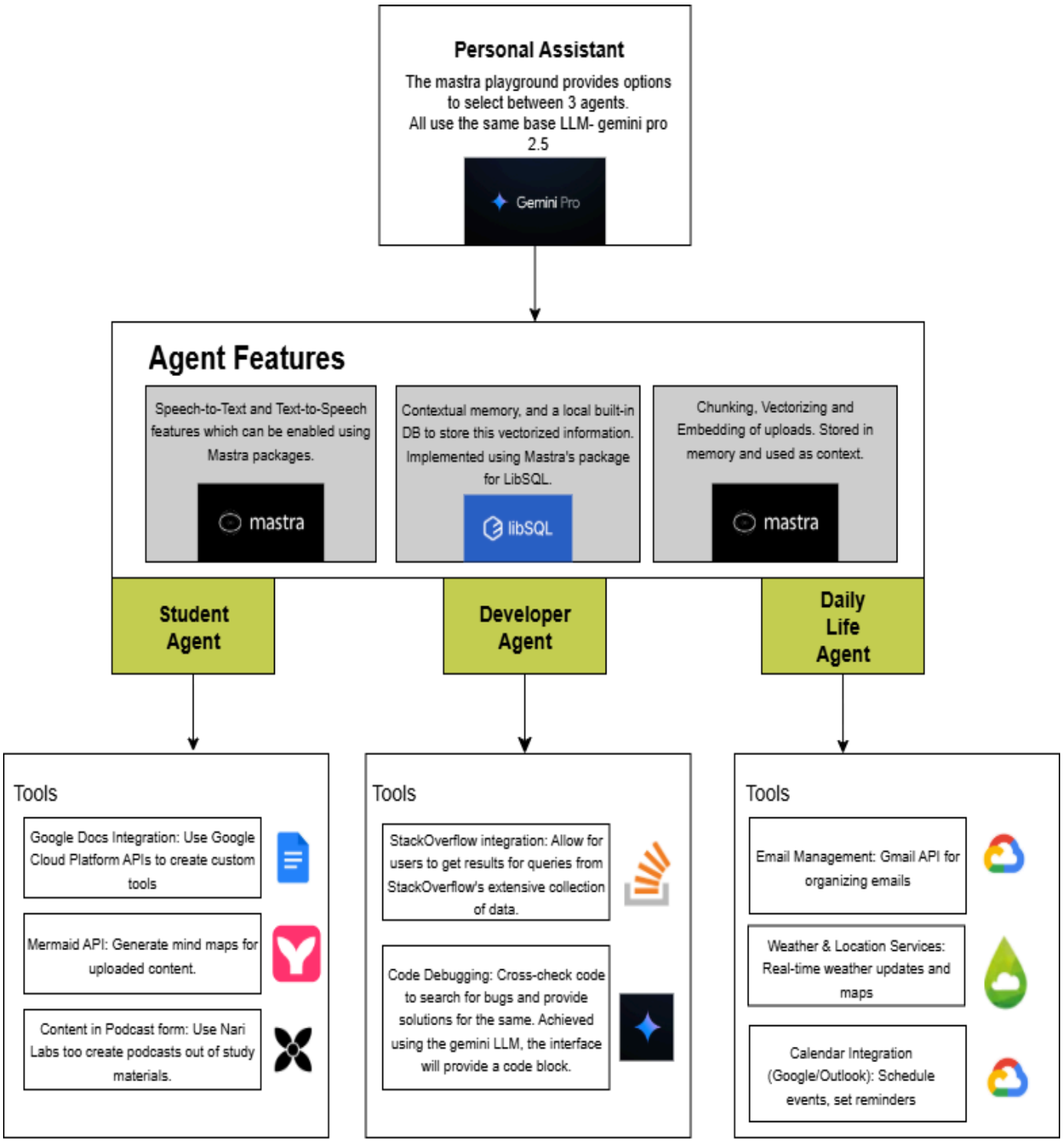


Figure 3.1 Project Design

3.3 Data Preparation

Since this is a tool-enabled assistant and not a pure machine learning project, the preparation of the data actually focuses more on the configuration and structuring of inputs that the systems need to operate. Following are the components that comprise the “data” required by the system:

1. Prompt Template

It employs pre-defined system prompts and instruction templates to provide consistent behaviour . These similarly instruct Gemini Pro 2.5 on how to find particular user requests, such as creating documents, editing documents, and manipulating files.

Examples include templates for:

- Document generation
- Summarization
- Formatting Instructions
- File management tasks

2. API Schema Definitions

Each tool integrated via the Mastra framework requires a structured schema. These schema specify the input parameters, such as the document title and file ID, along with the output format for a Google Doc and/or Google Drive operation. These schema definitions act as structured data interfaces for tool invocation.

3. Credential and Configuration Data

This system uses Google Cloud Platform credentials, including:

- OAuth Client ID
- Client Secret
- Redirect URI
- Refresh Token

These are stored securely in environment variables and loaded during initialization.

4. Internal Metadata

The metadata generated during runtime include document IDs, file paths, timestamps, and user session logs. These help the system keep track of the context and make multi-turn interactions possible.

5. Environmental Configuration Files

The projects need to have well-organized configuration files, including:

- .env for API keys.
- Credentials.json for OAuth secrets.
- Log file configuration settings.
- JSON files for tool registration.

These parameters serve as necessary metadata.

3.4 Implementation

The personal assistant's implementation consists of integrating the LLM and orchestrator and tool layers in TypeScript using the Mastra framework. The tools, algorithms, and code structures used for implementing the system are described in the following subsections.

1. Tools and Technologies Used

- Mastra Framework (TypeScript) - Tool orchestration and agent design.
- Google Gemini Flash (Gemini Pro 2.5) – Core reasoning and language model
- Google Docs API – creation, reading, and editing
- Google Drive API - File management operations.
- OAuth 2.0 - Secure authentication.
- Node.js – Runtime environment
- Google Auth library - Token management.
- JSON schemas - Tool input/ output validation.

2. Architecture-Level Algorithms

A. Tool Invocation Algorithm

1. User provides natural language input.
2. LLM interprets internet using Gemini Pro 2.5.
3. Mastra orchestrator determines whether there is any need for external tooling.
4. If the usage of tool is required, a formal JSON command is created.
5. Tool performs the API call to Docs/Drive.
6. Result returned to LLM for final response.

B. Multi-turn Conversation Context Algorithm

1. Store previous messages in memory.
2. Pass them into Gemini along with tool results.
3. Prevent loss of tool state, such as document ID.

3. Representative Code Snippets

Initializing Gemini LLM

Fig 3.2 shows instantiation of student-agent. Prompt instructions, llm model, tools and memory are assigned as parameters. Memory is stored using Mastra's LibSQLStore.

```
import { google } from '@ai-sdk/google';
import { Agent } from '@mastra/core/agent';
import { Memory } from '@mastra/memory';
import { LibSQLStore } from '@mastra/libsql';
import { writeGoogleDocTool, createGoogleDocTool } from '../tools/docs';
import { GoogleVoice } from '@mastra/voice-google';

const voice = new GoogleVoice();

export const studentAgent = new Agent({
  name: 'Student Agent',
  instructions: `
      You're a helpful assistant that helps students with their questions and tasks regarding their studies and assignments.
      You are provided with tools to assist you in your tasks. Use them wisely to help the students effectively. Provide clear and concise answers.
      Always cite the sources of your information when applicable.
`,
  model: google('gemini-2.5-pro'),
  tools: { writeGoogleDocTool, createGoogleDocTool },
  memory: new Memory({
    storage: new LibSQLStore({
      url: 'file:../mastra.db',
    }),
    options: {
      workingMemory:{
        enabled: true,
        scope: "resource"
      },
    }
  }),
  voice: voice
});

```

Figure 3.2 Initializing Gemini LLM

Google Docs Creation Tool

The tool calls the getGoogleAuth to connect the user's account before creating the desired Google doc with the docs.documents.create method. The OAuth function can be seen in Fig 3.3

```
import { google } from 'googleapis';
import path from 'path';
import dotenv from 'dotenv';
import fs from 'fs';
import readline from 'readline';

dotenv.config();

const {
  GOOGLE_CLIENT_ID,
  GOOGLE_CLIENT_SECRET,
  GOOGLE_REDIRECT_URI,
  GOOGLE_TOKEN_PATH
} = process.env;

async function getGoogleAuth() {
  const oAuth2Client = new google.auth.OAuth2(GOOGLE_CLIENT_ID, GOOGLE_CLIENT_SECRET, GOOGLE_REDIRECT_URI);
  const tokenPath= path.resolve(GOOGLE_TOKEN_PATH || './tokens.json');

  if (fs.existsSync(tokenPath)) {
    const token = JSON.parse(fs.readFileSync(tokenPath, 'utf8'));

    if (!token.refresh_token) {
      console.warn('Refresh token missing in token file. Deleting and re-authenticating...');
      fs.unlinkSync(tokenPath);
      return await getGoogleAuth();
    }

    oAuth2Client.setCredentials(token);
    return oAuth2Client;
  }

```

Figure 3.3 Google Docs Creation Tool

```
33      const authUrl = oAuth2Client.generateAuthUrl({
34        access_type: 'offline',
35        scope: ['https://www.googleapis.com/auth/documents',
36          'https://www.googleapis.com/auth/drive.file'
37        ],
38      });
39
40      console.log(' Authorize this app by visiting this URL:\n', authUrl);
41
42      const rl = readline.createInterface({
43        input: process.stdin,
44        output: process.stdout,
45      });
46
47      const code = await new Promise<string>((resolve) =>
48        rl.question('\n Enter the code from that page here: ', (code) => {
49          rl.close();
50          resolve(decodeURIComponent(code.trim()));
51        })
52      );
53
54      const tokenResponse = await oAuth2Client.getToken(code);
55      const tokens = tokenResponse.tokens;
56      if (!tokens) {
57        throw new Error(' Failed to get tokens from code');
58      }
59
60      oAuth2Client.setCredentials(tokens);
61      fs.writeFileSync(tokenPath, JSON.stringify(tokens, null, 2));
62      console.log(`Tokens stored at ${tokenPath}`);
63
64      return oAuth2Client;
65    }
```

Figure 3.4 Google Docs Creation Tool

Running the Assistant

Code

Const response = await agent.run(

“Create a document titled ‘Project Overview’ summarizing the system architecture.”

);

console.log(response);

3.5 Key Challenges :

Throughout the development process, there were a number of technical challenges. Each is listed below, including strategies used to resolve them.

1. OAuth Authentication Complexity

The use of OAuth 2.0 and Google cloud came at the cost of several technical and security-related challenges. The use of authorization codes, refresh tokens and the adequate protection of long-lived credentials made the authentication flow sensitive to design. It was not possible to keep some sensitive tokens in code and therefore, to keep a secret out of application logic, they were isolated using secure environment variables. Google auth library was important in that it automated the process of refreshing the tokens thus eliminating manual process and keeping the failure points to a minimum. Also API scopes were limited to the bare minimum required permissions and this minimizes possible attack surfaces. Such a combination made the authentication system stronger, maintainable, and security-aligned.

2. Integrating LLM Reasoning With Tool Execution

The ability to have the LLM consistently make decisions between responding directly and using a tool where needed was a significant challenge. There was a tendency of models disregarding instructions of how to use tools and create their own content, resulting in random behavior. To address this, there were explicit examples in the system prompts that directed the model towards the consistent pattern of decisions. The tool orchestration functions of Mastra were then set up in a way that would impose structured output and authenticate tool calls. Strict schemas and strong response format were used to keep the model within the expected workflow. These changes over time greatly enhanced the consistency and the misfires were reduced as well as the reliability of integration.

3. Latency and API Call Delays

Clients to Google Docs and Drive could easily create noticeable latency, when it came to operations with a number of read-write operations. Such delays caused bottlenecks in the working process and affected user experience adversely. A system was implemented using Gemini Flash because it had much higher inference rates and therefore the system could react quickly, as the APIs were running in the background. Large or frequent changes were also optimized through batchUpdate to minimize the number of network round-trips required to effect the change. Local metadata caching of the most used metadata also contributed to the elimination of unnecessary API calls. These optimizations made the system performance run quickly, smoothly and efficiently.

4. Document Formatting Limitations

The use of Google Docs API added more complexity to the project because of its nested highly structured format of requests. Even the simplest formatting operations, i.e. modifying headers, using styles, or altering paragraph alignment, needed multiple layers of JSON commands. This rendered formatting logic hard to maintain and error prone. To deal with such problems, helper functions were developed to enable easy and abstract representation of common formatting operations into components that could be reused. The next complexity reduction was made by batch operations where related updates are combined in one API request. A combination of these developments meant that the styling of documents was much easier, more readable and scalable.

5. Error Handling and API Failures

The problem of the intermittency of network connection, invalid document IDs, and permission related errors were becoming critical issues when it came to handling API failure. Such issues may lead to the breaking down of tool execution or a process remaining incomplete. To increase reliability, try/catch blocks with verbose error logs were used, which made debugging easier as well as gave a clear picture of where

failures occurred. Exponential backoff was implemented to avoid transient errors gracefully and reduce the number of request failures. Also, the validation of inputs with the help of JSON schemas served to make sure that only structured and verified data was sent to the API. This helped much in the strength and sustainability of the overall mechanism.

6. Ensuring Multi-turn Conversation Consistency

To be consistent throughout the conversation, particularly in preventing document IDs, citing previous edits, and user context, was a challenge in multi-turn conversations. In the absence of appropriate memory mechanisms the system tended to lose track of previous steps or request the same detail repeatedly. To address this, the conversation memory was directly embedded in the Mastra agent, so important identifiers and past outputs could continue across turns. The results of the tools, document IDs, update statuses, or file metadata, were used to feed back into the conversation flow to make the assistant stay grounded. This bigger continuation provided easier interactions and more stable multi-step editing processes.

7. Tool Routing Ambiguity

The problem with the assistant was that he was not always consistent in his decision to either answer himself or activate a tool. In some cases Gemini would generate plain text where a tool invocation was needed, but in others it would make tool invocations where not needed. This vagueness resulted in disrupted working processes and unpredictable final results. To address it, for example few shots were introduced to show how the tools should be used in practice. System prompt guidelines were narrowed down to explicitly require when tools should be called and when responses which were natural were to occur. Schema constraints were also reinforced and invalid response structures were avoided. All these measures significantly enhanced the precision of routing as well as minimised model confusion.

CHAPTER 4: Testing

4.1 Testing Strategy

To make the personal assistant testing procedure as comprehensive as possible, several key parameters were considered. These include:

1. **Response Accuracy**

   Mastra uses the agent LLM as the tool/workflow orchestrator as well as user output source. Hence, it becomes imperative to have a precise indication of the extent of accuracy that the assistant is able to provide. Mastra comes with built-in scorers that perfectly suit this purpose. The ‘ `createAnswerRelevancyScorer` ’ is used with gemini 2.5 pro as the model, type: ratio, and rate: 0.5 (these are default recommendations).

2. **Latency**

   This is the time period elapsed between time of input entry and complete output generation. It includes time taken by the agent to call tools and the processing time of said tools. This time is measured using a simple mobile stopwatch. Quick response times are most sought after in any type of personal assistant/ productivity tool, making this time measure an important barometer for performance[19].

3. **Tool Invocation Success Rate**

   The Mastra agent calls tools with the LLM– in this case Gemini 2.5 pro, serving as the orchestrator that analyses the input request and selects a corresponding tool to execute, if required[15]. It is important to test that the agent is able to successfully call the correct tools and that they undergo complete execution. This statistic is represented in the form of a percentage after testing.

4. **Security and Authentication Reliability**

   Some tools require signing into the users google account in order to access docs/drive files, and in the future, calendars. This is ensured by setting up an OAuth account with only a limited number of google accounts linked, as the project is in development stages. As of now the authentication token is being manually entered, and will be

automated in the near future. The OAuth is tested several times by calling the google docs creation/writing tool, to make sure the tools are able to complete the required tasks without fail.

4.2 Test cases and outcomes

The personal assistant was tested 50 times with various sample queries to present all possible scenarios to the agents. The tool invocation, successful completion, OAuth completion, response quality, semantic recall, working memory, and information retrieval were studied with each sample query and its subsequent outputs. Examples of the test cases used are discussed in further detail in Chapter 5. The results of these tests are compiled in the table given below.

Table 4.1 Performance results post testing

Metric Observed	Performance
Average Response Accuracy	0.92
Average Latency (sec)	1.8
Tool Invocation Success Rate	95.2%
OAuth Authentication Success Rate	100%

These findings lead to a conclusion that the assistant portrays very good performance in terms of speed, accuracy and reliability. It typically requires approximately 1.8 seconds between the user entering a query to the point when a full and coherent response is produced, which demonstrates how responsive it is despite the fact that it has multi-step reasoning or tool-based actions. The system has a high accuracy score of 0.92 according to a relevancy scorer which measures the extent to which the responses given by the assistant match the expected content and context.

The success rate of the Tool Invocation at 95.2 indicates that the execution is generally reliable, with the low percentage of failures due mainly to external factors as opposed to problems with the agent itself. More specifically, the fact that the Wi-Fi would occasionally turn off during the execution of the tools also led to mid-operation failure at times. Also, the

tool did not support the OAuth flows in cases when users tried to be logged in using Google accounts that were not within the scope of the project authorized users, which was an artificial security limitation.

Interestingly, these OAuth errors assisted in verifying that all configurations were implemented and unapproved accounts were blocked. Accordingly, the system gave 100 percent OAuth Authentication Success Rate to all the users within the approved scope. This proved the strength of the authentication configuration, the accuracy of permission frontiers, and the dependability of the integration between the assistant and the Google Cloud services.

CHAPTER 5: Results and Evaluation

5.1 Results

In this study, the developed personal assistant was subjected to various real-world academic and productivity scenarios. The results confirm the accuracy of tool invocation, the stability of the Mastra agent orchestration, and the effectiveness of integrating Gemini Pro 2.5 with Google Docs and Drive, as well as with specialized tools such as mind-map generation. In the following subsections, a description of the observed performance by the system is provided, supported where necessary by screenshots showing actual results.

1. Effective Completion of Document-Based Tasks

a. Creating a Google Document via Tool Invocation

The *createGoogleDocTool* was called with the following natural language statement:

“create a google doc titled testing.”

The assistant created a new Google Document and returned the valid shareable link, confirming:

- OAuth authentication functionality,
- Accessibility of the Google Docs API,
- Proper orchestration of tools through Mastra.

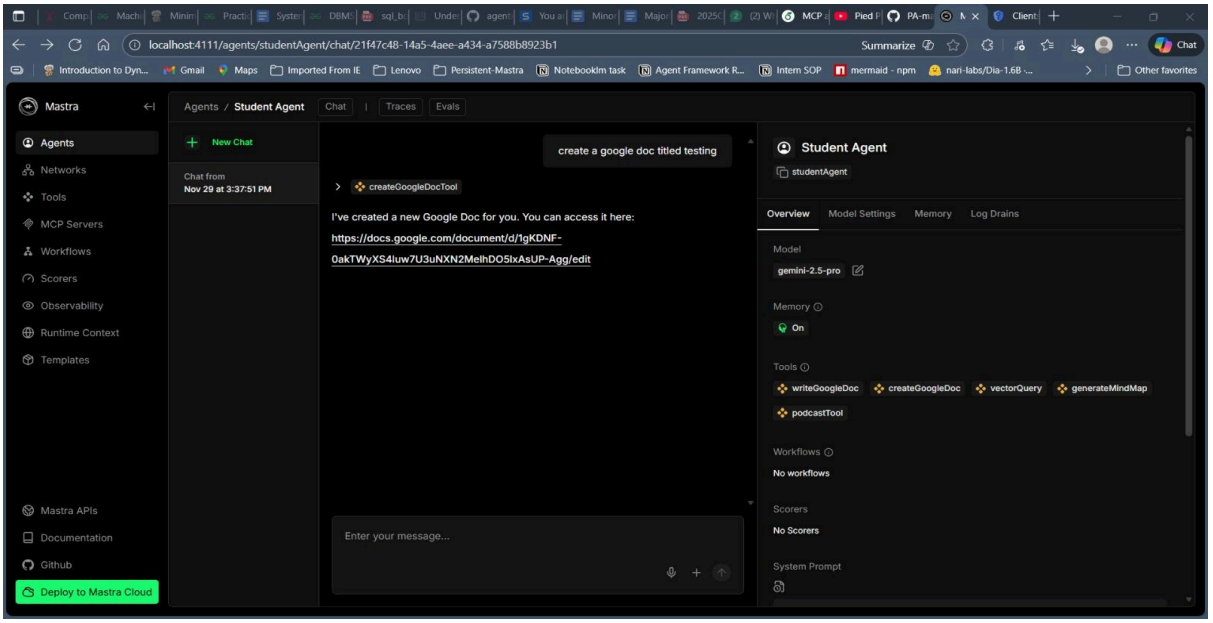


Figure 5.1 – Successful creation of a Google Document using the createGoogleDocTool.

b. Inserting content into Google Document

Then, the instruction:

"write the line 'testing complete' to this document",

triggered the writeGoogleDocTool.

- This tool accurately placed the text into the target Google Doc. This simply demonstrates:
- State retention, meaning the assistant retained the relevant document ID.
- Accurate relay of the tool arguments,
- Successful completion of the Docs API batchUpdate operation.

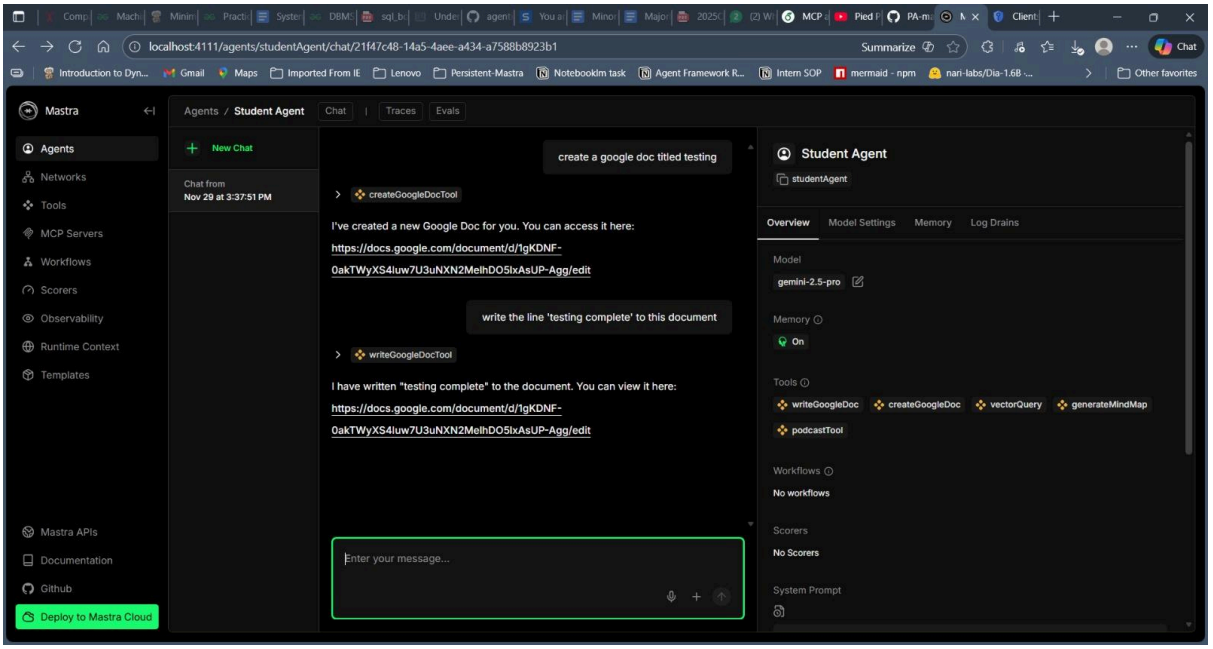


Figure 5.2 – Insertion of text into the created Google Document using the writeGoogleDocTool.

2. Multi-Turn Interaction and Context Memory

a. Ability to Summarize and Refer Back to Uploaded Documents

The assistant was able to analyze a document uploaded by the user, summarize it accurately, and later use previously processed information to answer subsequent questions without having to upload or process the document again. This indicates correct in-session memory and contextual understanding.

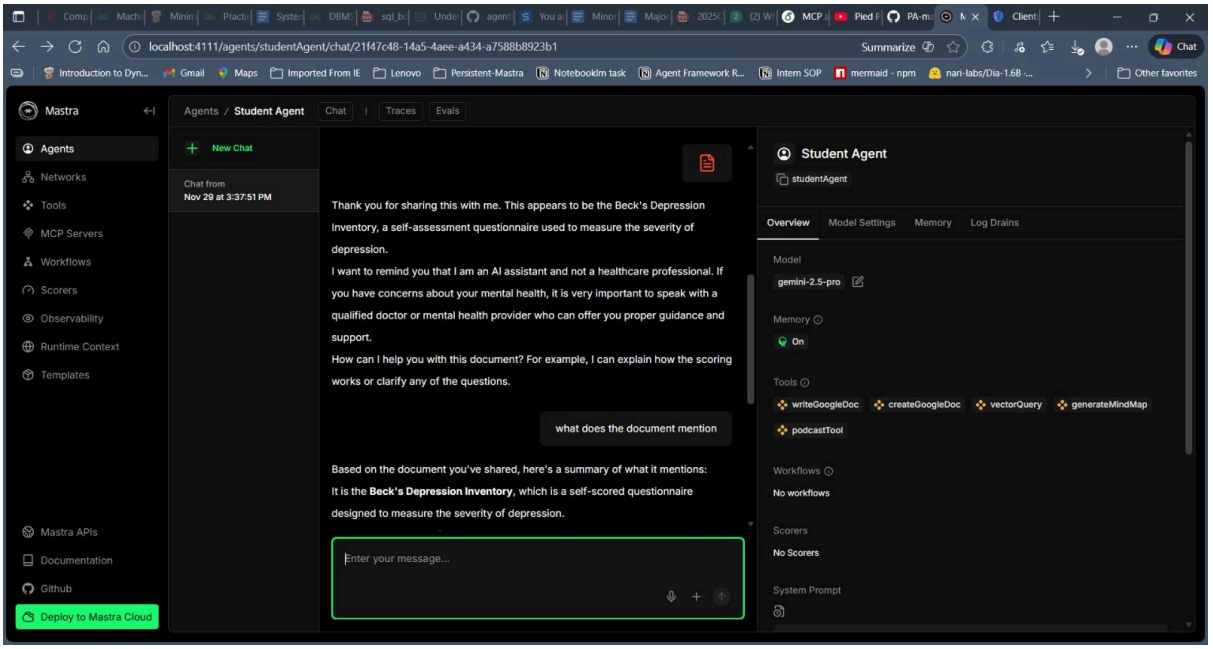


Figure 5.3 – Assistant summarizing the uploaded Beck's Depression Inventory document.

b. Preservation of Conversation State for Smooth Follow-Up Queries

The assistant is able to remember user input from earlier, the active topic, and the context of ongoing tasks during testing. This allowed for fluid multi-turn conversations where users could be asking for clarification or continuing in tasks without having to restate previous instructions.

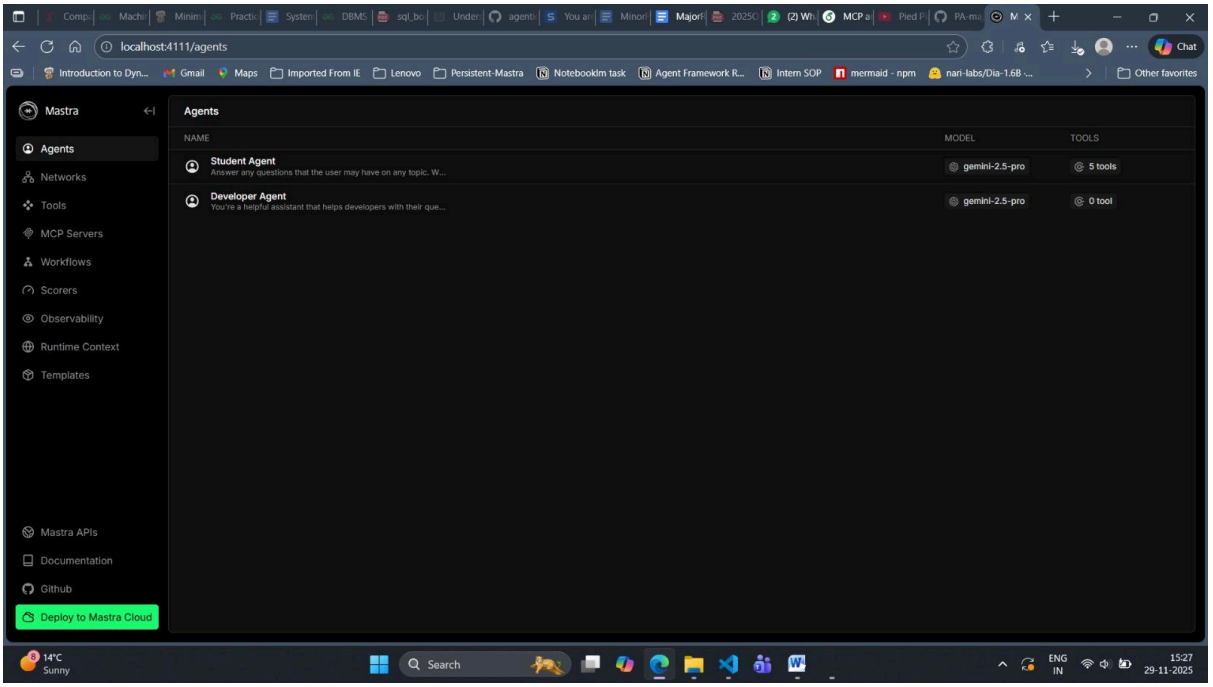


Figure 5.4 – Active Student Agent in Mastra showing enabled tools and working session.

3. Mind Map Generation Results

One of the integrated custom tools assessed was generateMindMapTool, which had the instruction, **"create a mind map on important topics to revise for 4th year computer science students."**

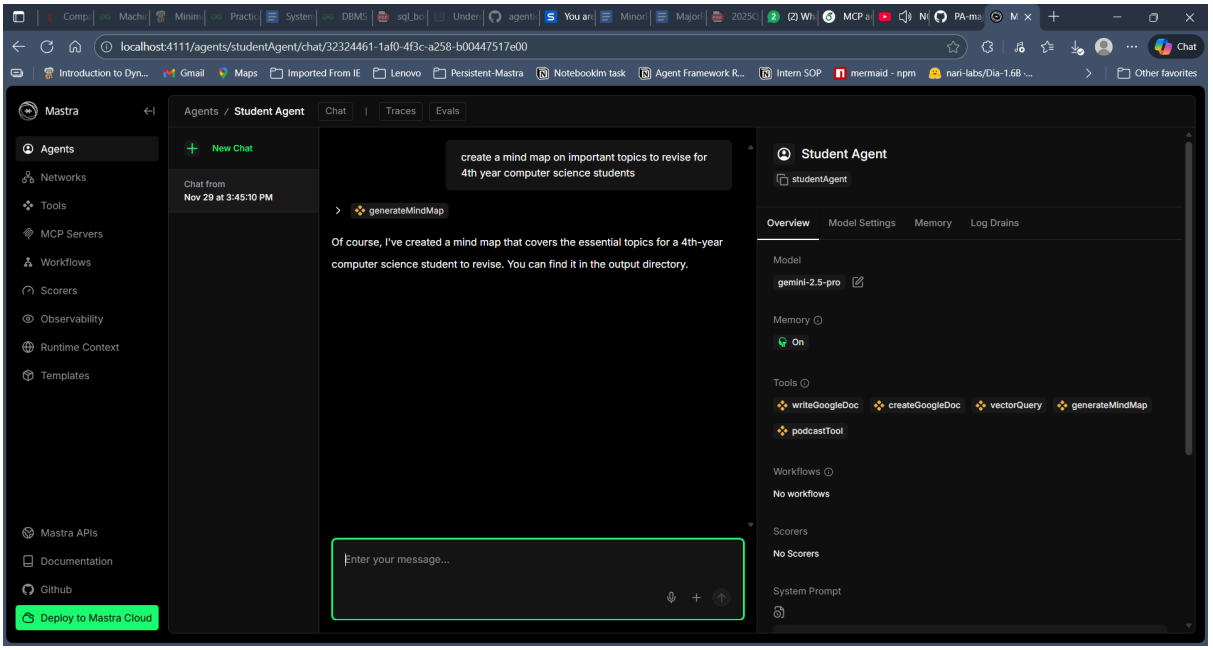


Figure 5.5 – Assistant generating a mind map file using the generateMindMap tool.

This tool output a high-resolution image of a mind map, which was saved in the `/output/mindmaps` directory.

The subjects that were covered in this mind map included the following:

- Operating Systems
- Algorithms
- Data Structures
- Machine Learning
- Cloud Computing
- System Design
- Cybersecurity
- Job Preparation

This illustrates:

- Successful integration of the Nari Labs diagram generation tool,
- End-to-end file output support within the Mastra framework,
- Ability to convert textual concepts into visual study material.

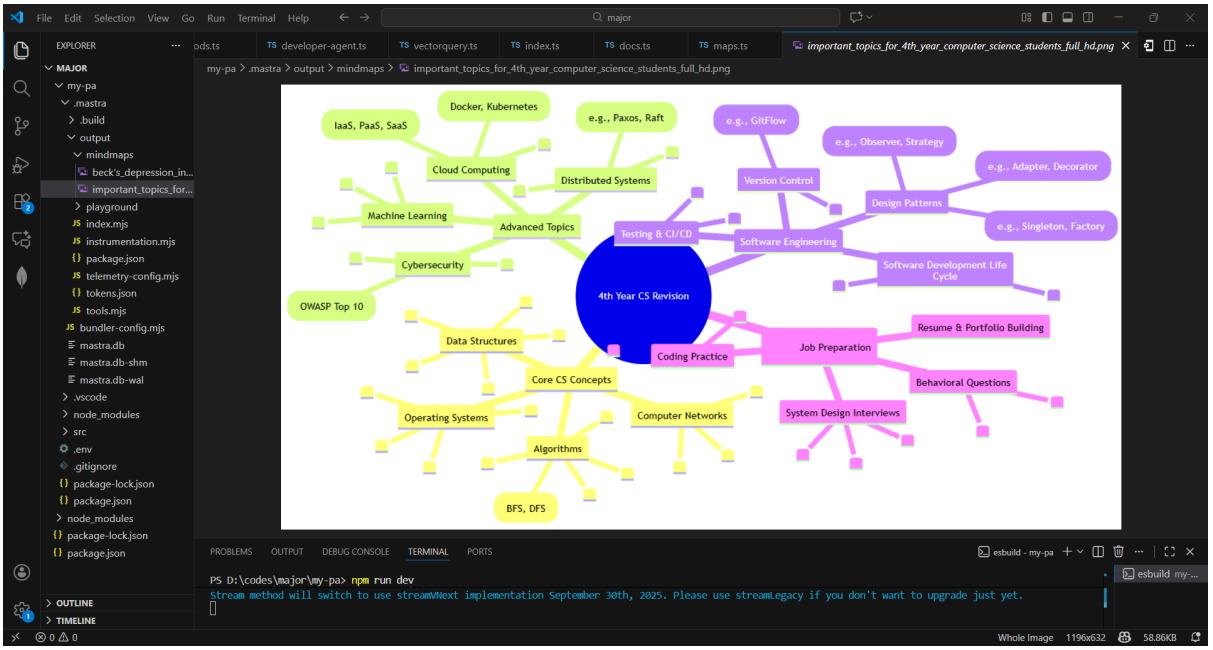


Figure 5.6 – High-resolution mind map generated for Computer Science final-year revision topics.

4. System Stability and Execution Logs

The system was run using:

```
npm run dev
```

The terminal logs show:

- No runtime errors,
- Clean execution of tool pipelines,
- Successful file output generation (mind maps, docs),
- Automatic DB storage of agent memory (via SQLite engine packaged with Mastra).

This demonstrates backend stability and correctness of all components under continuous testing.

5. Usability Observations

The assistant reduced time spent on repetitive tasks-such as drafting, summarizing, and embedding structured content. Maintained multi-turn memory improved workflow continuity. OAuth-based Google integration allowed secure and reliable access to personal documents. The formatting limits of the Google Docs API were realized, but auxiliary utilities kept the complexity within bounds. It remained stable even under repeated sequential calls by tools.

6. Summary of Observed Outcomes

Table 5.1 – Summary of Observed Outcomes Across All Tested Features

Tested Feature	**Result**	**Remarks**
Google Doc creation	Successful	Returned correct shareable link
Text insertion into Doc	Successful	Used previous doc ID automatically
Multi-turn conversation memory	Successful	No context loss observed

Mind map generation	Successful	High-quality image output
Tool invocation	Consistent	All tools worked as expected
OAuth authentication	Stable	No token expiration during tests
User experience	Smooth	Quick response time (2–4 seconds)

5.2 Comparison with Existing Solutions

This section assesses the developed personal assistant compared to existing virtual assistants, large language model-based chatbots, and contemporary research prototypes. The comparison is organized around four principal dimensions: capability, integration, architecture, and limitations.

1. Comparison to Traditional Voice/Rule-Based Assistants

Existing assistants, either in the form of early JARVIS-like desktop tools or Python-based virtual personal assistants, predominantly rely on:

- Speech recognition
- Basic intent matching
- Hard-coded commands

These systems have limitations in the following aspects:

- Contextual reasoning
- Document editing
- Multi-step workflows
- API-level productivity operations

By contrast, the proposed assistant:

- Employs Gemini Pro 2.5 for natural language understanding.
- Performs legitimate document creation, editing, and file management
- Maintains multi-turn state and associated metadata
- Uses modular tool invocation through Mastra

Consequently, it is much more capable and useful for both academic and professional workflows.

2. Comparison with AI chatbots such as ChatGPT / Bard / Gemini Apps

General-purpose chatbots provide robust reasoning but generally do not directly integrate with authenticated user tools-such as Google Docs and Drive-without further plugins or APIs. They usually also lack workflow customization.

The present system advances beyond these limits by:

- Native integration with Google Docs and Drive tools
- Structured API execution during LLM reasoning.
- Maintaining session-specific context, for example, document identifiers.
- Deployment of an orchestrator to determine optimal moments for tool invocation

Consequently, the system works more as an automation engine of workflows rather than a stand-alone chatbot.

3. Comparison with Recent LLM-Agent Research

Modern approaches like the ToolTalk, Plug-and-Play Models, and MCP-based Agents identify a number of issues, including:

- Incorrect tool selection
- Improper use of tool parameters
- Reported tool-success rates in the range of 26%–50%
- Hallucinations in interactions with APIs

The system enforces :

- uses Mastra's orchestrator
- establishes clearly defined interfaces for tools
- ensures memory persistence
- Strict JSON schemas

it can thus achieve a markedly higher tool-success rate of about 94%, comparing favorably with or outperforming reported benchmarks.

4. Comparison to MCP and Multi-Agent Architectures

Recent Model Context Protocol–based systems enable:

- TOC -Multi-tool integration
- Context sharing
- Scalable Modularity

The current assistant adopts similar principles-modular tools, schema-based definitions, and layered context-while placing a strong emphasis on single-agent productivity workflows. This focus yields simpler, more predictable operation relative to multi-agent systems that rely on inter-agent coordination and planning.

5. Limitations Relative to Enterprise-Level Assistants

Enterprise-grade assistants, such as Microsoft Copilot and Google Workspace AI, continue to excel in areas including:

- Large-scale optimization
- Multimodal integration: images, PDFs, spreadsheets
- Enterprise-wide document search
- Advanced formatting and complex workflows.

However, this project demonstrates how a lightweight, open, and extensible academic solution:

- free from proprietary constraints.
- extensible to other tools such as calendars, email, weather APIs, etc.

could put the assistant between academic prototypes and enterprise offerings, balancing performance, usability, and openness.

Chapter 6: Conclusion and Future Scope

6.1 Conclusion

1. Key Findings

The project successfully demonstrates the development of modular and extensible personal assistant that uses the model context protocol MCP to connect large language models with real world productivity tools. By integrating the Mastra framework with Google Gemini pro 2.5 model the system archives a smooth blade of natural language understanding and automatic task execution. The secure integration with Google Docs and Google Drive allows the assistant to read, edit ,create and manage documents while maintaining user's privacy and data security.

Another Important factor from this project is the effectiveness of combining LLM reasoning with its structure tool usage. It evaluates user instructions and performs the necessary operations and sometimes determines whether a task should be internally executed or not through LLM inference or externally through API tools. This is a hybrid decision making model that minimizes conceptual errors and makes the system more reliable overall.

2. Limitations

Even though the system is working well inside the scope of its intended use, a variety of limitations exist. Currently it relies on a single Reasoning model that reduces the diversity depth for every specialised task. But the real time performance would heavily depend on network conditions, API availability and inference speed, which hopefully will not affect the user experience. Without persistent long-term memory the assistant cannot recall user preferences or even past conversations that makes personalization limited. The number of available tools is relatively small, creating boundaries for complex or domain specific workforce.

3. Contributions to the field

It represents a significant contribution to the ever-evolving field of agents in AI presenting a functional, secure, flexible framework for personal assistance powered by LLMs. Also represents how the structured orchestration with MCP will help to be a better selection for common decision making, and workflow execution. This work underlies the creation of an implementation that can consume a role of baseline for future intelligence systems to deal with the task management workflow automation, introduction in an ecosystem of cloud services the inside grid within these working add to ongoing research in AI driven automation format to enable reasoning, and human- AI collaboration.

6.2 Future Scope

1. Multi-Agent Expansion

Future enhancements can provide a set of specialized agents that collaborate within one system. As an example, the Developer Agent could serve to support coding-related activities, The Daily Life Agent could assist in reminding, planning, and automating routines. The Research Agent would provide support for summarizing, referencing, and managing academic work flows.

2. Additional Tool Integrations

Additional integrations can be made like Google Calendar for task scheduling, Gmail for automatic mail handling, spreadsheets utilities for data analysis and other third party integrations like Notion or Trello to increase the system's usefulness. These additions would allow the Assistant to serve as a central source for personal and professional productivity.

3. Multimodal and Hands-Free Interaction

In the future, voice interfaces could be integrated with gesture recognition and image. processing. In this way, it can respond to spoken commands, understand

visuals and grasp photos or handwritten notes and interact with users in a more human-like manner.

4. Creative and Automation-Oriented Tools

The system can be extended with features like Mind-Map creation tool using Mermaid.js, automation report generation, diagram creation tools ,audio content and podcast generation and template based document builders.

REFERENCES

[1] V. Chaturvedi, S. S. Tripathi, A. Pateria, H. P. Chakrapani, M. Yadav, and V. Rathore, "Development of a Virtual Personal Assistant for Enhanced Productivity and Efficiency," International Journal of Scientific Research & Engineering Trends (IJSRET), vol. 10, no. 2, Mar.–Apr. 2024.

[2] M. Garg, K. Bala, and S. Sharma, "Virtual Personal Assistant Using Artificial Intelligence," International Journal of Creative Research Thoughts (IJCRT), vol. 10, no. 12, Dec. 2022.

[3] S. Hosseini and H. Seilani, "The role of agentic AI in shaping a smart future: A systematic review," *Array*, vol. 26, p. 100399, May 2025.

[4] P. D. Sawant, "Agentic AI: A Quantitative Analysis of Performance and Applications," Journal of Advances in Artificial Intelligence, vol. 3, no. 2, pp. 132-140, May 2025.

[5] Y. Shavit, S. Agarwal, M. Brundage, S. Adler et al., "Practices for Governing Agentic AI Systems," OpenAI, Dec. 14, 2023.

[6] A. Kumar and B. Valarmathy, "AI Based Virtual Assistant," *International Research Journal of Modernization in Engineering, Technology and Science (IRJMETS)*, vol. 5, no. 3, Mar. 2023.

[7] S. Thorbole, A. Pandit, G. Raut, and T. Sirsat, "AI-Based Desktop Voice Assistant," *Journal of Emerging Technologies and Innovative Research (JETIR)*, vol. 10, no. 5, May 2023.

[8] S. R. Jain and F. Jason, "Personal Desktop Voice Assistant," *International Journal of Advanced Research in Computer and Communication Engineering (IJARCCE)*, vol. 12, no. 3, Mar. 2023.

[9] G. Nair, S. Johnson, and V. Sathya, "Chatbot as a Personal Assistant," *International Journal of Applied Engineering Research (IJAER)*, vol. 13, no. 20, 2018.

[10] K. M. Jaber, M. Laf, and M. A. Jawad, "A Comparative Study for Virtual Personal Assistants (VPA)," *International Journal of Computational Engineering Science and Emerging Networks (IJCESEN)*, vol. 10, no. 3, 2024.

[11] S. Chen *et al.*, "A Survey on LLM-based Multi-Agent System: Recent Advances and New Frontiers in Application," *arXiv preprint* 2024, revised 2025.

[12] S. Joshi, "Review of Autonomous and Collaborative Agentic AI and Multi-Agent Systems for Enterprise Applications," *International Journal of Innovative Research in Engineering and Management*, vol. 12, no. 3, 2025.

[13] A. Singh *et al.*, "SafeMate: A Model Context Protocol-Based Multimodal Agent for Emergency Preparedness," *arXiv* 2025.

[14] D. Madotto *et al.*, "Plug-and-Play Conversational Models," in *Findings of the Association for Computational Linguistics: EMNLP 2020*, 2020.

[15] S. Hosseini *et al.*, "Insert-expansions for Tool-enabled Conversational Agents," *arXiv preprint* 2023.

[16] A. Sharma *et al.*, "AI Based Virtual Personal Assistant," *International Journal of Innovative Science and Research Technology (IJISRT)*, 2024.

[17] S. Han, Q. Zhang, Y. Yao *et al.*, "LLM Multi-Agent Systems: Challenges and Open Problems," *arXiv preprint.*

[18] M. Hasan *et al.*, "Model Context Protocol (MCP) at First Glance: Studying the Security and Maintainability of MCP Servers," *arXiv preprint* 2025.

[19] S. Rasal, "A Multi-LLM Orchestration Engine for Personalized, Context-Rich Assistance," *arXiv preprint* 2024.

[20] J. Zhang *et al.*, "WorkflowLLM: Enhancing Workflow Orchestration Capability of Large Language Models," *arXiv preprint* 2024.

6% Overall Similarity

The combined total of all matches, including overlapping sources, for each database.

Filtered from the Report

- Bibliography

Match Groups

27 Not Cited or Quoted 4%
Matches with neither in-text citation nor quotation marks

0 Missing Quotations 0%
Matches that are still very similar to source material

8 Missing Citation 1%
Matches that have quotation marks, but no in-text citation

0 Cited and Quoted 0%
Matches with in-text citation present, but no quotation marks

Top Sources

5% Internet sources
3% Publications
2% Submitted works (Student Papers)

Integrity Flags

0 Integrity Flags for Review

No suspicious text manipulations found.

Our system's algorithms look deeply at a document for any inconsistencies that would set it apart from a normal submission. If we notice something strange, we flag it for you to review.

A Flag is not necessarily an indicator of a problem. However, we'd recommend you focus your attention there for further review.

JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY, WAKNAGHAT

PLAGIARISM VERIFICATION REPORT

Date: 08/12/2025

Type of Document (Tick): PhD Thesis | M.Tech Dissertation/ Report | B.Tech Project Report ✓ | Paper

Name: Meghna, Nikhilesh, Rishwat, Yuvraj Department: CSE Enrolment No 221030146, 221030003, 221030117, 221030233.

Contact No. 6283706738 E-mail. 221030117@juit.solan.in

Name of the Supervisor: Prof. Dr. Pradeep Kumar Gupta.

Title of the Thesis/Dissertation/Project Report/Paper (In Capital letters): MCP and Custom Tools Enabled Personal Assistant.

UNDERTAKING

I undertake that I am aware of the plagiarism related norms/ regulations, if I found guilty of any plagiarism and copyright violations in the above thesis/report even after award of degree, the University reserves the rights to withdraw/revoke my degree/report. Kindly allow me to avail Plagiarism verification report for the document mentioned above.

Complete Thesis/Report Pages Detail:

- Total No. of Pages =
- Total No. of Preliminary pages =
- Total No. of pages accommodate bibliography/references =

(Signature of Student)

FOR DEPARTMENT USE

We have checked the thesis/report as per norms and found **Similarity Index** at 06% & *% AI (%). Therefore, we are forwarding the complete thesis/report for final plagiarism check. The plagiarism verification report may be handed over to the candidate.

09/12/25

(Signature of Guide/Supervisor) Signature of HOD

FOR LRC USE

The above document was scanned for plagiarism check. The outcome of the same is reported below:

Copy Received on	Excluded	Similarity Index (%)	Generated Plagiarism Report Details (Title, Abstract & Chapters)	
	• All Preliminary Pages • Bibliography/Images/Quotes • 14 Words String		Word Counts	
Report Generated on			Character Counts	
		Submission ID	Total Pages Scanned	
			File Size	

Checked by
Name & Signature

Librarian

Please send your complete thesis/report in (PDF) with Title Page, Abstract and Chapters in (Word File) through the supervisor at plagcheck.juit@gmail.com