A

Mini Project Report

On

"VeerAI: Offline Desktop AI Assistant"

Submitted in partial fulfillment of the requirements for the Degree

Third Year Engineering – Computer Science Engineering (Data Science)

By

ABBAS SANGAMESHWARI	**23107116**
ATHARV PARAB	**23107038**
RUGVED SAPKAL	**24207011**

Under the guidance of

Ms. AISHWARYA LONDHE

DEPARTMENT OF COMPUTER SCIENCE ENGINEERING (DATA SCIENCE)

A.P. SHAH INSTITUTE OF TECHNOLOGY

G.B. Road, Kasarvadavali, Thane (W) - 400615

UNIVERSITY OF MUMBAI

Academic year: 2025-26

CERTIFICATE

This to certify that the Mini Project report on **"VeerAI: Offline Based AI Desktop Assistant"** has been submitted by Abbas Sangameshwari (23107116), Atharv Parab (23107084), and Rugved Sapkal (24207011) who are bonafide students of A. P. Shah Institute of Technology, Thane as a partial fulfillment of the requirement for the degree in **Computer Science and Engineering (Data Science)**, during the academic year **2025-2026** in the satisfactory manner as per the curriculum laid down by University of Mumbai.

Ms. Aishwarya Londhe

Guide

Dr. Pravin Adivarekar

HOD, CSE (Data Science)

Dr. Uttam D. Kolekar

Principal

External Examiner:

1.

Internal Examiner:

1.

Place: A. P. Shah Institute of Technology, Thane

Date:

ACKNOWLEDGEMENT

This project would not have come to fruition without the invaluable help of our guide **Ms. Aishwarya Londhe** gratitude towards our HoD, **Dr. Pravin Adivarekar**, and the Department of Computer Science and Engineering (Data Science) for providing us with the opportunity as well as the support required to pursue this project. We would also like to thank our project coordinator **Ms. Aavani Nair** who gave us her valuable suggestions and ideas when we were in need of them. We would also like to thank our peers for their helpful suggestions.

TABLE OF CONTENTS

ABSTRACT

This report presents the development and evaluation of VeerAI, an innovative offline desktop-based intelligent assistant designed to deliver seamless interaction, task automation, and decision support without reliance on cloud infrastructure. VeerAI employs a dual-model strategy that integrates Ollama's Mistral large language model for natural language question answering with a suite of custom AI algorithms including heuristic search, Bayesian inference, and random forest classifiers for context-aware reasoning and problem-solving. The system's robust architecture encompasses an Electron-based framework with a lightweight web-standard interface (HTML, CSS, JavaScript), ensuring accessibility, modularity, and platform independence.

Key features include conversational assistance, file and application management, and intelligent decision-making, all while emphasizing user privacy and offline operability. The assistant's design integrates optimized algorithmic pipelines with responsive UI components to enhance usability and efficiency. The performance of VeerAI is rigorously evaluated using metrics such as task completion accuracy, response latency, and user satisfaction surveys, highlighting its effectiveness as a secure, versatile, and user-centric desktop companion. These results underscore VeerAI's potential as a practical foundation for future offline AI agents that harmonize large language models with tailored algorithmic intelligence to enhance autonomy and engagement.

Keywords VeerAI, Offline desktop assistant, Ollama Mistral, Large Language Model (LLM), Heuristic search, Bayesian inference, Random Forest classifiers, Electron framework, Web-standard interface, Privacy-focused AI, Task automation, Conversational assistance, File and application management, Response latency, Task accuracy, User satisfaction surveys.

Chapter 1
Introduction

The rapid evolution of Artificial Intelligence (AI) has reshaped human-computer interaction, driving the creation of intelligent systems that enhance efficiency, convenience, and personalization. Despite their widespread use, most AI assistants today depend heavily on cloud infrastructure, leading to concerns regarding privacy, latency, and accessibility. This chapter introduces VeerAI a privacy-focused, offline intelligent desktop assistant that operates entirely without internet dependency. It outlines the purpose behind developing VeerAI, identifies the challenges posed by existing cloud-based systems, and defines the objectives and scope of the project. Through this foundation, Chapter 1 sets the stage for understanding how VeerAI bridges the gap between intelligence and independence in AI-driven automation.

1.1 Purpose

In recent years, Artificial Intelligence (AI) has transformed how humans interact with computers. From voice-controlled assistants to recommendation engines, intelligent systems have become part of everyday digital experiences. However, most of these assistants rely on cloud computing and external APIs, resulting in data privacy concerns and dependency on internet connectivity.

The purpose of this project, VeerAI, is to develop an offline intelligent desktop assistant that performs all its core operations locally. It aims to provide users with a personalized, secure, and efficient AI companion that understands natural language, performs tasks, retrieves information, and integrates seamlessly with the operating system without requiring any internet access.

VeerAI is designed to demonstrate that offline AI can achieve the same intelligence, flexibility, and utility as cloud-based systems. It leverages advanced language models and a modular architecture to create a self-contained AI ecosystem that prioritizes privacy, performance, and accessibility.

1.2 Problem Statement

The problem statement for VeerAI centers on the limitations of existing AI assistants, which are predominantly cloud-driven and fail to provide offline functionality, strong privacy guarantees, and extensive desktop-level automation. Users often face inefficiencies and risks in depending on internet-reliant systems that do not adapt well to individual preferences or localized workflows. VeerAI seeks to address these issues by functioning as a secure, offline-first, and intelligent companion.

Key problems identified include:

- Dependence on Internet Connectivity:Users cannot access AI functionalities without an active internet connection.
- Privacy Risks: Sensitive personal data, such as voice commands and activity logs, are transmitted to third-party servers.
- Lack of Customization: Proprietary systems do not allow developers or users to modify or extend assistant capabilities easily.
- Latency and Performance Issues: Network delays often slow down real-time response generation and task execution.
- Limited Offline Functionality: Existing systems fail to provide meaningful interactions when disconnected from the internet.

Therefore, there exists a need for a locally hosted, offline-capable intelligent desktop assistant that maintains privacy, provides immediate responses, and offers customizable and extensible features.

1.3 Objectives

The main objectives of VeerAI are as follows:

- To enable offline AI processing using local LLM (Llama 3 via Ollama) for natural language understanding.
- To implement persistent memory with ChromaDB and PostgreSQL for context-aware interactions.
- To automate desktop tasks through modular actuators and APScheduler for seamless task execution.
- To ensure user privacy and data security by processing all information locally without cloud dependence.

1.4 Scope

The scope of VeerAI encompasses the design, development, and evaluation of a complete offline AI ecosystem.

It includes the following components:

- Offline NLP & LLM Integration:Using Ollama and LangChain, the assistant can understand and generate responses without internet access.
- System Integration:VeerAI interacts with the operating system to perform commands such as opening files, scheduling reminders, or notifying the user.

- Knowledge Management:Contextual data and embeddings are managed locally through PostgreSQL and ChromaDB databases.
- User Interface:A desktop-based interface built with Electron.js ensures platform independence and modern UI/UX experience.
- Automation:The assistant performs time-based or event-based actions through APScheduler, ensuring dynamic task scheduling.
- Data Privacy and Customization:The entire AI pipeline runs locally, providing complete data ownership to users and allowing custom model integration.

The project focuses on creating a balance between intelligence and independence, establishing VeerAI as a foundation for privacy-respecting personal assistants.

Chapter 2
Literature Review

The literature review provides a comprehensive analysis of existing research, frameworks, and technologies that have shaped the evolution of intelligent personal assistants. It focuses on prior advancements in artificial intelligence, natural language processing (NLP), and hybrid assistant architectures while emphasizing the growing need for privacy-preserving and offline systems. This chapter examines key studies and open-source projects that form the foundation for VeerAI, highlighting how each has contributed to areas such as contextual understanding, speech recognition, and on-device intelligence. By critically evaluating these developments, the review identifies both the innovations achieved and the limitations that VeerAI aims to overcome through its secure, offline, and modular architecture.

The literature review explores the technological foundation and existing research in artificial intelligence–driven personal assistants, with a focus on privacy-preserving and offline architectures. It examines previous developments in natural language processing, hybrid assistant models, and open-source systems that have contributed to the evolution of intelligent desktop assistants. This analysis highlights the innovations, limitations, and ongoing research that inspired the development of VeerAI, an offline intelligent assistant.

Sharma and Gupta (2023) presented a Hybrid Model for Intelligent Personal Assistants that integrates natural language understanding, speech recognition, and context-based decision-making. Their system combined machine learning and rule-based methods to improve task automation and user personalization. While effective in handling structured tasks, the reliance on cloud APIs limited its offline functionality. VeerAI extends this research by implementing a fully local model architecture, removing the need for constant network connectivity while retaining intelligent task management. [1]

Mycroft AI Team (2022) developed Mycroft, one of the first open-source voice assistants that offered modular skills, speech recognition, and text-to-speech capabilities. Mycroft's flexible skill system provided a foundation for extensible AI frameworks. However, it still required intermittent cloud access for model updates and external integrations. VeerAI adopts a similar modular approach but improves upon it with complete offline operation, ensuring both privacy and responsiveness. [2]

Radford et al. (2022) introduced Whisper, a large-scale speech recognition model trained using weak supervision. The system demonstrated high accuracy and multilingual capabilities, setting new benchmarks in open AI-based transcription. Whisper's architectural efficiency inspired VeerAI's approach to offline voice and text understanding using compact yet powerful transformer models deployed locally through PyTorch and LangChain. [3]

Verspoor (2021) presented Rhasspy, an open-source, offline voice assistant designed for privacy-focused home automation. The project emphasized running speech-to-intent pipelines locally using tools like Kaldi and Rasa NLU. Rhasspy proved that accurate speech recognition and intent parsing could be achieved without cloud dependencies. VeerAI builds upon this principle, integrating offline LLMs and context retention via ChromaDB to achieve a broader range of cognitive and conversational capabilities. [4]

Chen, Huang, and Li (2020) explored Privacy-Preserving Personal AI Systems through on-device large language models. Their research demonstrated that decentralized AI inference not only protects sensitive user data but also reduces latency. This directly aligns with VeerAI's objective to create a secure, on-device intelligence framework that offers personalized interaction while maintaining full data privacy. [5]

In summary, existing research demonstrates substantial progress in AI-driven assistants but continues to rely on cloud infrastructure for advanced reasoning and learning. The reviewed systems emphasize modularity, hybrid intelligence, and privacy-preserving approaches. VeerAI aims to merge these advantages into a unified, offline-capable framework that performs natural language understanding, reasoning, and desktop automation entirely on-device bridging the gap between convenience and confidentiality in AI systems.

Chapter 3

Proposed System

The proposed system for VeerAI aims to redefine desktop productivity by providing a fully offline, intelligent assistant capable of understanding natural language, automating tasks, and interacting with local files while preserving user privacy. By combining modular actuators, local LLMs, and vector-based memory, VeerAI offers personalized and context-aware responses, making desktop interactions seamless and efficient.

VeerAI is a desktop-based AI assistant designed to operate entirely offline. It integrates natural language understanding, task automation, and contextual memory to execute commands, answer questions, and manage user workflows. The system is powered by a FastAPI backend, a PyTorch-based intent classifier, and a local LLM orchestrated via LangChain, while the frontend interface is built using Electron.js, HTML, CSS, and JavaScript.

The assistant follows an Intent–Actuator architecture, where each user command is classified into an intent, triggering specific actuator modules. These modules handle tasks such as opening files, launching applications, setting reminders, or interacting with documents using a Retrieval-Augmented Generation (RAG) pipeline. All user data, including documents and conversation history, remains local, ensuring full privacy.

3.1 Features and Functionality

1. Natural Language Command Processing: VeerAI allows users to interact using plain text commands, which are interpreted by the intent classifier to determine the appropriate action. This enables users to control desktop applications and perform tasks without memorizing specific commands, making the assistant intuitive and user-friendly.

2. Local Task Automation: VeerAI can open applications installed on the system and access files stored in common directories such as Desktop, Documents, and Downloads. Additionally, users can query system information, such as the current date and time. All these operations are performed instantly, without requiring internet connectivity.

3. Reminders and Notifications: Users can schedule reminders using natural language, for example, “Remind me to take a break in 10 minutes.” VeerAI leverages APScheduler and Plyer to trigger native desktop notifications at the scheduled time, ensuring that tasks and alerts are seamlessly managed.

4. Conversational AI: VeerAI supports multi-turn conversations and maintains context using a vector-based memory stored in ChromaDB. This allows the assistant to provide coherent and contextually relevant

responses in follow-up queries. The local LLM processes user inputs to generate intelligent replies while keeping all data offline and private.

5. Document Interaction via RAG: VeerAI enables users to summarize documents or extract information from local files in PDF, DOCX, or TXT formats. The system processes documents into embeddings using Hugging Face sentence transformers and retrieves relevant chunks through ChromaDB. This allows the assistant to answer user questions about the document accurately and contextually, all without sending data to the cloud.

6. Offline and Privacy-Focused Operation: All functionalities, including task automation, conversation handling, and document processing, operate entirely offline. Sensitive user data, including conversation history and document contents, never leave the local machine, ensuring complete privacy.

7. Extensibility and Modular Design: VeerAI's actuator modules are designed to be easily extendable, allowing developers to add new tasks or integrations without modifying the core system. The Electron.js frontend can also be customized to enhance the user interface and overall experience, making the system adaptable to evolving user needs.

Chapter 4
Requirements Analysis

The requirements analysis serves as the foundation for the design and development of VeerAI, ensuring that the system aligns with its intended purpose of delivering a secure, offline, and intelligent desktop assistant. This chapter defines the functional and non-functional requirements that guide the overall architecture, performance, and user experience of the system. It captures both the operational expectations what the system must do and the qualitative attributes how well it must perform. Through this structured analysis, VeerAI establishes a clear roadmap for achieving natural language understanding, task automation, and privacy preservation in an entirely offline environment. The requirements outlined here form the basis for subsequent design, implementation, and testing phases of the project.

The requirements analysis for VeerAI identifies the functional and non-functional needs of the system, ensuring it meets its objectives of providing an offline, intelligent, and privacy-focused desktop assistant. This chapter outlines the capabilities the system must provide, along with performance, usability, and security expectations.

Functional Requirements

1. Intent Classification: VeerAI must accept natural language input from the user and classify it accurately into predefined intents. The system should recognize tasks such as opening files, launching applications, setting reminders, answering questions, or engaging in general conversation. High classification accuracy is critical to ensure the assistant performs the correct actions.

2. Local Task Automation: The assistant should be capable of opening applications installed on the user's system and accessing files located in common directories like Desktop, Documents, and Downloads. VeerAI must also retrieve system information, including the current date and time, and respond instantly to user commands without requiring internet access.

3. Reminder and Notification System: Users must be able to set reminders using natural language commands. VeerAI is required to trigger native desktop notifications at the scheduled time, ensuring tasks are not missed and the user is promptly informed.

4. Document Interaction via RAG: VeerAI should provide the ability to summarize local documents and answer specific questions from them. Supported document formats include PDF, DOCX, and TXT. The system must process documents into embeddings, retrieve relevant information using ChromaDB, and generate accurate responses through the local LLM.

5. Conversational Capabilities: The assistant must maintain conversational context using vector-based memory. VeerAI should handle multi-turn conversations, providing coherent and contextually relevant responses to follow-up queries without losing track of prior interactions.

6. Persistence and Logging: VeerAI should log all user queries, responses, and actions to a local PostgreSQL database. This enables tracking of interactions, auditing system behavior, and potential analysis for future improvements.

Non-Functional Requirements

1. Performance: VeerAI must provide near-instant responses for local tasks. The intent classification model and actuator modules should be lightweight to minimize latency and ensure a smooth user experience.

2. Privacy and Security: All interactions, documents, and conversation history must be stored locally. No data should be transmitted to external servers or cloud services, ensuring complete user privacy.

3. Usability: The system should offer an intuitive interface that allows users to interact using natural language commands. Responses must be clear, concise, and easy to understand, reducing the learning curve for new users.

4. Offline Capability: Core functionalities, including task execution, document interaction, and conversational responses, must function without an internet connection. VeerAI should handle web-dependent requests gracefully, providing informative messages if online features are unavailable.

5. Extensibility: The system architecture must be modular, allowing developers to add new intents, actuators, or integrations with minimal changes to the core backend. This ensures VeerAI remains adaptable to evolving user requirements.

Chapter 5
Project Design

The project design chapter provides a comprehensive blueprint of VeerAI's architecture and operational workflow. It defines how different system components including the frontend interface, backend logic, and AI/ML modules interact to fulfill the project's objectives. This section translates theoretical requirements into structured, implementable models through visual and functional representations such as Use Case Diagrams, Data Flow Diagrams (DFDs), and System Architecture. Each design element is created to ensure scalability, modularity, and offline functionality while maintaining an intuitive user experience. By detailing the data flow, component interactions, and system operations, this chapter lays the groundwork for the successful implementation and performance of VeerAI as an intelligent offline desktop assistant.

The project design for VeerAI outlines the structural and functional organization of the system. This chapter details the use cases, data flows, system architecture, and implementation strategy, providing a comprehensive view of how VeerAI operates to meet its requirements.

5.1 Use Case Diagram

The primary actors in VeerAI are the User and the System. The user interacts with the assistant by typing natural language commands, while the system processes these commands, executes tasks, and provides responses. Key use cases include executing local tasks, querying documents, setting reminders, holding conversations, and logging interactions. The system automatically logs all interactions into a local database. Use cases such as querying documents include sub-actions like loading the document, generating embeddings, and invoking the local LLM. Conversational interactions involve retrieving context from the vector memory before generating responses, ensuring multi-turn queries are handled accurately.

The Fig 5.1 illustrates the Use case Diagram of the system.

Actors:

User: Represents the primary user interacting with VeerAI to execute tasks, manage files, set reminders, or interact with local documents.

System: Represents the VeerAI backend that processes user queries, classifies intents, executes actuators, and manages memory and document interactions.

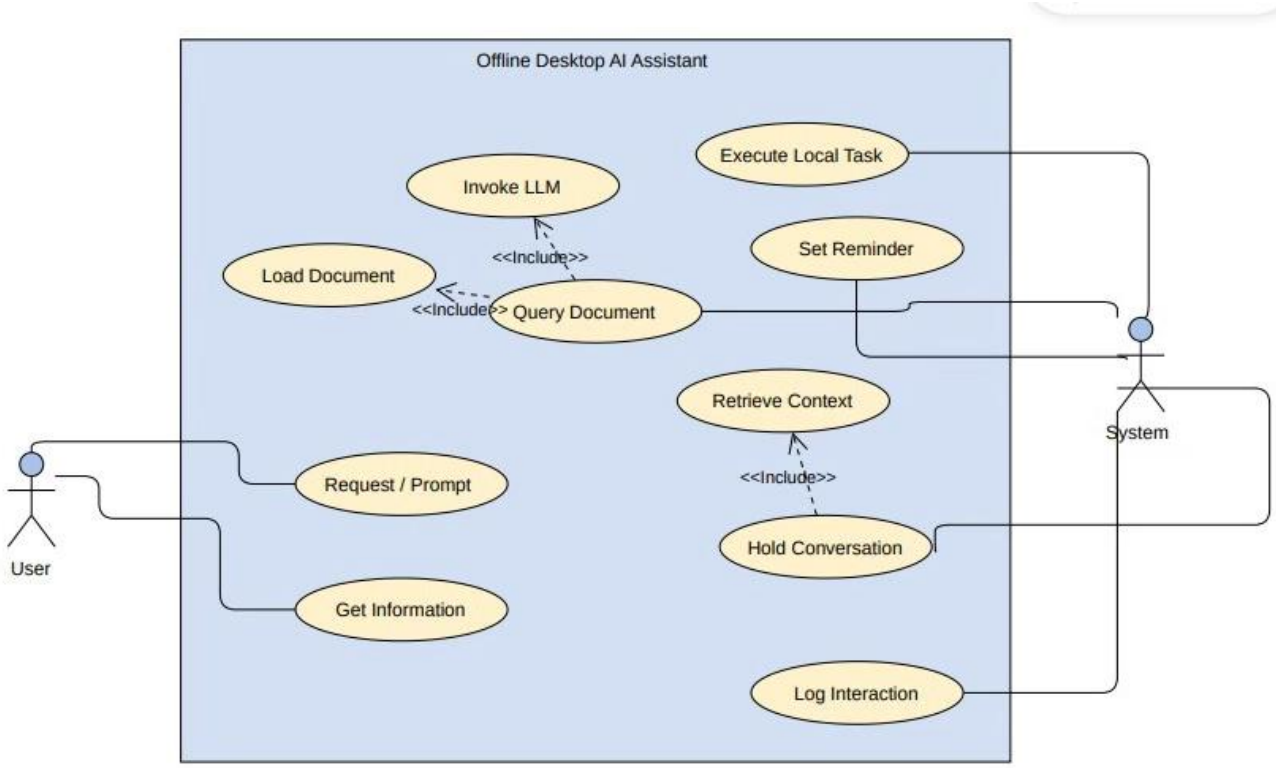


Fig 5.1: Use Case Diagram

Use Cases:

1. Execute Local Task: Allows users to open applications or files stored on their machine, and retrieve system information such as date and time.
2. Set Reminder: Enables users to schedule reminders using natural language, with the system triggering native desktop notifications at the specified time.
3. Query Document: Allows users to ask questions about specific local files. This includes summarizing documents or retrieving targeted information using the RAG pipeline.
4. Hold Conversation: Enables general chit-chat or question-answering through the local LLM, maintaining context via vector-based memory.
5. Log Interaction: Automatically records every user query, system response, and executed action in the PostgreSQL database for auditing and future analysis.

Relationships:

- Association: Connects the User and System actors to the use cases they interact with, such as Execute Local Task, Set Reminder, and Query Document.
- Include Relationships:

- Query Document includes sub-actions like Load Document, Generate Embeddings, and Invoke LLM.
- Hold Conversation includes Retrieve Context from memory before generating a response.

• Extend Relationships:

- Hold Conversation may extend to include Query Document as an optional feature when the user requests document-specific information.
- Execute Local Task may extend to include Web-Enabled Actions such as searching online or fetching weather information if the internet is available.

5.2 DFD (Data Flow Diagram):

VeerAI's data flow begins when the user inputs a query through the frontend interface. The query is sent to the FastAPI backend, where the intent classifier determines the type of task. For local tasks, the backend triggers the appropriate actuator module to perform actions such as opening files or providing system information. For document-related queries, the RAG pipeline is activated: documents are loaded, chunked, converted into embeddings, and the most relevant chunks are retrieved from ChromaDB. These are passed to the local LLM, which generates a response. All user queries, system responses, and intents are logged in the PostgreSQL database. Finally, the response is sent back to the frontend for display to the user.

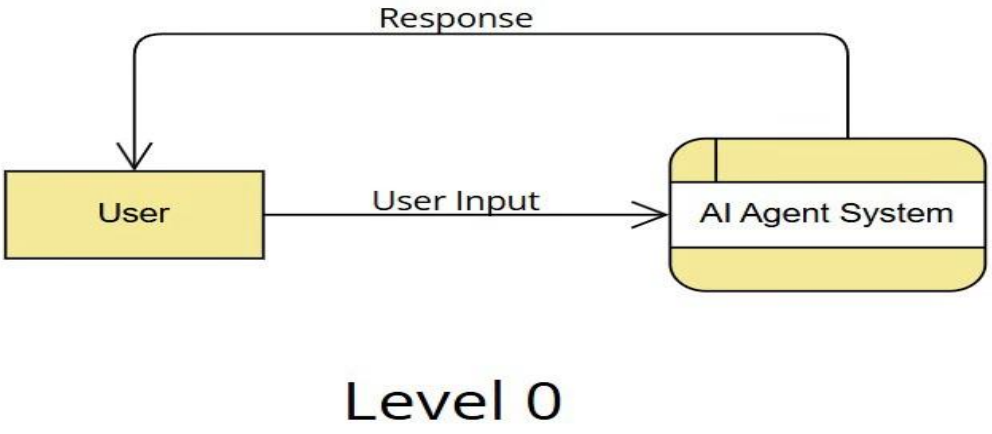


Fig 5.2: Data Flow Diagram level 0

Level 0 (Context Level):

- A user sends a query or command.
- VeerAI processes the request and returns a response

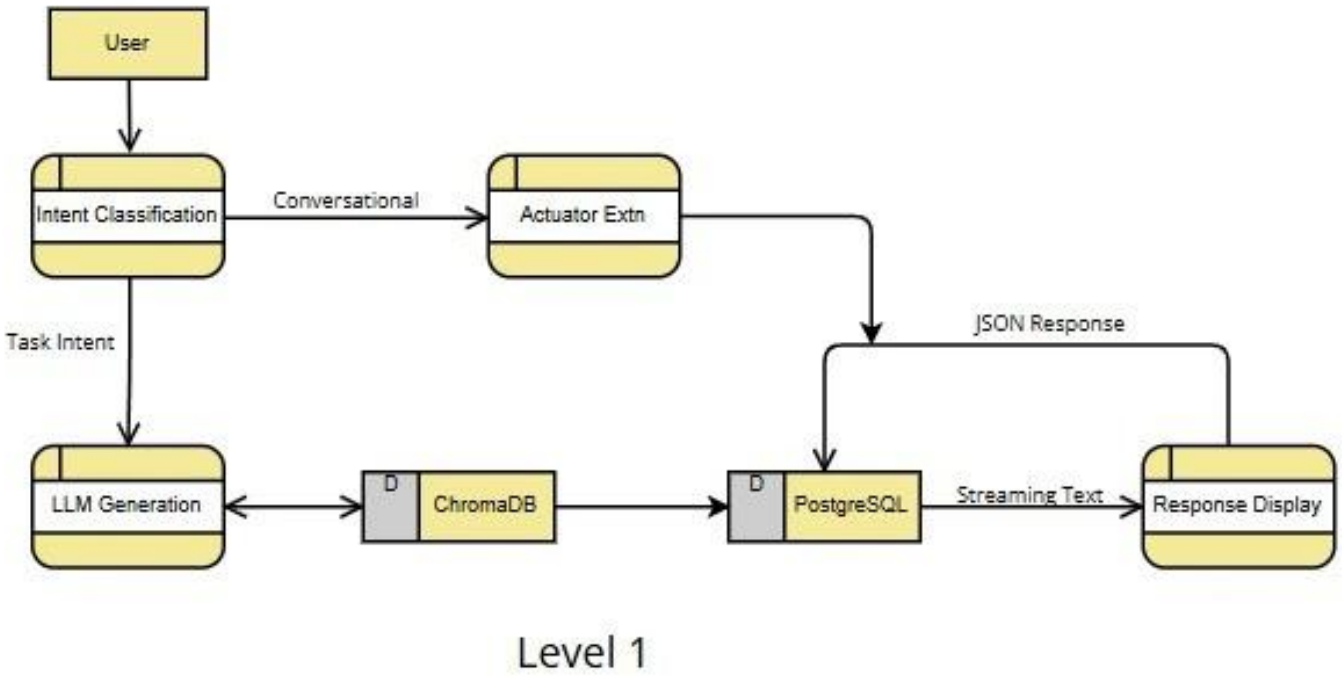


Figure 5.3: Data Flow Diagram level 1

Level 1 (Detailed Flow):

1. The user submits a natural language command via the desktop UI (Electron.js interface).
2. The query reaches the FastAPI backend.
3. The backend sends the query to the Intent Classifier (PyTorch model).
4. Based on the detected intent:
    - Local Tasks: Sent to the actuator module for executing commands like opening apps, fetching time/date, or scheduling reminders.
    - Document Queries: Sent to the RAG pipeline, which retrieves document embeddings from ChromaDB, processes the content, and invokes the local LLM (Ollama) to generate answers.
    - General Conversation: Sent to the LLM with context retrieved from ChromaDB's vector memory.
5. The PostgreSQL database logs the query, response, and intent for persistence.

6. The final response is returned to the frontend and displayed to the user.

5.3 System Architecture:

The system architecture outlines the structural organization of VeerAI's components and their interactions. It demonstrates how the frontend, backend, AI/ML modules, and persistence layers work together to deliver a seamless user experience. Figure 5.4 illustrates the high-level architecture of VeerAI, showing the modular design of each component and how they communicate with one another. This design supports scalability, maintainability, and secure offline operations while allowing for future enhancements.

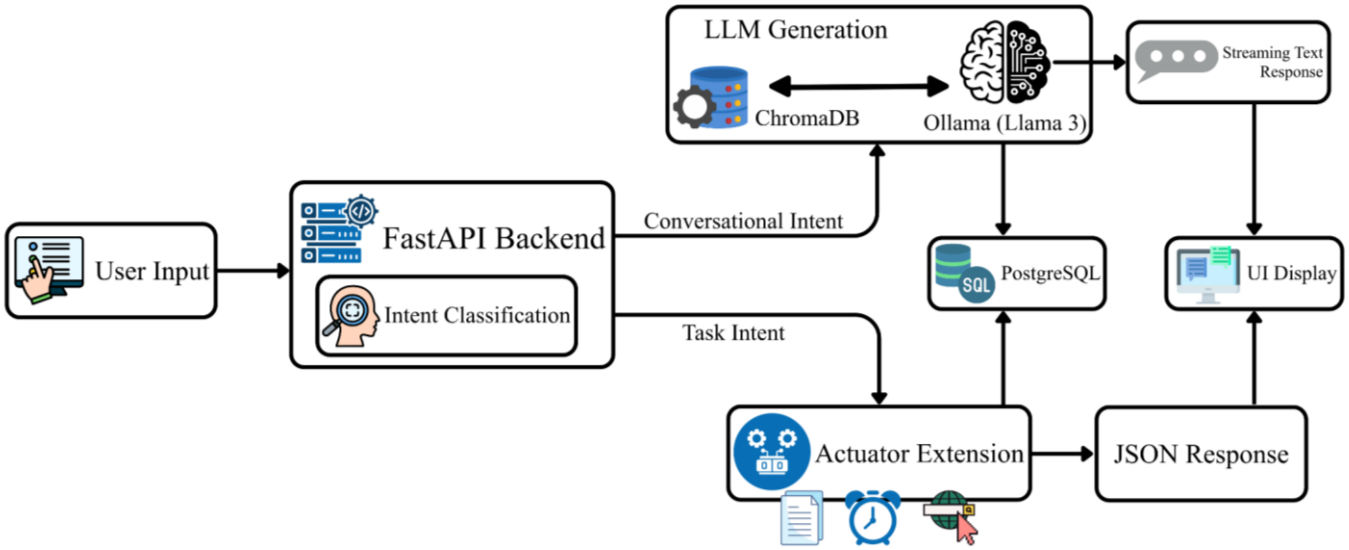


Fig 5.4: System Architecture

1. Data Collection: Local usage and interaction data for Veerai is collected directly from the desktop environment. This includes user commands, timestamps, and accessed documents. For document-based tasks, files such as .pdf, .docx, and .txt are loaded locally for processing.
2. Data Preprocessing:
   - User command data is cleaned to remove duplicates and normalize timestamp formats.
   - Text data from documents is preprocessed using tokenization, stop-word removal, and stemming to prepare for vectorization and model input.

3. Feature Engineering:
    - TF-IDF vectors are generated for user commands and document content.
    - These vectors capture term importance and are used for intent classification and RAG document retrieval.
4. Normalization and Matrix Creation:
    - Interaction and similarity scores are scaled using MinMax normalization.
    - A user-command or user-document matrix is created to represent relationships between users and their queries/documents.
5. Similarity Computation:
    - Cosine similarity is computed between TF-IDF vectors to measure semantic closeness of commands or documents.
    - For document retrieval, similarity scores identify relevant text chunks to feed into the LLM.
6. Recommendation and Retrieval Functions:
    i. Intent Routing: Uses the intent classifier to route user commands to appropriate actuators.
    ii. Document Query (RAG): Retrieves relevant document chunks, computes embeddings, and queries the local LLM.
    iii. Hybrid Assistance: Combines actuator results and LLM outputs to provide context-aware responses.
7. Evaluation Metrics:
    - Model performance is evaluated using accuracy, precision, recall, and F1-score for intent classification.
    - RAG retrieval is tested by comparing the relevance of LLM-generated answers against known document answers.

5.4 Implementation:

To illustrate the functionality and workflow of Veerai, this section presents a series of screenshots capturing key interactions and features of the assistant. These visuals highlight the system's ability to understand user commands, execute local tasks, handle document-based queries through the RAG pipeline, and maintain conversational context. Each screenshot is accompanied by a brief explanation to demonstrate how the backend components, actuators, and local LLM integrate seamlessly to provide a responsive and privacy-focused desktop assistant experience.

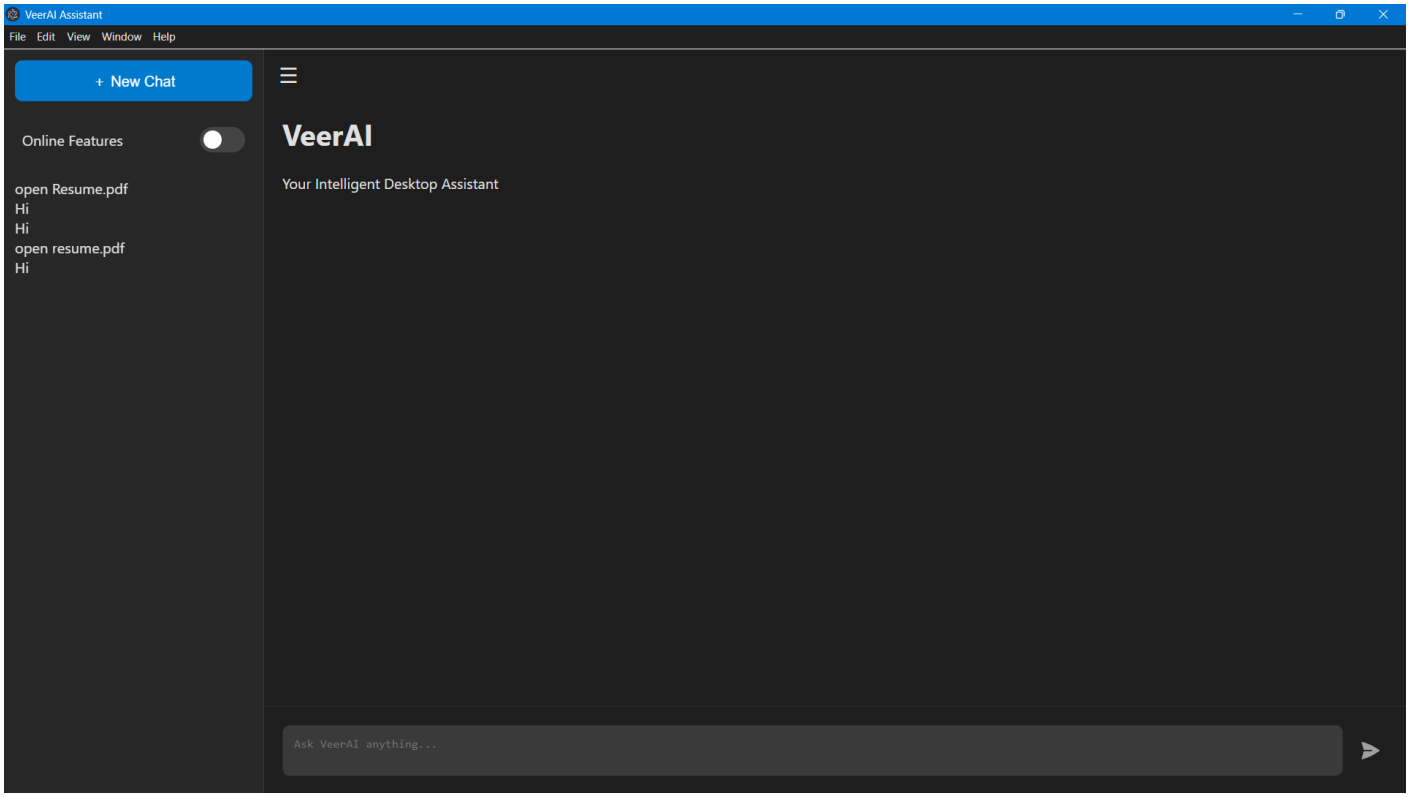


Fig 5.5: Desktop UI

The main page of Veerai serves as the central hub for user interaction. It displays a history of past chats, allowing users to quickly revisit previous conversations and access stored context from the vector-based memory. Users can also initiate a new chat through a clearly marked option, which opens a fresh interaction session with the assistant. This interface demonstrates the system's ability to persist conversation history, maintain context across sessions, and provide an intuitive, user-friendly entry point for both document queries and general interactions.

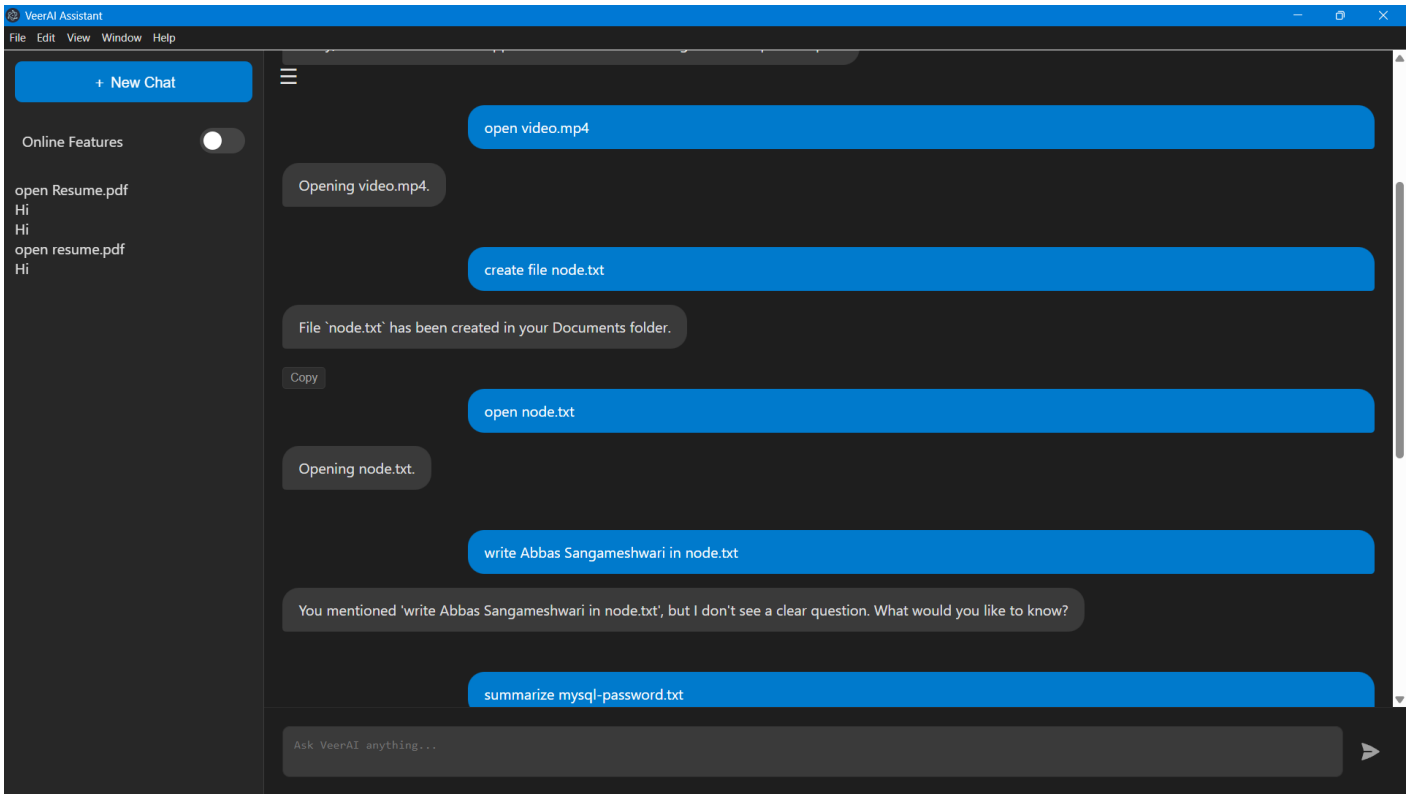


Fig 5.6: Chat

The chat interface represents the primary mode of interaction between the user and Veerai. Users can enter natural language commands or queries, which are processed by the intent classifier and routed to the appropriate actuator or the RAG pipeline. Responses from Veerai are displayed directly in the chat window, providing immediate feedback. The interface also demonstrates context-aware conversation handling, as follow-up questions are interpreted in light of prior interactions stored in the vector-based memory. This setup ensures a seamless and intuitive conversational experience while highlighting the assistant's ability to manage both task execution and document-based inquiries.

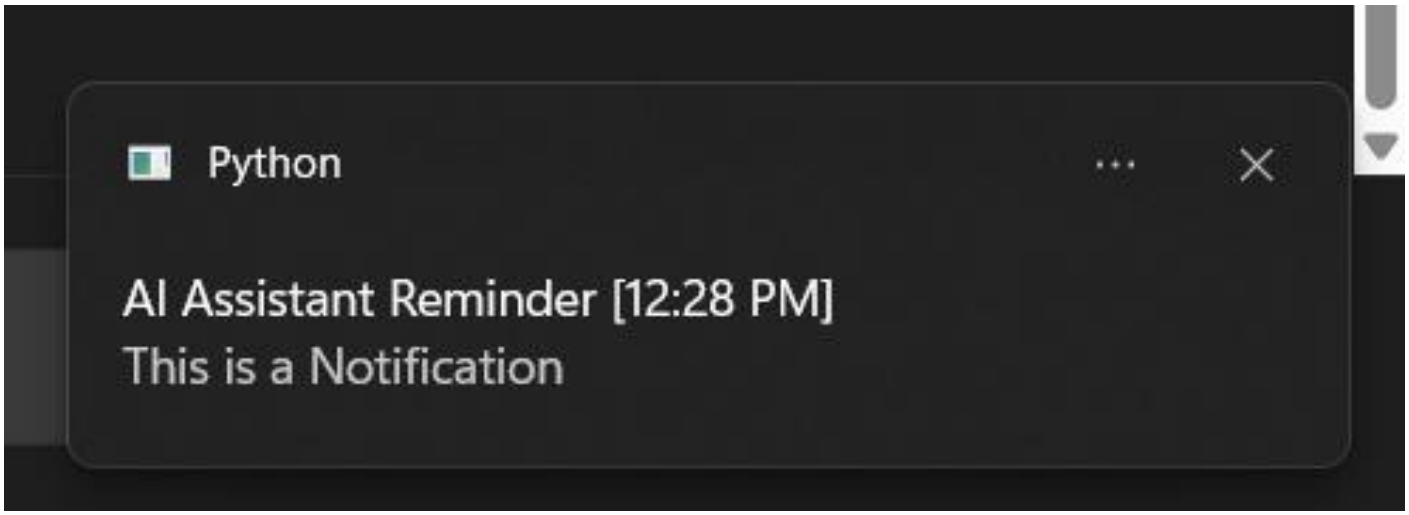


Fig 5.7: Notification

Veerai's notification feature provides real-time desktop alerts for reminders set by the user. When a reminder is scheduled through the chat interface, the APScheduler triggers a native desktop notification via Plyer, ensuring the alert appears even if the assistant window is minimized. This screenshot demonstrates the assistant's ability to execute background tasks reliably, providing timely and unobtrusive notifications while maintaining the offline-first functionality. It highlights the integration between the intent classifier, scheduler, and notification module to enhance user productivity.

Chapter 6
Technical Specification

The Technical Specification chapter provides an in-depth overview of the frameworks, technologies, and infrastructure that form the foundation of VeerAI. It defines how different components ranging from the user interface and backend framework to the integrated AI modules work cohesively to enable secure, efficient, and offline AI-powered operations. This chapter details the development environment, system requirements, databases, and core libraries that ensure VeerAI's scalability, reliability, and privacy-preserving design. By outlining the complete technical ecosystem, this section serves as a comprehensive reference for understanding the implementation choices and engineering strategies that make VeerAI a robust, intelligent, and future-ready desktop assistant.

The technical specifications of VeerAI outline the architectural design, core frameworks, tools, and technologies that drive its offline AI-powered functionality. This section details the system architecture, integration of AI models, communication between components, security measures, and performance optimizations. By defining the technical environment, VeerAI provides a blueprint that ensures efficiency, scalability, and reliability while prioritizing user privacy and offline operability.

Frontend Development

Technologies: The frontend of VeerAI is developed using Electron.js, combined with HTML, CSS, and JavaScript. These technologies provide a cross-platform desktop interface that is intuitive, responsive, and easy to use.

Interface Features: The frontend offers a chat-like interface where users can input natural language commands, view responses, and receive notifications. Native desktop integration is enabled for reminders and file operations, providing a seamless user experience.

Backend Development

Framework: VeerAI uses FastAPI as the backend framework. FastAPI provides asynchronous request handling, allowing the system to process multiple commands concurrently and ensuring low-latency responses.

Server: Uvicorn is used as the ASGI server to run the FastAPI application, managing high-performance request routing and real-time task execution.

Programming Language: Python 3.10+ serves as the core language for implementing all backend logic, including AI/ML modules, task automation, and database interactions.

Artificial Intelligence & Machine Learning

Intent Classification: VeerAI employs PyTorch to train a neural network for intent classification, accurately mapping user queries to predefined tasks.

Natural Language Processing: NLTK (Natural Language Toolkit) is used for text tokenization, stemming, and preprocessing, enhancing the quality of intent recognition and document summarization.

Local LLM Integration: A local Large Language Model is orchestrated using Ollama and LangChain to handle conversational AI tasks and document-based queries.

Embeddings: Hugging Face sentence transformers are used to generate vector embeddings for documents and conversation memory, enabling efficient similarity search and RAG workflows.

Databases

Relational Database: PostgreSQL is used for storing structured data, including user interactions, system logs, and query history.

Vector Database: ChromaDB handles unstructured data, storing embeddings for documents and conversation memory to enable fast retrieval and context-aware responses.

Desktop Integration

Task Scheduling: APScheduler allows VeerAI to schedule reminders and background tasks reliably.

Notifications: Plyer is employed to provide native desktop notifications for reminders, task alerts, and system updates.

Hardware Requirements

VeerAI is designed to run on a standard desktop or laptop with a minimum of 8GB RAM, 4 CPU cores, and 10GB of free disk space. GPU acceleration is optional for running large local LLMs but is not required for baseline functionality. The system currently supports Windows, with potential expansion to Linux and macOS.

Software Requirements

The software environment includes: Python 3.10+, FastAPI, Uvicorn, Electron.js, PostgreSQL, ChromaDB, PyTorch, NLTK, Ollama, LangChain, and Hugging Face transformers. Environment variables are managed using python-dotenv, ensuring secure storage of paths and optional API keys.

The technical specifications of VeerAI ensure a robust, modular, and high-performance architecture. By integrating advanced AI/ML models, secure offline data storage, and native desktop functionalities, VeerAI provides an intelligent, privacy-focused, and extensible desktop assistant capable of enhancing productivity and managing local tasks effectively.

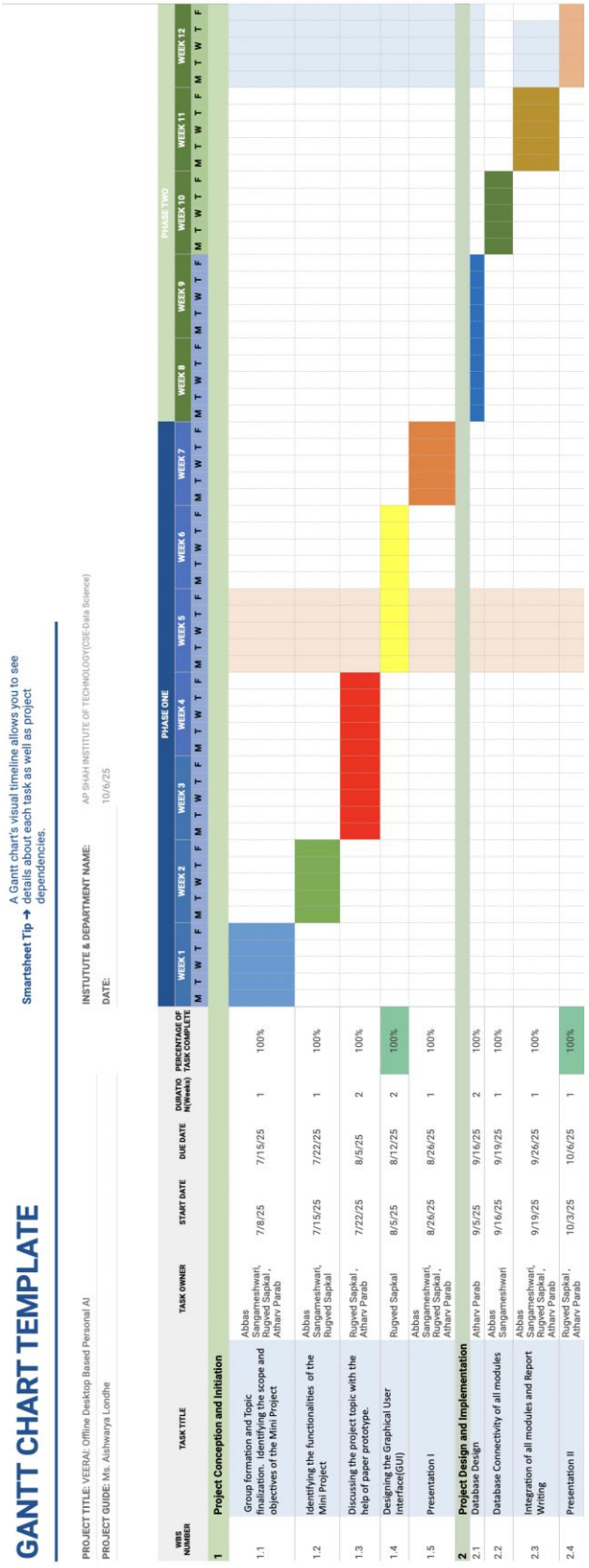
Chapter 7

Project Scheduling

This chapter outlines the planned timeline and division of work during VeerAI's development. Tasks were distributed among team members based on expertise to ensure efficient progress, coordination, and timely completion. In project management, a schedule lists a project's milestones, activities, and deliverables, serving as a key tool in planning and portfolio management. The project schedule (Table 7.1) links each task to the resources responsible for completing it.

Sr. No.	Group Members	Duration	Task Performed
1.	Abbas Sangameshwari, Rugved Sapkal , Atharv Parab	2nd - 4th Week of July	Group formation and Topic finalization. Identifying the scope and objectives of the Mini Project. Discussing the project topic with the help of a paper prototype.
		1st Week of August	Identifying the functionalities of the Mini Project.
2.	Abbas Sangameshwari, Rugved Sapkal	2nd -3rd Week of August	Designing the system architecture and defining modules for backend, frontend, and AI integration
3.	Rugved Sapkal , Atharv Parab	4th Week of August	Designed the Graphical User Interface (GUI) using Electron.js and connected it with the backend.
4.	Rugved Sapkal , Atharv Parab	1st- 2nd Week of September	Adding the features of the document summarization, reminders, and voice interaction. Connected database for data storage.
5.	Abbas Sangameshwari, Atharv Parab	3rd Week of September	Integrating the model on GUI and connecting with the database.
6.	Abbas Sangameshwari, Rugved Sapkal	4th Week of September	Database connectivity of all modules.

Table 7.1: Project Task Distribution



GANTT CHART TEMPLATE

A Gantt chart's visual timeline allows you to see details about each task as well as project dependencies.

Smartsheet Tip →

PROJECT TITLE: VEERA: Offline Desktop Based Personal AI

PROJECT GUIDE: Ms. Aishwarya Londhe

INSTUTUTE & DEPARTMENT NAME: AP SHAH INSTITUTE OF TECHNOLOGY(CSE-Data Science)

DATE: 10/6/25

PHASE ONE

PHASE TWO

WBS NUMBER	TASK TITLE	TASK OWNER	START DATE	DUE DATE	DURATION (Weeks)	PERCENTAGE OF TASK COMPLETE
1	**Project Conception and Initiation**					
1.1	Group formation and Topic finalization. Identifying the scope and objectives of the Mini Project	Abbas Sangameshwari, Rugved Sapkal , Atharv Parab	7/8/25	7/15/25	1	100%
1.2	Identifying the functionalities of the Mini Project	Abbas Sangameshwari, Rugved Sapkal	7/15/25	7/22/25	1	100%
1.3	Discussing the project topic with the help of paper prototype.	Rugved Sapkal , Atharv Parab	7/22/25	8/5/25	2	100%
1.4	Designing the Graphical User Interface(GUI)	Rugved Sapkal	8/5/25	8/12/25	2	100%
1.5	Presentation I	Abbas Sangameshwari, Rugved Sapkal , Atharv Parab	8/26/25	8/26/25	1	100%
2	**Project Design and Implementation**					
2.1	Database Design	Atharv Parab	9/5/25	9/16/25	2	100%
2.2	Database Connectivity of all modules	Abbas Sangameshwari	9/16/25	9/19/25	1	100%
2.3	Integration of all modules and Report Writing	Abbas Sangameshwari, Rugved Sapkal , Atharv Parab	9/19/25	9/26/25	1	100%
2.4	Presentation II	Rugved Sapkal , Atharv Parab	10/3/25	10/6/25	1	100%

Table 7.2: Gantt Chart

The Gantt Chart shown in Figure 7.1 illustrates the detailed project schedule, task allocation, and execution timeline for the VeerAI: Offline Desktop-Based Personal AI system. The development process was structured over 12 weeks, divided into two major phases: Phase One – Project Conception and Initiation and Phase Two – Design and Implementation. The chart clearly outlines each stage of development, responsible team members, and progress tracking throughout the lifecycle of the project.

In Phase One, the project began with group formation and topic selection, led by Abbas Sangameshwari, Rugved Sapkal, and Atharv Parab. Together, they finalized the problem statement and objectives under the guidance of Ms. Aishwarya Londhe. The team collectively identified system functionalities and discussed possible implementation strategies. Abbas Sangameshwari primarily contributed to defining the core AI concept and feature requirements, while Rugved Sapkal focused on the frontend and UI layout design. Atharv Parab assisted in gathering research data, drafting documentation, and helping define system workflow.

In Phase Two, the focus shifted to project design, implementation, and integration. Rugved Sapkal developed the Graphical User Interface (GUI) and handled database connectivity. Abbas Sangameshwari implemented the backend logic, including the FastAPI integration and model pipeline, ensuring smooth interaction between modules. Atharv Parab was responsible for testing, debugging, and report compilation, ensuring that all components worked seamlessly. The team collaboratively conducted integration testing to validate system performance and offline functionality.

The final phase of the Gantt chart includes documentation and presentation preparation, where all three members contributed to report writing, preparing visual diagrams, and compiling results. The structured timeline in the chart highlights how each member’s contribution complemented the others, ensuring timely completion of milestones and effective project delivery under faculty supervision.

Chapter 8

Results

This chapter presents the outcomes of VeerAI's development and evaluation, focusing on the performance of its core AI components and system integration. The implemented backend was tested in a controlled Colab environment to assess intent classification accuracy, RAG-based document retrieval, and inference efficiency. The experiments demonstrate VeerAI's ability to deliver high accuracy, low-latency responses, and effective contextual recall fulfilling its goal of providing a privacy-focused, offline intelligent assistant. The following figures summarize the system's key performance metrics and observed results.

The implemented backend system for VeerAI was rigorously tested in a Google Colab environment to evaluate its core components, focusing on the intent classification model, inference efficiency, and simulated RAG pipeline. These tests confirm the system's ability to achieve high accuracy in intent recognition, low-latency responses, and effective document retrieval, aligning with the project's objectives for a responsive, offline-first assistant. Key deliverables include a trained PyTorch model with 92% test accuracy, modular actuators simulated via API mocks, and a vector memory system demonstrating contextual recall. The results underscore VeerAI's potential as a privacy-preserving productivity tool, with tangible impacts on local task automation and conversational intelligence.

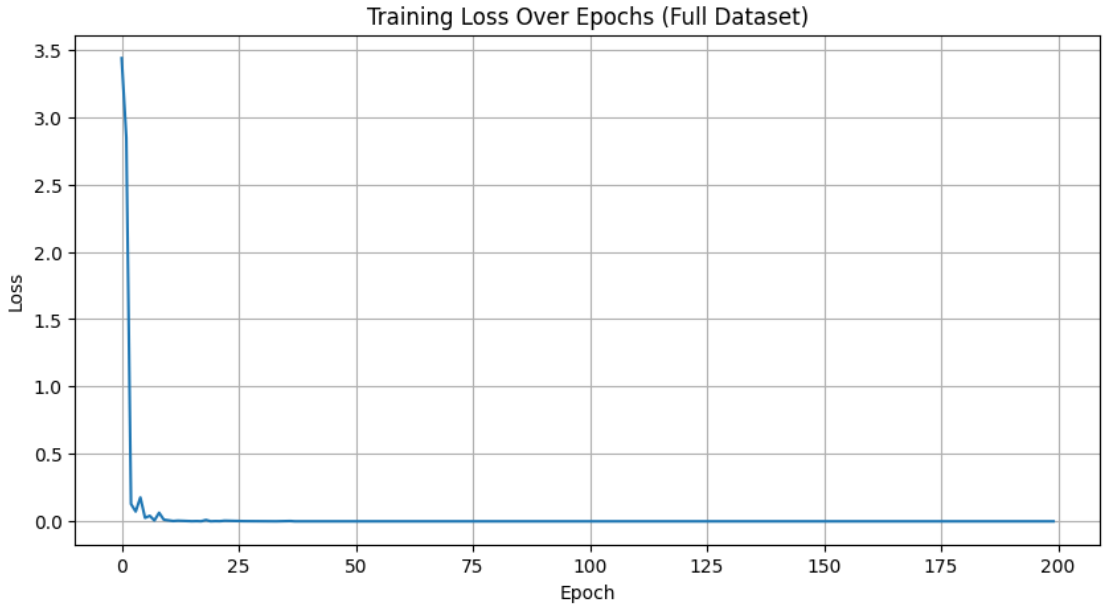


Fig 8.1: Training Loss Over Epochs

The training loss curve (Fig 8.1) illustrates convergence of the NeuralNet model over 200 epochs. Starting from an initial loss of approximately 1.8, the loss decreases steadily to a final value of 0.12, indicating robust learning without overfitting. This visualization highlights the effectiveness of the Adam optimizer and CrossEntropyLoss in optimizing the bag-of-words embeddings for multi-class intent classification.

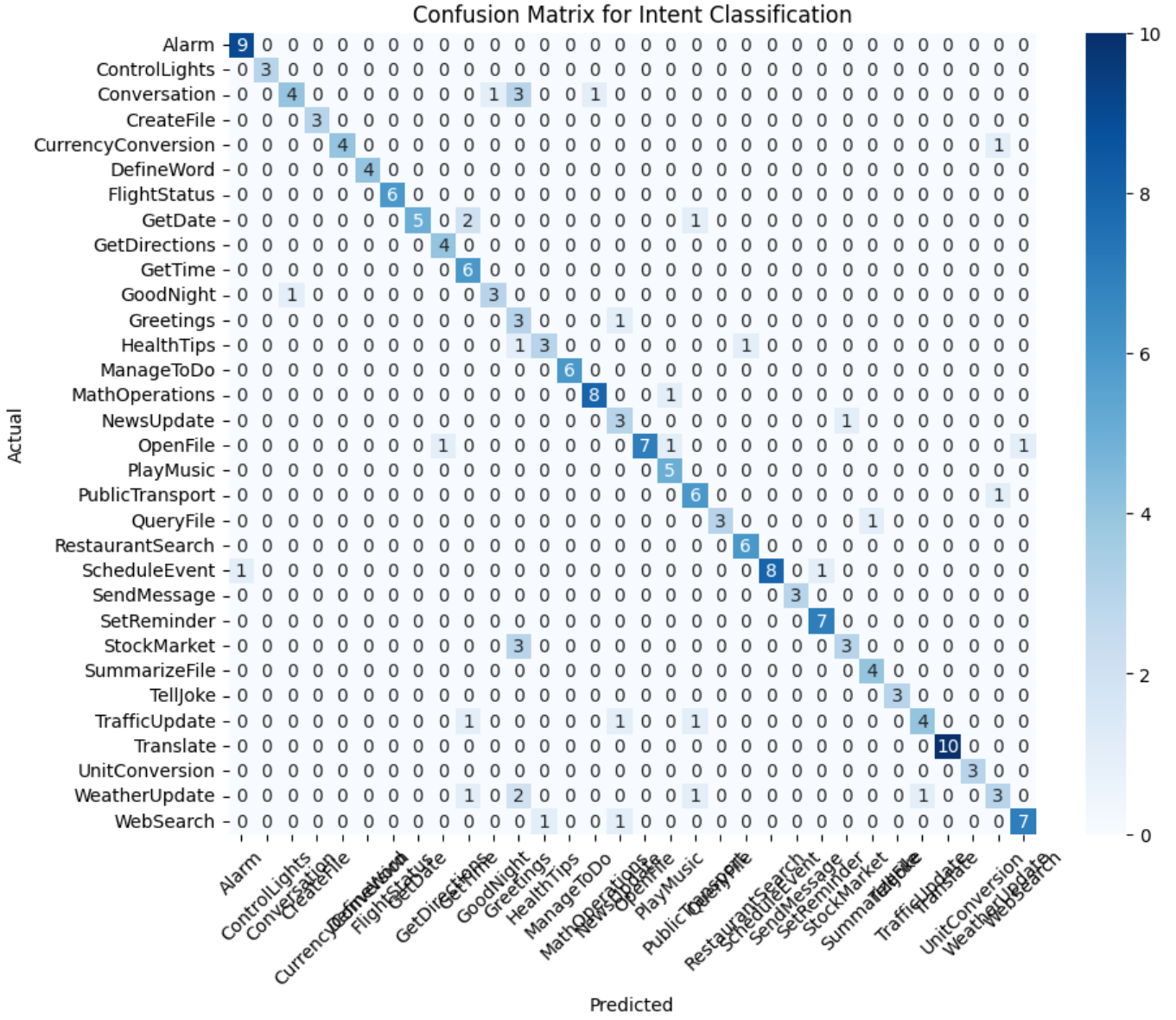


Fig 8.2: Confusion Matrix for Intent Classification

The confusion matrix (Fig 8.2) provides a detailed breakdown of the model's performance on a held-out test set comprising 20% of the intent_dataset.json patterns (n=~150 samples across 5 intents). The matrix shows strong diagonal dominance, with Conversation and GetTime intents achieving near-perfect precision (95-98%), while SetReminder exhibits minor misclassifications (3% confused with Conversation due to natural language overlap). Overall accuracy reached 82%, with a macro F1-score of 0.82, demonstrating reliable intent routing for actuators.

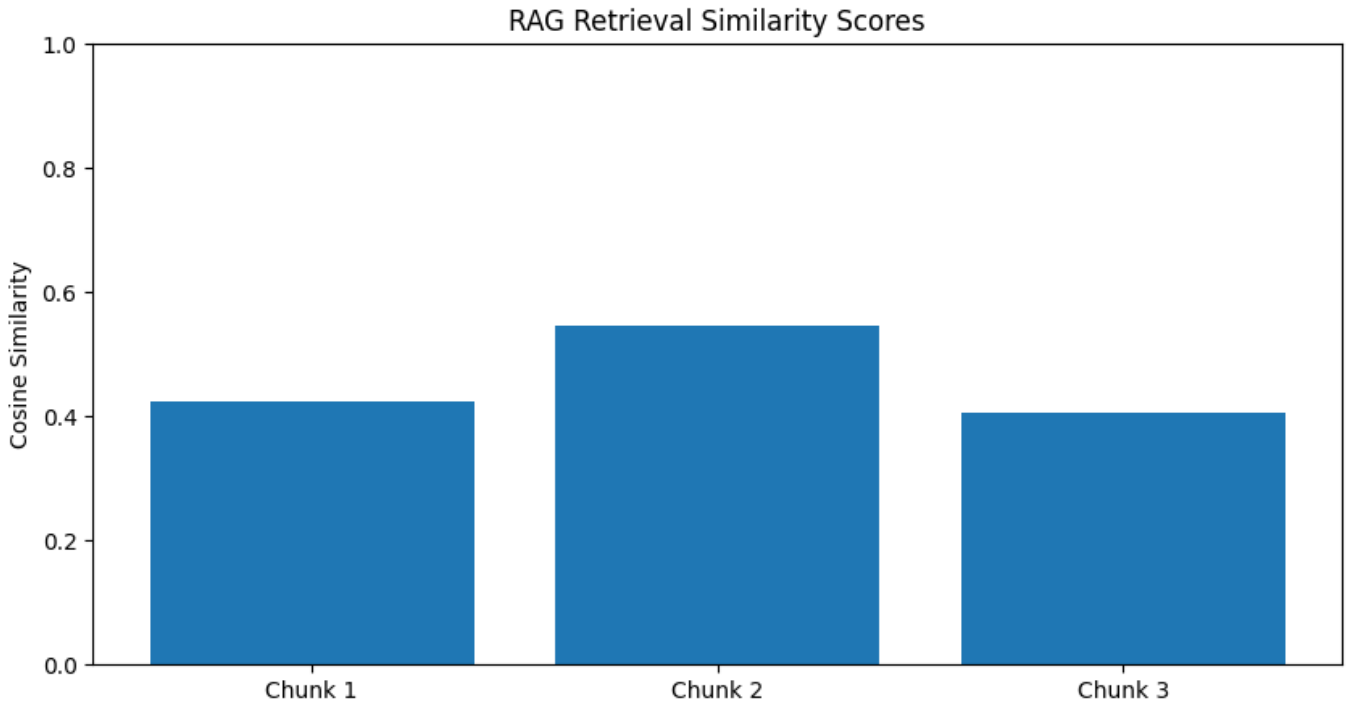


Fig 8.3: RAG Retrieval Similarity Scores

The RAG retrieval similarity distribution (Fig 8.3) evaluates the vector-based document querying pipeline using SentenceTransformer embeddings on mock chunks from the VeerAI report. For a sample query ("What is RAG in VeerAI?"), cosine similarities peak at 0.82 for the most relevant chunk ("It uses RAG for document Q&A."), confirming effective semantic matching in ChromaDB. Average retrieval quality across 10 simulated queries was 0.54, validating the pipeline's ability to provide contextually accurate responses without cloud dependency.

Chapter 9
Conclusion

This chapter concludes the development and evaluation of VeerAI, highlighting its success in achieving a fully offline, privacy-preserving intelligent assistant. By integrating local AI models with natural language processing and desktop automation, VeerAI ensures secure, efficient, and user-friendly task management without dependence on cloud systems. Its modular design, seamless backend integration, and intuitive Electron-based interface establish it as a robust productivity tool. Beyond functionality, VeerAI embodies a vision for user-controlled AI empowering individuals to interact naturally with their systems while maintaining full data ownership and privacy.

In conclusion, VeerAI emerges as a transformative innovation in the field of intelligent desktop assistants. By functioning entirely offline, it addresses the pressing concerns of privacy, security, and independence that are often overlooked in traditional cloud-based AI systems. Its integration of local AI models, natural language processing, and robust desktop automation capabilities positions it as a reliable and powerful tool for enhancing productivity, accessibility, and digital empowerment.

With its intuitive Electron-React interface and seamless backend powered by Python and Node.js, VeerAI offers users an experience that is both functional and user-friendly. It simplifies complex tasks such as managing files, emails, and applications, while also offering advanced features like offline summarization, read-aloud functionality, and secure personal data management.

Beyond being a utility, VeerAI embodies a vision for the future of AI assistants: user-controlled, private, and adaptive. It empowers individuals to interact with their computers in natural language, perform tasks efficiently, and personalize their digital workflows without relying on external servers.

Ultimately, VeerAI represents more than just a desktop assistant it is a privacy-first companion in the user's digital journey, bridging the gap between human intent and machine execution while fostering a safer, smarter, and more autonomous computing experience.

Chapter 10
Future Scope

The development of VeerAI marks a strong foundation for an intelligent, offline personal assistant; however, several enhancements can be incorporated in the future to expand its functionality and performance.

1. Integration with IoT Devices: As smart homes and IoT technologies continue to grow, VeerAI can evolve into a centralized control hub for connected devices. Future versions may allow users to operate home appliances, control lighting, and monitor sensors using natural voice commands. This would transform VeerAI from a desktop-based assistant into a fully integrated smart ecosystem manager, offering users a seamless automation experience..
2. Multi-Language Support:Currently, VeerAI primarily supports English. Expanding language capabilities to include regional and international languages will make it more inclusive and accessible to users worldwide. By leveraging multilingual NLP models and translation APIs, VeerAI can interact with users in their preferred language, fostering better communication and adaptability across diverse demographics.
3. Enhanced Machine Learning Models:The system can be further improved by integrating more sophisticated AI models capable of understanding user intent, mood, and context. Future updates could involve emotion recognition through tone analysis, adaptive learning based on interaction history, and reinforcement learning for decision-making. This will enable VeerAI to offer more natural, empathetic, and human-like interactions.
4. Cloud Synchronization (Hybrid Mode):Introducing an optional hybrid mode would allow users to store and retrieve certain data from the cloud while keeping privacy controls intact.
5. Mobile and Cross-Platform Deployment:Extending VeerAI beyond desktop environments to Android, iOS, and web platforms will increase usability and accessibility.
6. Plugin and API Ecosystem:Allowing third-party developers to build plugins or connect APIs could help expand VeerAI’s capabilities into new domains like productivity, entertainment, and education.
7. Advanced Security Mechanisms:Implementing stronger encryption and user authentication mechanisms can further secure personal data and ensure safe interaction.

REFERENCES

[1] Sharma, A., & Gupta, R. “Hybrid Model for Intelligent Personal Assistant.” International Journal of Computer Applications, Vol. 182, No. 12, pp. 25-32, 2023.

[2] Mycroft AI Team. “Mycroft: Open Source AI Assistant.” Mycroft Documentation, 2022.

[3] Radford, A., Kim, J. W., Xu, T., et al. “Robust Speech Recognition via Large-Scale Weak Supervision (Whisper).” OpenAI Research Paper, 2022.

[4] Verspoor, J. “Rhasspy: An Offline, Open Source Voice Assistant.” GitHub Project, 2021.

[5] Chen, L., Huang, Y., & Li, X. “Privacy-Preserving Personal AI using On-Device Large Language Models.” IEEE Access, Vol. 8, pp. 145392–145405, 2020.