Semester Long Internship Spring 2026

On

AI Chatbot Integration in eSim

Submitted by

POOJA M

SRM Institute of Science and Technology

Under the guidance of

Prof. Prabhu Ramachandran

Principal Investigator
Department of Aerospace Engineering
Indian Institute of Technology Bombay

May 10, 2026

Abstract

This report details the review, testing, and enhancement of eSim Copilot, an intelligent AI assistant integrated within the eSim Electronic Design Automation environment. The project addresses critical usability challenges in circuit design workflows—particularly the steep learning curve for Ngspice simulations, cryptic netlist debugging, and fragmented access to documentation—by extending and improving a multimodal, context-aware help system that operates entirely offline to ensure privacy and accessibility.

The work involved identifying and resolving bugs in the existing implementation, followed by significant feature enhancements. The assistant is built upon three core technological pillars: a Retrieval-Augmented Generation (RAG) system that indexes official eSim manuals using ChromaDB and nomic-embed-text embeddings, a computer vision module integrating MiniCPM-V for automated analysis of circuit schematic images, and a fact-based netlist analyzer that performs static and semantic validation of SPICE files to detect errors like floating nodes, missing models, and simulation conflicts.

Key contributions include a ChatGPT-style persistent chat history system with multi-session sidebar, bubble-style UI redesign, dynamic model selection supporting qwen2.5:3b, qwen2.5-coder:3b, and tinyllama:1.1b, voice input integration, copy response functionality, and a seamless installation guide for Ubuntu 22.04. The system leverages local Large Language Models via Ollama, eliminating cloud dependencies. This work significantly enhances the usability and robustness of the eSim Copilot, advancing the FOSSEE mission of making powerful EDA tools more accessible and effective for students and engineers.

Keywords: eSim, AI Assistant, Retrieval-Augmented Generation (RAG), Computer Vision, Offline LLM, Circuit Debugging, Ngspice, PyQt5 Integration, FOSSEE, Ollama, SPICE Analysis, Chat History, Multi-Session, Voice Input.

Acknowledgements

I express my sincere gratitude to **Prof. Prabhu Ramachandran** for providing me with the opportunity to be part of the FOSSEE internship program and for his continued efforts in promoting open-source engineering tool development. His leadership and vision have been instrumental in fostering meaningful student participation in the open-source ecosystem.

I also acknowledge **Prof. Kannan M. Moudgalya** for his foundational role in establishing and strengthening the FOSSEE initiative. His contributions to open-source education and the development of the FOSSEE fellowship framework have been pivotal in creating the academic and organisational platform through which this internship was undertaken.

First and foremost, I am profoundly indebted to my guide, **Mr. Sumanto Kar**, eSim Project Lead at FOSSEE, IIT Bombay. His exceptional technical mentorship, insightful feedback during weekly reviews, and patient guidance through the complexities of the eSim codebase were instrumental in shaping this project. His vision for making eSim more intelligent and accessible provided the foundational direction for this work.

I extend my sincere appreciation to my internal mentors, **Mr. Varad Patil** and **Ms. Shanthi Priya**, for their coordination, technical inputs, and constructive feedback during the internship. Their support played an important role in maintaining clarity, direction, and steady progress in the execution of the work.

POOJA M
SRM Institute of Science and Technology
May 2026

Contents

Chapter 1

Introduction

1.1 Background and Motivation

Recent advancements in Artificial Intelligence (AI), particularly Large Language Models (LLMs), have enabled the development of intelligent conversational systems capable of understanding and generating human-like responses. These technologies are increasingly being integrated into domain-specific applications to improve usability, accessibility, and user productivity.

In the field of Electronic Design Automation (EDA), users often face challenges while working with circuit simulations, SPICE netlists, component configurations, and debugging workflows. Beginners and students especially struggle with understanding NgSpice errors, interpreting simulation outputs, and navigating extensive technical documentation. Most existing workflows require users to switch between the application, manuals, forums, and external resources, which interrupts the design process and increases the learning curve.

The FOSSEE (Free and Open Source Software for Education) project at IIT Bombay develops and promotes open-source engineering tools for educational institutions across India. One of its flagship EDA tools is eSim, an open-source platform for schematic design, circuit simulation, and PCB design. While eSim provides a powerful environment for electronics design, it previously lacked an integrated intelligent assistant capable of guiding users directly within the application.

This project focuses on reviewing, testing, debugging, and enhancing the eSim Copilot system by improving its usability, stability, and interaction workflows. The work extends the existing AI assistant with additional features such as persistent chat history, dynamic model selection, voice interaction, improved user interface components, and better offline AI integration using local Large Language Models through Ollama.

1.2 Introduction to eSim

eSim is a free and open-source Electronic Design Automation (EDA) software developed under the FOSSEE project at IIT Bombay. It provides an integrated platform for circuit schematic design, SPICE simulation, waveform analysis, and PCB design. The software combines multiple open-source tools within a unified Python and PyQt5-based graphical environment.

The major functionalities provided by eSim include:

- Schematic capture and component placement using KiCad
- Netlist generation and simulation using NgSpice
- Support for analog, digital, and mixed-signal circuit analysis
- Waveform visualization and post-processing tools
- PCB design integration
- Cross-platform support for Linux, Windows, and macOS

eSim is widely adopted in educational institutions because it allows students to learn circuit design and simulation without relying on expensive proprietary software. However, many users encounter difficulties while debugging simulations, understanding SPICE netlists, and configuring components correctly. These challenges motivated the development of an intelligent AI-based assistant integrated directly into the eSim environment.

1.3 Need for an Intelligent eSim Assistant

Although eSim provides extensive simulation and design capabilities, the absence of built-in intelligent guidance creates a steep learning curve for new users. Circuit debugging often requires manually inspecting SPICE netlists, interpreting technical error messages, and searching through documentation or online forums for solutions. This process can be time-consuming and difficult, especially for beginners.

Some major challenges faced by users include:

- Difficulty understanding NgSpice simulation errors
- Lack of context-aware assistance during debugging
- Manual inspection of generated SPICE netlists
- No integrated support for analyzing uploaded schematic images
- Frequent workflow interruptions caused by switching between applications and documentation
- Limited accessibility for students unfamiliar with EDA tools and simulation workflows

To address these challenges, the eSim Copilot was developed as an offline AI-powered assistant capable of providing real-time assistance directly within the application. The chatbot integrates Retrieval-Augmented Generation (RAG), computer vision, voice interaction, and netlist analysis to create a more interactive and educational design experience.

The enhanced system developed during this project further improves the usability and robustness of the chatbot through multi-session chat history, voice input and output support, dynamic model selection, improved UI rendering, and enhanced system stability across Linux-based development environments.

1.4 Project Objectives

The primary objective of this project was to enhance and stabilize the eSim Copilot system while improving its usability, accessibility, and interaction workflow for students and engineers. The major objectives of the project are as follows:

1. To integrate and improve a native AI chatbot panel within the eSim PyQt5 application.
2. To enable fully offline AI inference using Ollama without external data transmission.
3. To implement Retrieval-Augmented Generation (RAG) for answering user queries using eSim documentation and manuals.
4. To support schematic image analysis using vision-capable AI models.
5. To implement SPICE netlist analysis and provide plain-language debugging assistance.
6. To develop persistent multi-session chat history with sidebar-based session management.
7. To integrate voice input functionality for interactive communication.
8. To implement voice output responses for improved accessibility and user interaction.
9. To support dynamic model selection for multiple Ollama-based LLMs.
10. To redesign and improve the chatbot user interface using bubble-style message rendering and enhanced interaction workflows.
11. To identify, document, and resolve runtime issues, dependency conflicts, and startup crashes encountered during development and testing.
12. To prepare a simplified installation and setup workflow for new users using Ubuntu WSL, Python virtual environments, Ollama, and required AI dependencies.

1.5 Development Methodology

The project was carried out in multiple stages involving environment setup, feature enhancement, debugging, integration, and testing within the eSim environment. The development methodology followed during the project is summarized below:

Phase	**Description**
Environment Setup	Installation and configuration of Ubuntu WSL, Python virtual environment, Ollama, and required AI models.
Codebase Review	Understanding the existing eSim Copilot architecture and workflow.
Feature Enhancement	Implementation of multi-session history, voice interaction, model selection, and UI improvements.
AI Integration	Integration of local LLMs and vision models using Ollama.
Debugging and Testing	Resolving dependency conflicts, Qt plugin issues, rendering problems, and runtime crashes.
Workflow Validation	End-to-end testing of chatbot interaction, image analysis, voice communication, and netlist debugging.
Documentation	Preparation of installation guides, feature documentation, and project report.

Chapter 2

Literature Survey

2.1 Large Language Models (LLMs)

Large Language Models (LLMs) are advanced Artificial Intelligence systems trained on massive collections of textual data to understand and generate human-like language. These models use deep learning architectures, particularly Transformer-based neural networks, to perform tasks such as question answering, summarization, reasoning, code generation, and conversational interaction.

Modern LLMs have significantly improved human-computer interaction by enabling natural language communication within software applications. In technical domains such as Electronic Design Automation (EDA), LLMs can assist users in understanding simulation errors, explaining circuit concepts, generating documentation, and debugging workflows.

In this project, locally hosted LLMs were integrated into the eSim environment using Ollama. Models such as `qwen2.5:3b` and `tinyllama` were used for conversational assistance, circuit-related query handling, and contextual response generation. The use of local models ensures privacy, offline accessibility, and reduced dependency on cloud-based AI services.

2.2 Offline AI using Ollama

Ollama is a framework that enables users to run Large Language Models locally on personal systems without requiring cloud connectivity. It provides a lightweight and simplified environment for downloading, managing, and interacting with AI models directly through local hardware resources.

The use of Ollama in this project enabled complete offline AI inference for the eSim Copilot system. This approach offers several advantages, including:

- Privacy preservation since no user data is transmitted externally
- Offline accessibility without internet dependency

- Reduced latency during inference
- Flexibility in switching between multiple models
- Simplified deployment within Linux and WSL environments

Multiple Ollama-supported models were integrated and tested during development, including `qwen2.5:3b`, `qwen2.5-coder:3b`, `tinyllama`, and `minicpm-v` for vision-based tasks. Ollama served as the primary backend for conversational AI and multimodal processing within the chatbot architecture.

2.3 Vision-Language Models

Vision-Language Models (VLMs) are AI systems capable of processing both textual and visual information simultaneously. These models combine computer vision techniques with natural language understanding to interpret images and generate meaningful descriptions or responses.

In the context of EDA tools, VLMs can assist users by analysing circuit schematic images, identifying components, and explaining circuit functionality. This capability is particularly useful for students who may upload screenshots, textbook diagrams, or hand-drawn circuit schematics for clarification.

The eSim Copilot integrates MiniCPM-V, a lightweight vision-language model capable of analysing circuit images and providing descriptive explanations. The system processes uploaded images, extracts relevant information, and generates context-aware responses to help users understand circuit structures and simulation-related issues.

The integration of vision-language models improves the educational value of the chatbot by enabling multimodal interaction instead of relying solely on text-based queries.

2.4 Optical Character Recognition (OCR)

Optical Character Recognition (OCR) is a technology used to detect and extract textual information from images. OCR systems are commonly used in document digitization, image analysis, and intelligent automation workflows.

In this project, OCR functionality was used as part of the circuit image analysis pipeline. PaddleOCR was integrated to extract textual labels, component names, values, and annotations from uploaded circuit schematic images.

The OCR pipeline helps the chatbot identify important textual elements present in circuit diagrams before passing the information to the vision-language model

for further interpretation. This improves the accuracy of component detection and circuit understanding.

The integration of OCR enhances the chatbot's ability to analyse schematic screenshots and educational diagrams effectively within the eSim workflow.

2.5 PyQt5 for Scientific GUI Applications

PyQt5 is a Python binding for the Qt framework used to develop cross-platform graphical user interfaces (GUIs). It provides extensive support for windows, widgets, layouts, event handling, threading, and multimedia integration.

eSim itself is built using Python and PyQt5, making PyQt5 the ideal framework for integrating the AI assistant directly into the application. The chatbot interface developed in this project uses PyQt5 components for:

- Chat window rendering
- Sidebar-based session management
- Bubble-style message display
- Button controls and model selectors
- Image attachment handling
- Voice interaction controls

PyQt5 also supports asynchronous operations using QThread, which was useful for handling AI inference and preventing interface freezing during long-running tasks.

The use of PyQt5 enabled seamless integration of the chatbot within the existing eSim environment while maintaining consistency with the original application design.

2.6 Speech Recognition and Voice Systems

Speech recognition systems convert spoken language into textual input that can be processed by software applications. Voice-based interaction improves accessibility and provides a more natural method of communication between users and intelligent systems.

In this project, voice input functionality was implemented using the `SpeechRecognition` Python library along with `PyAudio` for microphone access. The chatbot allows users to provide spoken queries directly through the interface, which are then converted into text and processed by the AI assistant.

Additionally, voice output functionality was integrated to allow the chatbot to read generated responses aloud. This improves accessibility and enhances the interactive experience for users.

Voice interaction features are particularly beneficial for beginners and educational environments where users may prefer conversational guidance rather than manually typing technical queries.

2.7 Retrieval-Augmented Generation (RAG)

Retrieval-Augmented Generation (RAG) is an AI architecture that combines information retrieval with language generation. Instead of relying only on pre-trained knowledge stored within a language model, RAG systems retrieve relevant external documents and use them as additional context while generating responses.

The eSim Copilot uses a RAG-based knowledge retrieval system to answer user queries related to eSim workflows, NgSpice simulations, and circuit design concepts. Documentation files and manuals were converted into embeddings using the `nomic-embed-text` model and stored in a ChromaDB vector database.

When a user asks a question, the system retrieves the most relevant documentation content and provides it to the language model as contextual information. This improves response accuracy and ensures that answers remain relevant to the eSim environment.

The integration of RAG significantly improves the chatbot's capability to provide domain-specific guidance and reduces the possibility of incorrect or hallucinated responses.

Chapter 3

User Guide and Workflow

3.1 System Requirements

The eSim Copilot system requires a Linux-based development environment with support for Python, Ollama, and PyQt5-based graphical applications. The chatbot was primarily developed and tested using Ubuntu 22.04 running through Windows Subsystem for Linux (WSL).

The minimum system requirements are listed below:

Hardware Requirements

- Intel i5/Ryzen 5 processor or higher
- Minimum 8 GB RAM
- At least 15 GB free storage space
- Internet connection for initial model downloads

Software Requirements

- Ubuntu 22.04 or WSL environment
- Python 3.10 or above
- Git
- Ollama
- PyQt5
- NgSpice
- Python virtual environment support

The project uses several Python libraries including PyQt5, SpeechRecognition, PyAudio, ChromaDB, PaddleOCR, OpenCV, and LangChain-related dependencies.

3.2 Installation and Environment Setup

The chatbot was developed within a Python virtual environment to isolate project dependencies from the system installation.

Step 1: Create a Virtual Environment

```
python3 -m venv test_env
```

Step 2: Activate the Environment

```
source test_env/bin/activate
```

Step 3: Install Dependencies

```
pip install -r requirements.txt
```

Additional Linux packages required for microphone support were installed using:

```
sudo apt install python3-pyaudio
```

3.3 Installing Ollama and Required Models

Ollama was used as the local inference engine for running Large Language Models offline.

Step 1: Install Ollama

```
curl -fsSL https://ollama.com/install.sh | sh
```

Step 2: Verify Installation

```
ollama list
```

Step 3: Download Required Models

```
ollama pull qwen2.5:3b
ollama pull qwen2.5-coder:3b
ollama pull tinyllama
ollama pull minicpm-v
ollama pull nomic-embed-text
```

Step 4: Start Ollama

```
ollama serve
```

If the message:

```
address already in use
```

appears, Ollama is already running.

3.4 Launching the eSim Copilot

After configuring the environment and dependencies, the chatbot can be launched using:

```
source test_env/bin/activate

PYTHONPATH=src python3 -m frontEnd.Application
```

Once launched successfully, the chatbot window opens with the conversational interface, session sidebar, model selector, and interaction controls.

3.5 Chatbot Interface Overview

The chatbot interface was designed using PyQt5 and follows a modern conversational layout similar to popular AI chat platforms.

The major interface components include:

- Chat area for displaying conversations
- Text input field for queries
- Send button for submitting prompts
- Voice input button for microphone interaction
- Sidebar panel for session history
- Model selector for switching AI models
- Image upload support for schematic analysis

The interface was redesigned during development to improve readability, usability, and workflow efficiency.

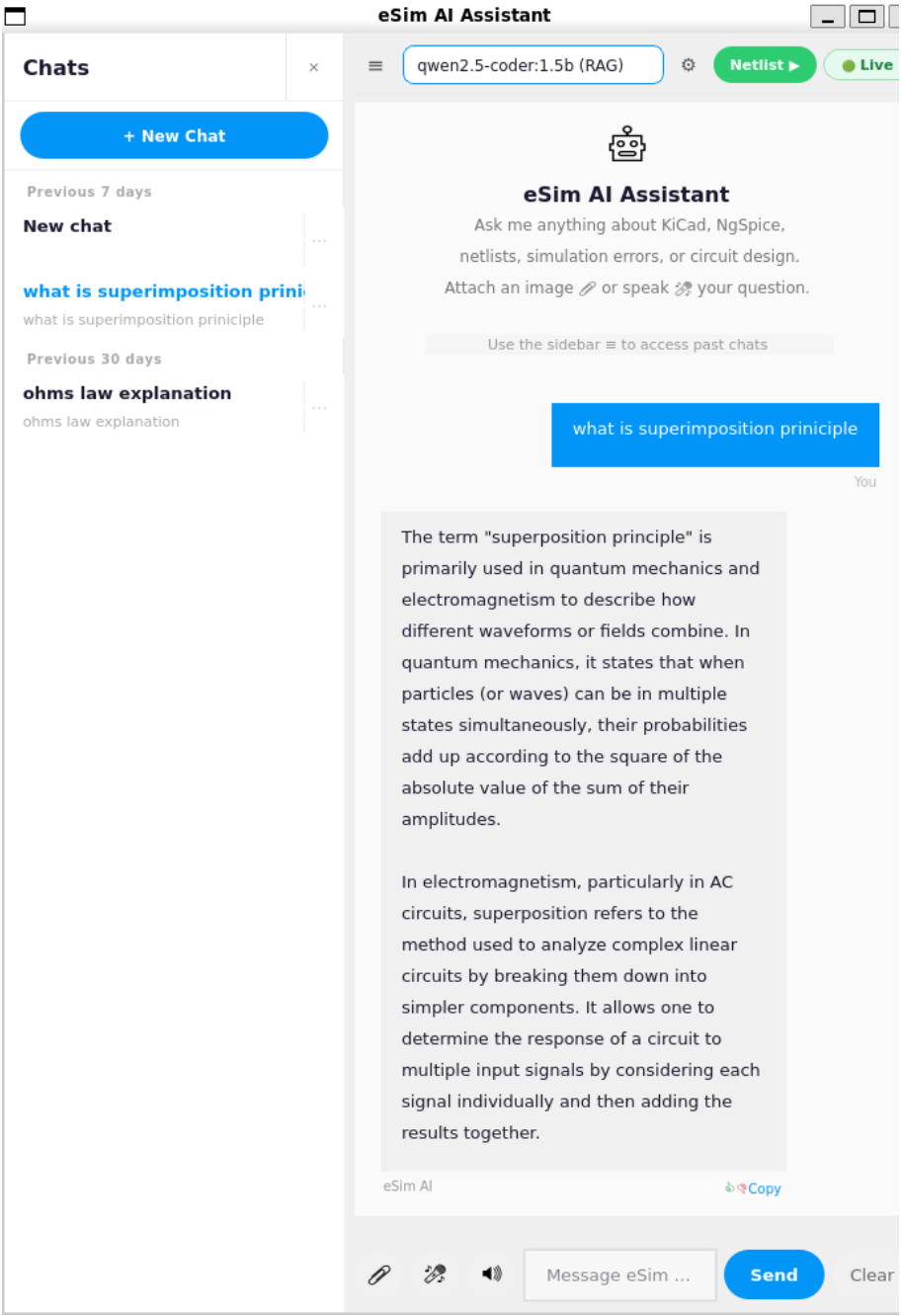

Figure 3.1: eSim Copilot chatbot interface showing the chat area, sidebar, model selection, and input controls

3.6 Sending Text Queries

Users can interact with the chatbot by entering natural language queries directly into the input field.

Example queries include:

- Why am I getting a singular matrix error?

- Explain transient analysis in NgSpice.
- How do I fix floating nodes in my circuit?
- What does this resistor configuration do?

The chatbot processes the query using locally running AI models combined with documentation retrieval through the RAG pipeline.

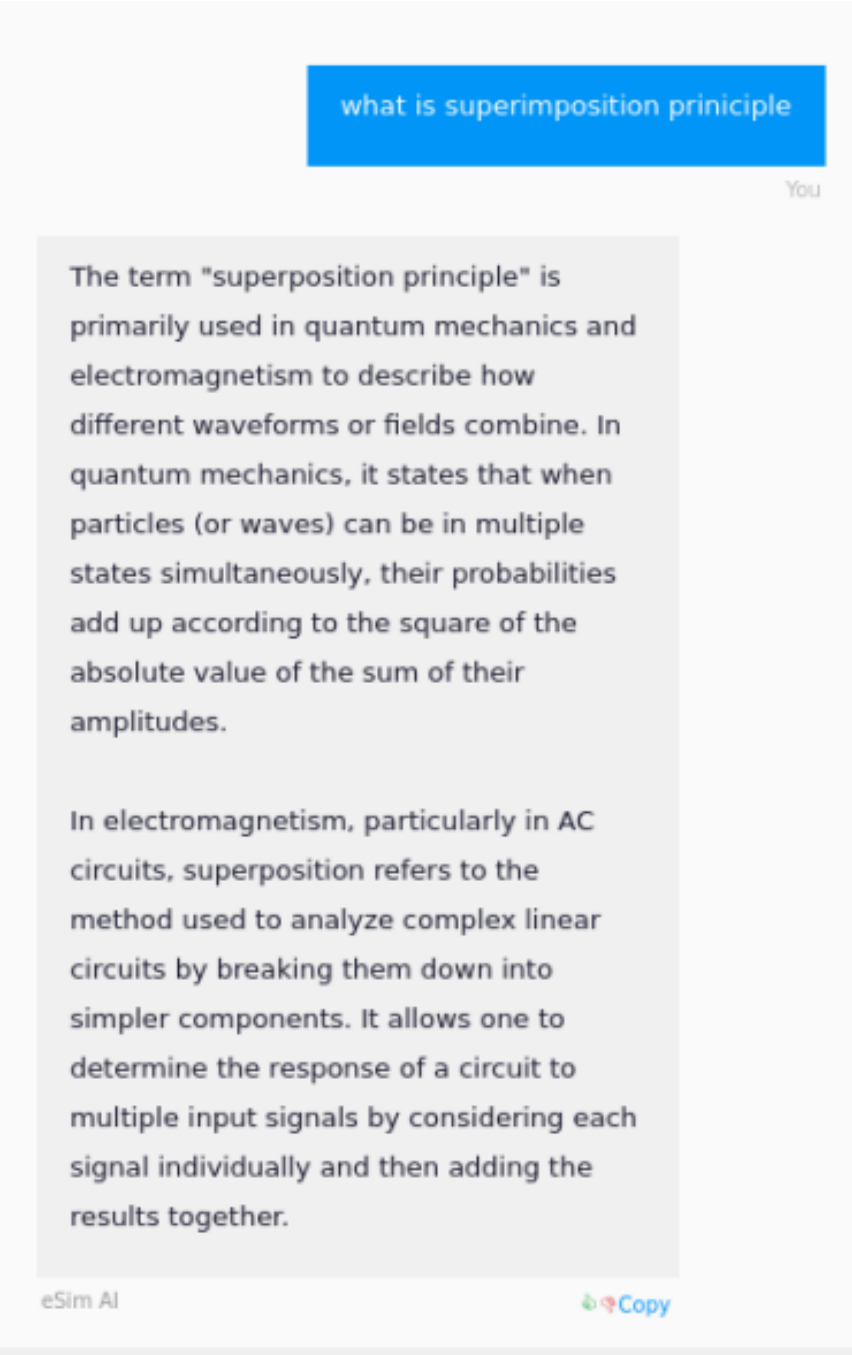

Figure 3.2: Sending natural language queries to the eSim Copilot

3.7 Analysing Circuit Schematic Images

The chatbot supports multimodal interaction through circuit image analysis.

The image analysis workflow includes:

- Uploading the schematic image
- OCR-based text extraction
- Vision-language model processing
- AI-generated explanation of the circuit

The system can identify components such as resistors, capacitors, voltage sources, and transistors while also explaining circuit functionality.

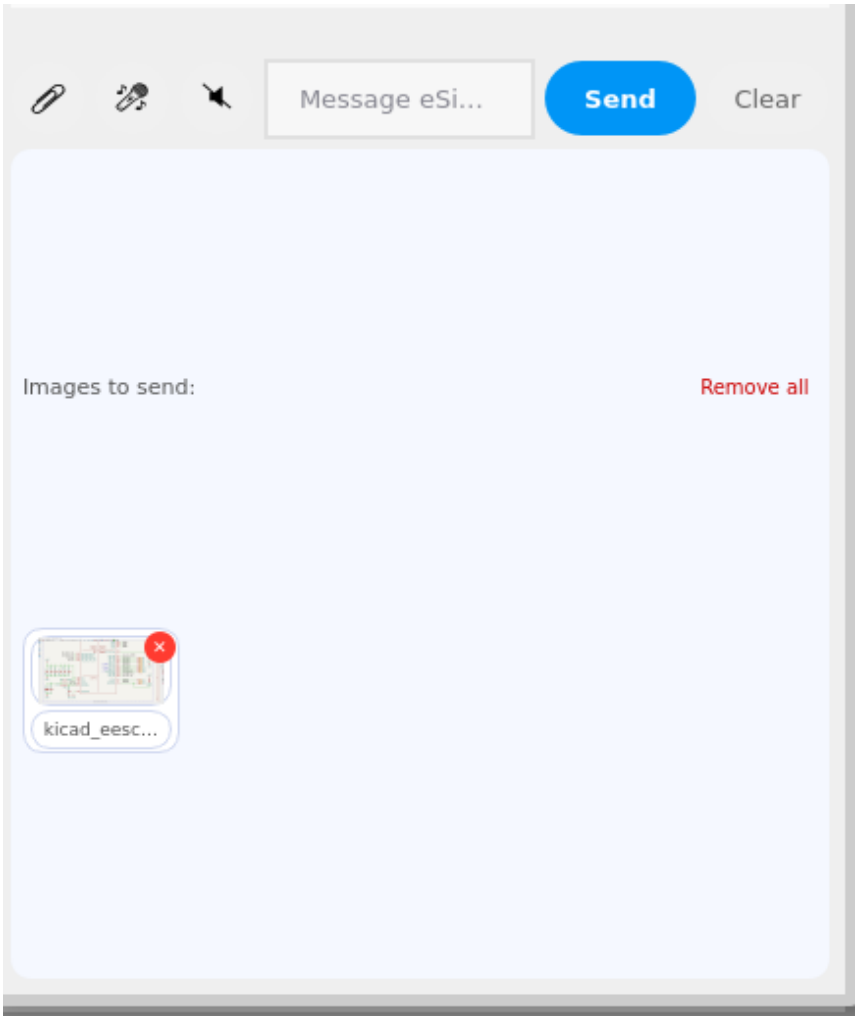

Figure 3.3: Circuit schematic image analysis workflow

3.8 Using Voice Input

Voice input functionality allows users to communicate with the chatbot using spoken language.

The feature was implemented using the SpeechRecognition library and PyAudio for microphone access.

To use voice input:

- Click the microphone button
- Speak the query clearly
- Wait for speech recognition processing
- The recognized text is inserted automatically

Voice interaction improves accessibility and provides a more interactive learning experience.

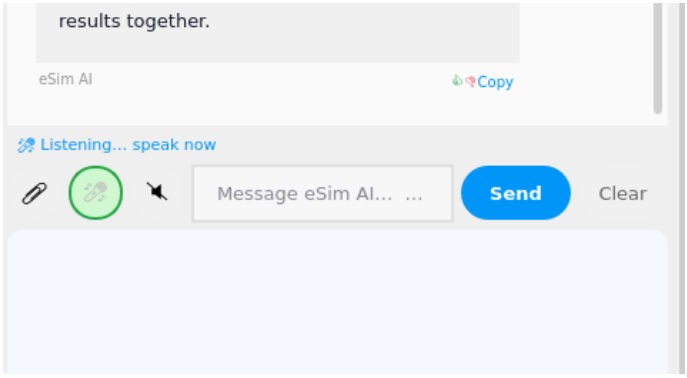

Figure 3.4: Voice input feature integrated into the chatbot

3.9 Voice Output Responses

The chatbot also supports voice-based output responses, allowing generated answers to be read aloud to the user.

This feature improves:

- Accessibility support
- Hands-free interaction
- Educational demonstrations
- User engagement

Voice output makes the assistant more conversational and beginner-friendly.

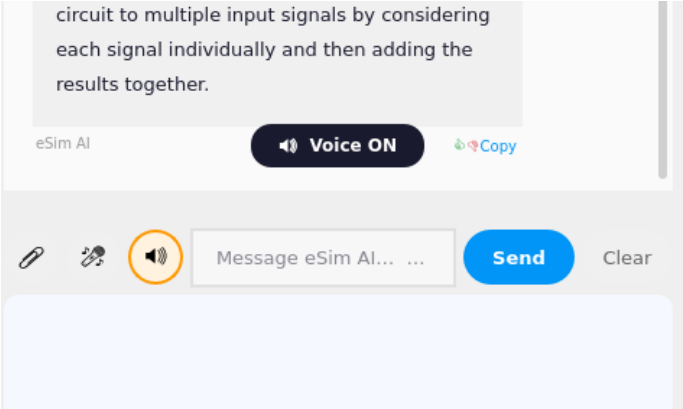

Figure 3.5: Voice output response generation in the chatbot

3.10 Session History and Sidebar Navigation

The chatbot supports persistent multi-session chat history similar to modern conversational AI platforms.

The session management system allows users to:

- Create multiple chat sessions
- Switch between saved conversations
- Continue previous discussions
- Delete unwanted sessions
- Maintain conversation history across restarts

This feature improves usability and allows users to organize discussions related to different projects or debugging tasks.

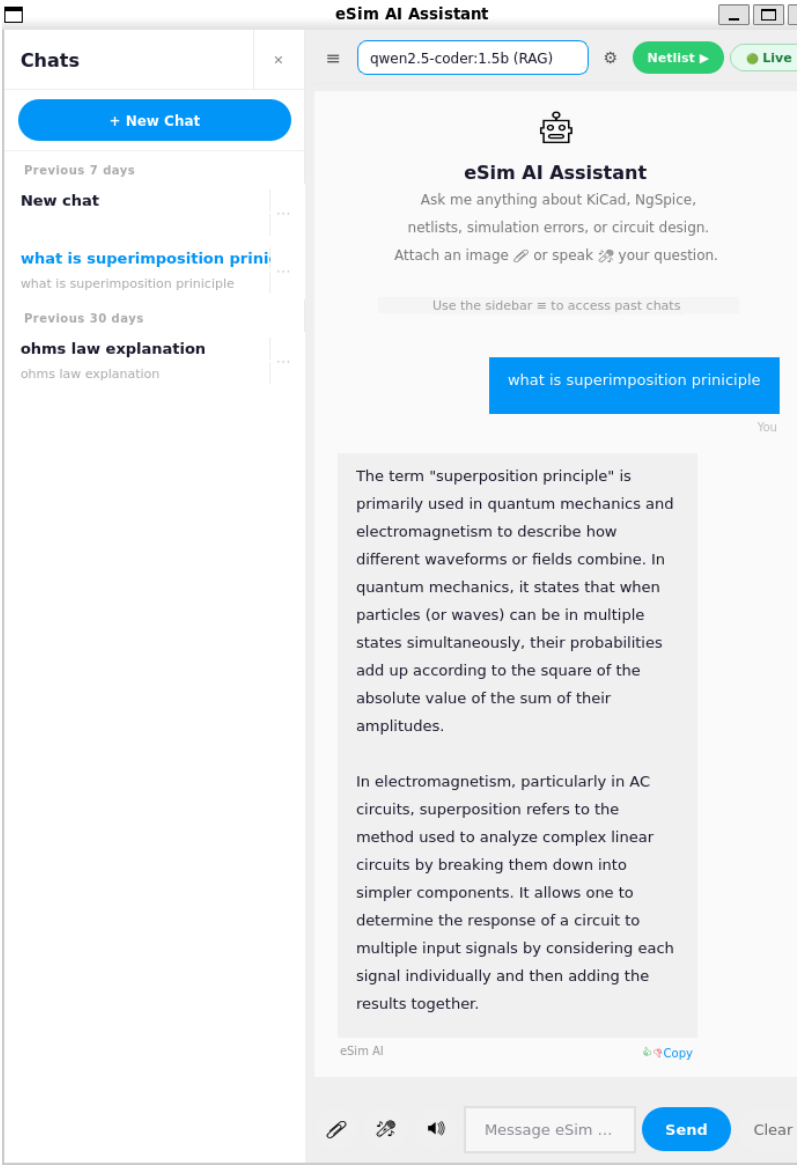

Figure 3.6: Persistent chat history and sidebar navigation

3.11 Changing AI Models

The chatbot supports dynamic switching between multiple Ollama-based AI models directly from the interface.

Supported models include:

- qwen2.5:3b
- qwen2.5-coder:3b
- tinyllama
- minicpm-v

Different models can be selected depending on:

- General conversational assistance
- Programming and debugging tasks
- Vision-based analysis
- Lightweight inference on low-resource systems

Dynamic model selection improves flexibility and adapts the chatbot for multiple use cases.

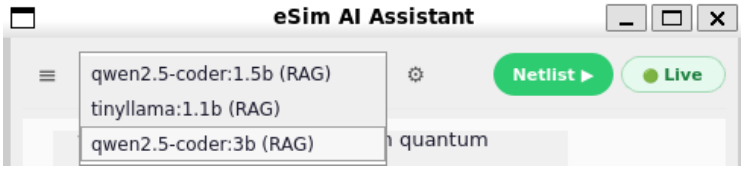

Figure 3.7: Dynamic AI model selection within the chatbot

Chapter 4

Problem Analysis

4.1 Existing Challenges in eSim

eSim is a comprehensive open-source Electronic Design Automation (EDA) platform that integrates schematic design, simulation, and PCB design within a single environment. Although the software provides powerful capabilities for circuit analysis and simulation, many users—especially students and beginners—experience difficulties while working with the platform. One of the major reasons is the steep learning curve associated with SPICE simulation workflows and technical debugging processes.

Users often struggle to understand simulation failures, incorrect waveform outputs, and NgSpice error messages. Since most errors are generated in textual SPICE netlists, debugging requires manually inspecting circuit files and cross-referencing them with documentation. For beginners who are unfamiliar with SPICE syntax and simulation workflows, this process becomes time-consuming and confusing.

Another limitation observed in the existing workflow was the lack of integrated intelligent assistance within the application itself. Whenever users encountered errors or conceptual doubts, they were forced to leave the eSim environment and search external resources such as forums, manuals, or tutorial videos. This fragmented workflow interrupted productivity and made the learning process less efficient.

The absence of multimodal interaction was also a significant limitation. Users could not directly upload circuit screenshots or schematic images for analysis, even though visual information often provides better context for understanding circuit behaviour. These challenges highlighted the need for a context-aware AI assistant capable of operating directly within the eSim environment.

4.2 Difficulties in Netlist Debugging

One of the most difficult aspects of using eSim is understanding and debugging SPICE netlists generated during simulation. NgSpice netlists contain textual descriptions of circuit connections, component definitions, and simulation commands. Although

these files are essential for simulation execution, they are not beginner-friendly and require technical knowledge to interpret correctly.

During simulation failures, users are often presented with complex error messages such as “singular matrix,” “floating node,” or “unknown subcircuit.” These messages rarely explain the exact root cause of the problem in simple language. As a result, users must manually inspect generated `.cir.out` files and compare them with documentation to identify the issue.

Several common debugging problems were repeatedly observed during development and testing. These included missing ground connections, incorrect voltage source configurations, invalid simulation parameters, missing component models, and unconnected nodes. In many cases, even small mistakes in the schematic generated large and difficult-to-understand netlist errors.

The lack of automated explanation and corrective guidance increased debugging complexity significantly. This motivated the integration of a netlist analysis system capable of interpreting errors and generating simplified explanations for users in natural language. The chatbot was designed to reduce this debugging burden by providing contextual assistance and possible solutions directly within the application interface.

4.3 Issues Faced During Development

The development and testing process involved several technical challenges related to dependency management, graphical rendering, Linux environment configuration, and AI model integration. Since the chatbot relied on multiple libraries including PyQt5, OpenCV, SpeechRecognition, PaddleOCR, and Ollama, maintaining compatibility between these components became an important part of the project.

Many issues were encountered while setting up the Ubuntu WSL environment, configuring graphical support, installing required dependencies, and integrating local AI models. Resolving these challenges was necessary to achieve a stable and fully functional chatbot system.

4.3.1 Dependency Installation Problems

During the initial setup process, several dependency installation issues were encountered while configuring the Python virtual environment and installing required libraries. Some packages depended on additional Linux system libraries that were not available by default in the Ubuntu environment.

A major issue occurred while installing PyAudio for voice input functionality. The installation repeatedly failed because the required Linux audio packages were missing. This problem was eventually resolved by manually installing the `python3-pyaudio` package through the Ubuntu package manager.

Additional problems included interrupted package downloads, timeout errors during pip installation, and dependency conflicts between OpenCV and PyQt5 libraries. Certain packages also required repeated installation attempts because of unstable Ubuntu mirror connections.

Managing dependencies inside a Python virtual environment helped isolate project packages from the system installation and reduced compatibility issues. However, maintaining a stable environment still required careful package management and repeated troubleshooting throughout development.

4.3.2 WSL and Ubuntu Configuration Issues

The project was developed primarily using Ubuntu 22.04 running inside Windows Subsystem for Linux (WSL). Although WSL provides a convenient Linux environment on Windows systems, several configuration-related issues were encountered during development.

One of the major challenges involved graphical application support within the WSL environment. Since the chatbot interface was built using PyQt5, proper graphical rendering support was necessary for successful execution. Incorrect runtime permissions and graphical subsystem inconsistencies occasionally prevented the chatbot window from opening correctly.

Additional issues included environment variable misconfigurations, virtual environment activation problems, and occasional loss of terminal state after reloading the VS Code window. File path differences between Windows and Linux environments also created confusion during directory navigation and project setup.

Despite these challenges, WSL provided a flexible and lightweight development platform that allowed Linux-based AI tools and Python libraries to be used efficiently on a Windows system. The environment was eventually stabilized after repeated configuration adjustments and testing.

4.3.3 Qt and OpenCV Plugin Conflicts

One of the most critical runtime problems encountered during development involved conflicts between Qt plugins used by PyQt5 and OpenCV. The chatbot occasionally failed to launch and produced errors related to the Qt platform plugin `xcb`.

These errors prevented the PyQt5 graphical interface from initializing correctly and caused the application to terminate immediately during startup. The issue originated from conflicts between OpenCV's GUI-related Qt components and the existing PyQt5 graphical environment within WSL.

Several debugging approaches were explored to resolve the issue. These included modifying Qt environment variables, testing different plugin paths, reinstalling

OpenCV libraries, and replacing `opencv-python` with `opencv-python-headless` to remove unnecessary GUI dependencies.

The issue was eventually stabilized through proper graphical environment configuration and by avoiding conflicting OpenCV GUI components. Resolving this problem was essential because the chatbot interface depended entirely on PyQt5 for rendering the user interface and handling interaction workflows.

4.3.4 Ollama Runtime and Port Issues

The chatbot relied on Ollama as the backend framework for running local AI models such as qwen2.5 and MiniCPM-V. While integrating Ollama, several runtime and connectivity-related issues were encountered.

One common issue occurred while manually starting the Ollama server using the `ollama serve` command. In many cases, the terminal displayed the message "address already in use," indicating that the Ollama service was already running in the background and occupying the default port.

Additional difficulties included understanding how models should be downloaded and managed, especially for first-time users unfamiliar with Ollama. Since some models were several gigabytes in size, downloads required considerable time and storage space. Verifying whether models had been installed correctly also became an important part of the setup process.

The runtime environment occasionally failed when models were incompletely initialized or when multiple models were loaded simultaneously. These issues were resolved through proper model management, checking running services, and validating model availability using commands such as `ollama list`.

Successfully stabilizing the Ollama environment enabled complete offline AI execution, which became one of the major strengths of the eSim Copilot system.

Chapter 5

Design and Implementation

5.1 Overall System Architecture

The eSim Copilot system was designed as a modular AI-assisted framework integrated directly into the eSim environment. The architecture combines conversational AI, Retrieval-Augmented Generation (RAG), computer vision, speech processing, and SPICE netlist analysis within a unified PyQt5-based interface. The system operates completely offline using locally hosted AI models through Ollama, ensuring privacy and accessibility without requiring cloud services.The architecture consists of multiple interconnected modules responsible for query handling, document retrieval, image processing, voice interaction, and response generation. Each module was designed independently to improve maintainability, debugging, and future scalability.The chatbot workflow begins with user interaction through text, image, or voice input. The request is then routed to the appropriate processing pipeline based on the query type. Responses are generated using local LLMs combined with contextual information retrieved from the RAG knowledge base.

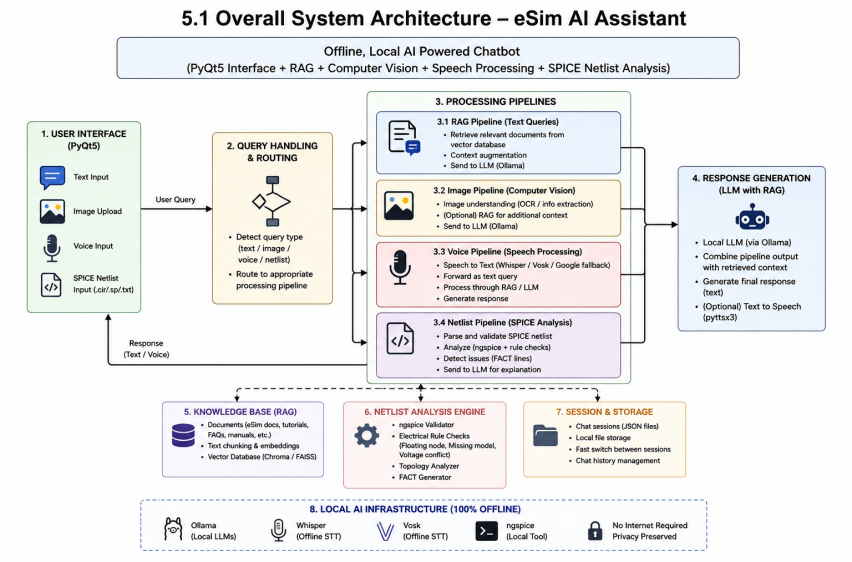

Figure 5.1: Overall system architecture of the eSim Copilot showing the interaction between user input, AI processing modules, RAG pipeline, and local LLM integration

5.1.1 Module Organization

The project follows a modular structure where different functionalities are separated into dedicated components. This modular organization simplified debugging and allowed independent testing of individual subsystems during development.

The major modules included in the system are:

- Chatbot interface module
- Query routing and processing module
- RAG knowledge retrieval module
- Image analysis module
- Voice input and output module
- Netlist analysis engine
- Session history management system
- Ollama model integration layer

The frontend components were implemented using PyQt5, while backend AI functionalities relied on Ollama, ChromaDB, PaddleOCR, and local language models.

5.1.2 Data Flow Between Components

The system follows a sequential data flow pipeline beginning from user interaction and ending with response generation. When a user submits a query, the chatbot first determines the input type, such as text, image, voice, or netlist-related content. Based on the classification result, the request is routed to the appropriate processing module.

For textual queries, the system forwards the request to the language model along with contextual information retrieved from the RAG system. Image-based queries pass through OCR and vision-language processing pipelines before generating a response. Voice queries are converted into text before entering the normal conversational workflow.

The generated response is finally displayed inside the PyQt5 interface and optionally converted into speech using the voice output module. The complete interaction is stored within the session management system for persistence and future retrieval.

5.2 Core Feature Development

5.2.1 AI Query Processing Pipeline

The chatbot uses a centralized query processing pipeline responsible for handling user requests and generating responses. The pipeline acts as the core orchestration layer connecting the frontend interface with backend AI services.When a user submits a query, the system first preprocesses the input to determine the query category. The request is then forwarded to the corresponding processing module such as conversational AI, image analysis, or netlist debugging.

The processed query is combined with contextual information and passed to the selected Ollama model for response generation. The generated output is formatted and displayed within the chat interface using bubble-style rendering.

5.2.2 RAG-Based Knowledge Retrieval

The chatbot integrates a Retrieval-Augmented Generation (RAG) system to improve the accuracy of responses related to eSim documentation and simulation concepts.

The implementation uses ChromaDB as the vector database and `nomic-embed-text` for embedding generation. Documentation files and manuals were converted into embeddings and stored within the vector database.

When a user submits a documentation-related query, the system retrieves the most relevant information using semantic similarity search and provides it to the language model during response generation.

5.2.3 Circuit Image Processing

The chatbot supports circuit schematic analysis through a multimodal image processing pipeline. Users can upload screenshots or schematic diagrams directly into the interface for analysis.

The uploaded image first passes through an OCR stage implemented using PaddleOCR to extract textual labels and annotations. The extracted information is then processed using the MiniCPM-V vision-language model.

The model identifies circuit components, interprets connectivity patterns, and generates descriptive explanations about circuit functionality.

5.2.4 Voice Input Module

Voice input functionality was implemented to improve accessibility and provide a more interactive conversational experience.

The implementation uses the SpeechRecognition library along with PyAudio for microphone access. Spoken queries are converted into text and processed through the normal chatbot workflow.

This feature improves usability for beginners and supports hands-free interaction within the chatbot interface.

5.2.5 Voice Output Module

The chatbot also supports voice-based response output by converting generated text responses into audio playback.

This feature enhances accessibility and creates a more conversational interaction experience. Users can listen to generated responses instead of reading them manually.

5.2.6 SPICE Netlist Analysis

The SPICE netlist analysis module was developed to simplify debugging within eSim. The analyzer automatically checks generated netlists and identifies common simulation-related issues.

The system detects problems such as floating nodes, missing ground references, invalid configurations, and missing component models. Instead of displaying complex NgSpice errors directly, the chatbot generates simplified explanations and possible corrective suggestions.

5.2.7 Session Persistence and Chat Storage

The chatbot implements a persistent session management system similar to modern conversational AI platforms. Conversations are stored locally and displayed through a sidebar-based navigation interface.

Users can create multiple chat sessions, switch between conversations, and continue previous discussions without losing context. This significantly improves usability and organization of chatbot interactions.

5.2.8 Ollama Model Integration

Ollama was integrated as the local inference backend for executing AI models offline. The chatbot supports multiple Ollama-compatible language and vision models including qwen2.5, qwen2.5-coder, tinyllama, and MiniCPM-V.

The integration layer handles model selection, query forwarding, response retrieval, and runtime communication with the Ollama service. A dynamic model selector was also added to allow users to switch between models directly from the interface.

5.3 User Interface Integration

5.3.1 PyQt5 Chat Window Design

The chatbot interface was designed using PyQt5 to maintain compatibility with the existing eSim application architecture. The layout follows a modern conversational design inspired by contemporary AI chat systems.

The interface includes a chat display area, query input field, session sidebar, voice interaction controls, image upload support, and model selection components.

Special attention was given to responsiveness and asynchronous processing to prevent UI freezing during AI inference.

5.3.2 Message Rendering and Interaction

The chatbot uses a bubble-style message rendering system to visually separate user and assistant responses. Messages are dynamically styled and aligned to improve readability and create a cleaner conversational appearance.

Additional interaction features implemented during development include copy response functionality, image attachment support, session switching, and dynamic message updates during response generation.

These interaction improvements significantly enhanced the professional appearance and usability of the chatbot interface.

5.4 Debugging and Improvements

5.4.1 Startup Crash Resolution

During development, the chatbot occasionally crashed during startup due to Qt platform plugin conflicts and OpenCV compatibility issues inside the Ubuntu WSL environment. Errors related to the `xcb` plugin prevented the PyQt5 interface from loading properly. These issues were resolved by configuring the correct Qt plugin paths, adjusting environment variables, and improving dependency management within the virtual environment.

5.4.2 Empty Output and Rendering Issues

In some cases, the chatbot launched successfully but failed to display responses inside the chat window. This issue was mainly caused by improper UI rendering and thread synchronization during AI response generation. The rendering logic was improved to ensure stable message updates, proper chat refresh behaviour, and smoother interaction during long-running tasks.

5.4.3 Stability and Performance Improvements

Several optimizations were implemented to improve the overall stability and responsiveness of the chatbot. Background processing was refined to reduce UI freezing during model inference, while session handling and message management were improved for better reliability. Additional fixes were also applied for smoother Ollama integration and better compatibility within the eSim environment.

Chapter 6

Testing and Validation

6.1 Functional Testing

Functional testing was carried out to verify whether all major modules of the eSim Copilot were operating correctly inside the eSim environment. The testing process focused on validating text-based interaction, circuit image analysis, voice input/output, session history management, AI model switching, and SPICE netlist analysis. Each feature was tested individually and later verified together through complete workflow execution.

The chatbot was launched both as a standalone interface and within the eSim application to ensure stability in different execution modes. Multiple prompts related to circuit simulation, NgSpice errors, component identification, and eSim workflow assistance were tested. The system successfully generated context-aware responses using locally running Ollama models.

Special attention was given to persistence-related features such as chat history saving, sidebar navigation, and session restoration. Tests confirmed that sessions were automatically saved and reloaded without corruption. Empty sessions were ignored correctly, and deleted chats no longer reappeared after restarting the application.

Voice input functionality was tested using microphone-based queries, while voice output responses were verified through text-to-speech playback. The chatbot successfully converted spoken queries into text and generated audible AI responses.

6.2 Netlist Analysis Results

The SPICE netlist analysis module was tested using multiple `.cir` and `.cir.out` files generated from eSim projects. The objective was to evaluate the chatbot's ability to identify common simulation and connectivity issues and provide understandable debugging suggestions.

Several faulty netlists were intentionally created containing floating nodes, miss-

ing grounds, invalid component values, convergence failures, and missing model definitions. The analysis engine successfully detected these issues and produced simplified explanations for users.

For example, when a "Singular Matrix" error occurred due to a missing ground connection, the chatbot identified the issue and suggested adding a proper reference ground. Similarly, convergence-related errors were explained along with recommendations such as reducing timestep values or checking component parameters.

The analysis responses were significantly easier to understand compared to raw NgSpice error logs, making debugging more accessible for beginner users.

6.3 Circuit Image Analysis Results

The circuit image analysis feature was tested using screenshots of analog and digital schematics. Images containing resistors, capacitors, voltage sources, transistors, and integrated circuits were uploaded through the chatbot interface.

The MiniCPM-V vision model successfully identified most components and generated high-level descriptions of circuit functionality. OCR-based extraction also detected several component labels and values from uploaded schematics.

The system performed well for clean screenshots and standard circuit diagrams. However, slightly reduced accuracy was observed for blurry images, handwritten labels, and complex IC connections. Despite these limitations, the feature proved useful for quickly understanding circuit structures and assisting users during debugging or learning.

Follow-up questioning was also tested. Users could ask additional questions such as "What is the role of this resistor?" or "Which component acts as the output stage?" and receive context-aware answers based on the uploaded image.

6.4 Voice Interaction Testing

Voice interaction testing included both speech-to-text input and text-to-speech output validation. The microphone module was tested with different accents and speaking speeds to evaluate recognition quality.

The speech recognition pipeline successfully converted spoken English queries into text for most test cases. Voice commands related to eSim workflows, simulation errors, and circuit explanations were processed correctly by the chatbot.

The voice output feature generated audible AI responses using the integrated text-to-speech system. This improved accessibility and created a more interactive user experience.

Some limitations were observed in WSL2 environments where microphone access and audio device detection occasionally failed. These issues were documented along with setup instructions and environment-specific fixes.

6.5 Overall Performance Evaluation

The overall performance of the system was evaluated in terms of usability, response time, stability, and resource utilization. Since the assistant operates fully offline using local LLMs through Ollama, performance depended primarily on system hardware and model size.

Text-based responses were generally generated within a few seconds, while image analysis required additional processing time because of OCR and vision-model inference. Session management operations such as loading chat history and switching conversations remained smooth even after storing multiple chats.

Memory usage increased when larger models such as MiniCPM-V were loaded, but the system remained stable during prolonged usage. Several earlier crashes and rendering problems were eliminated through debugging and optimization.

The final implementation successfully provided an integrated AI assistance system within eSim while maintaining offline functionality, privacy, and usability.

Chapter 7

Conclusion and Future Work

7.1 Conclusion

This project successfully enhanced and extended the eSim Copilot by improving its stability, usability, and AI-assisted workflow capabilities. The chatbot was integrated into the eSim environment using PyQt5 and Ollama-based local language models, enabling users to receive assistance directly within the application without depending on cloud services.

Several major features were implemented and refined during development, including persistent chat history, multi-session sidebar navigation, voice input, voice output responses, circuit image analysis, SPICE netlist debugging, and dynamic model selection. The chatbot was also redesigned with a modern bubble-style interface to improve user interaction and readability.

In addition to feature development, multiple bugs and compatibility issues were identified and resolved during the implementation process. These included dependency conflicts, rendering issues, startup crashes, OpenCV-Qt conflicts, Ollama runtime problems, and image-processing failures. Fixing these issues significantly improved the robustness and reliability of the system.

The completed implementation demonstrates how local AI models can effectively support Electronic Design Automation workflows while maintaining user privacy and offline accessibility. The project also contributes toward making eSim more beginner-friendly by simplifying debugging and reducing the learning curve associated with NgSpice simulations and circuit analysis.

7.2 Summary of Contributions

The major contributions of this work are summarized below:

- Enhanced the eSim Copilot interface using PyQt5 with modern chat-style rendering
- Implemented persistent multi-session chat history with sidebar navigation
- Added voice input functionality using speech recognition techniques
- Integrated voice output responses for interactive AI communication
- Improved circuit image analysis using vision-language models and OCR processing
- Developed automated SPICE netlist error analysis and debugging assistance
- Added dynamic AI model selection through Ollama integration
- Resolved multiple dependency, rendering, startup, and runtime issues
- Created installation and setup documentation for easier deployment
- Ensured complete offline operation without external data transmission

7.3 Future Enhancements

Although the current implementation provides a stable and functional AI assistant for eSim, several improvements can further enhance the system in the future.One possible enhancement is the addition of real-time circuit guidance during schematic creation. Instead of waiting until simulation time, the chatbot could proactively warn users about missing connections, invalid values, or incorrect wiring while the circuit is being designed.

The image analysis module can also be improved by supporting multiple-image uploads and higher-accuracy component recognition for complex schematics. Future versions may include automatic circuit summarization and generation of simplified explanations for educational purposes.Another useful enhancement would be exporting chat sessions and debugging reports as PDF documents. This would allow users to save troubleshooting discussions and share them with instructors or collaborators.

The voice system can be expanded further by supporting multilingual interaction and regional language assistance. This would improve accessibility for students and users from different backgrounds.Future work may also include integration of a stronger RAG-based documentation system capable of dynamically indexing additional eSim manuals, tutorials, and user-generated resources.

Additional future improvements may include:

- One-click automated fixes for common netlist errors
- Support for more Ollama language models
- Smarter AI reasoning for advanced circuit debugging
- Better optimization for low-end hardware systems
- Keyboard shortcuts and workflow automation inside eSim
- Improved vision analysis for handwritten schematics
- Cloud-sync support for optional cross-device session backup

Overall, the project establishes a strong foundation for integrating intelligent offline AI assistance into open-source EDA environments and opens possibilities for more advanced AI-driven circuit design support in the future.

Bibliography

[1] FOSSEE Team, IIT Bombay, *eSim: Open Source Electronic Design Automation Tool*. Available: `https://esim.fossee.in/`

[2] Ngspice Development Team, *Ngspice User Manual and Documentation*. Available: `https://ngspice.sourceforge.io/`

[3] Ollama, *Run Large Language Models Locally*. Available: `https://ollama.com/`

[4] Riverbank Computing Limited, *PyQt5 Documentation*. Available: `https://www.riverbankcomputing.com/software/pyqt/`

[5] Python Software Foundation, *Python Documentation*. Available: `https://www.python.org/doc/`

[6] P. Lewis, E. Perez, A. Piktus et al., "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks," *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

[7] ChromaDB, *Open-Source Embedding Database for AI Applications*. Available: `https://www.trychroma.com/`

[8] Alibaba Cloud, *Qwen2.5 Technical Report*, 2024. Available: `https://qwenlm.github.io/`

[9] OpenBMB, *MiniCPM-V: Vision Language Model Documentation*, 2024. Available: `https://github.com/OpenBMB/MiniCPM-V`

[10] Baidu Inc., *PaddleOCR Toolkit*. Available: `https://github.com/PaddlePaddle/PaddleOCR`