



# Semester Long Internship Spring 2026

On

## eSim Chatbot Development

Submitted by

**Neha Dhumal**

Vidyalankar Institute Of Technology

Under the guidance of

**Prof. Prabhu Ramachandran**

Principal Investigator
Department of Aerospace Engineering
Indian Institute of Technology Bombay

# Acknowledgment

I express my sincere gratitude to **Prof. Prabhu Ramachandran** for providing me with the opportunity to participate in the FOSSEE internship programme and for his sustained commitment to the advancement of open-source engineering education in India.

I also acknowledge **Prof. Kannan M. Moudgalya** for his foundational contributions to the FOSSEE initiative and for establishing the academic framework through which this internship was undertaken.

Sincere appreciation is owed to **Sumanto Kar** (Mentor) for his continuous technical guidance and feedback throughout the duration of this project, and to **Mr. Varad Patil** and **Ms. Shanthi Priya K** (Internal Mentors) for their coordination and technical inputs at each project phase.

Gratitude is also extended to the entire **FOSSEE team** for their coordination, resource accessibility, and timely support.

# Contents

# Chapter 1

# Introduction

## 1.1 Background

Large Language Models (LLMs) have enabled the development of intelligent conversational assistants capable of understanding natural language queries and providing context-aware responses. These AI systems are increasingly being integrated into domain-specific software applications to simplify complex tasks and improve user interaction.

In Electronic Design Automation (EDA), users work with circuit schematics, SPICE simulation commands, component configurations, and debugging procedures. For beginners, understanding simulation errors and navigating technical documentation can often be difficult and time-consuming.

The FOSSEE (Free and Open Source Software for Education) project at the Indian Institute of Technology Bombay develops open-source engineering tools for academic use. One of its major projects, eSim, is an open-source EDA platform built using KiCad and NgSpice for analogue, digital, and mixed-signal circuit design and simulation.

Although eSim provides powerful design and simulation capabilities, users previously depended on external documentation and online resources for guidance and troubleshooting. This project focuses on improving the Offline AI Chatbot integration in eSim using Ollama-based local LLM inference to provide context-aware assistance directly within the application while operating fully offline without external data transmission.

## 1.2 Problem Statement

This project addresses the gap between a user's circuit-design requirements and the technical complexity involved in using eSim effectively. Although eSim provides powerful simulation and design capabilities, users often face difficulties while understanding errors, configuring simulations, and analysing circuit behaviour.

The major challenges identified are as follows:

- **Lack of Context-Aware Assistance:** Traditional documentation and tutorials cannot provide real-time answers to project-specific questions such as

simulation errors, component selection, or circuit-debugging issues.

- **Manual and Time-Consuming Debugging:** Users are required to manually inspect netlists, simulation outputs, and error logs to identify problems, making the debugging process difficult and inefficient for beginners.
- **No Intelligent Schematic Analysis:** Existing workflows do not provide instant analysis of uploaded schematic images for component detection, circuit understanding, or identification of possible design mistakes.
- **Workflow Disruption:** Users frequently need to switch between eSim, documentation, forums, and external resources to resolve issues, which interrupts the overall design workflow and reduces productivity.
- **Dependence on Internet-Based AI Tools:** Many modern AI assistants require cloud connectivity, raising concerns related to internet dependency, latency, privacy, and external data transmission.

## 1.3 Project Objectives

The main objective of this project is to enhance the existing AI chatbot support within eSim by developing a fully offline and intelligent assistant capable of providing real-time guidance for circuit design, simulation, and debugging activities.

The key objectives of the project are listed below:

1. To integrate locally running Large Language Models (LLMs) through the Ollama framework in order to achieve secure and internet-independent AI processing.
2. To build a Retrieval-Augmented Generation (RAG) system using official eSim manuals and documentation so that the chatbot can generate accurate and context-relevant responses.
3. To implement a circuit schematic analysis pipeline capable of identifying electronic components, extracting labels and values, and understanding circuit functionality from uploaded images.
4. To design an automated error-analysis mechanism for detecting common NgSpice and eSim simulation problems and providing suitable corrective suggestions.
5. To improve chatbot conversation handling through intelligent query classification and semantic topic-switch detection.
6. To add offline speech-input functionality using faster-whisper for more interactive and user-friendly communication.
7. To integrate the chatbot smoothly within the PyQt5-based eSim interface without affecting the existing application workflow.
8. To maintain complete offline functionality and preserve user privacy by avoiding any dependency on external cloud-based AI services.

## 1.4 Scope of the Project

This project focuses on improving the Offline AI Chatbot integration within eSim to provide intelligent assistance during circuit design, simulation, and debugging workflows. The system is designed to support natural language query handling, RAG-based documentation support, schematic image analysis, automated error detection, and offline voice interaction. The project mainly targets eSim and NgSpice-related operations while ensuring complete offline execution, data privacy, and seamless integration within the existing eSim environment.

## 1.5 Methodology

The development of the improved Offline AI Chatbot for eSim was carried out in multiple stages to ensure systematic integration and testing of all features.

| **Phase** | **Description** |
|---|---|
| System Analysis | Study of the existing chatbot architecture, eSim workflow, and identification of limitations in the previous implementation. |
| Core Development | Integration of Ollama-based local LLM inference, chatbot routing logic, and RAG-based knowledge retrieval system. |
| Feature Enhancement | Implementation of schematic image analysis, OCR integration, semantic topic-switch detection, and offline voice-input support using faster-whisper. |
| Error Handling and Optimisation | Development of automated error-analysis mechanisms, debugging improvements, and optimisation of chatbot response handling. |
| Testing and Integration | End-to-end testing of the chatbot inside the eSim environment to verify stability, performance, and offline functionality. |

# Chapter 2

# Literature Review and Background

## 2.1 eSim and Existing Workflow

The FOSSEE (Free and Open Source Software for Education) project at the Indian Institute of Technology Bombay promotes the use of open-source software tools for engineering education. One of its major contributions is eSim, an open-source Electronic Design Automation (EDA) platform used for circuit design, simulation, and PCB development.

eSim integrates KiCad for schematic design and NgSpice for circuit simulation, providing support for analogue, digital, and mixed-signal circuits. The platform is widely used in academic institutions because it offers an open-source alternative to commercial EDA software.

Despite its capabilities, users often face difficulties while configuring simulations, understanding NgSpice errors, and debugging circuits. Existing documentation is static and requires users to search through manuals and online resources, interrupting the design workflow.

## 2.2 Artificial Intelligence in Engineering Applications

Artificial Intelligence technologies are increasingly being integrated into engineering tools to improve productivity and simplify technical workflows. AI-based assistants can help users understand simulation errors, analyse circuits, and provide context-aware technical guidance.

Most existing AI assistants are cloud-based and depend on internet connectivity for operation. Although cloud systems provide powerful AI capabilities, they introduce concerns related to privacy, latency, and internet dependency.

Recent developments in local Large Language Models (LLMs) and lightweight inference frameworks have enabled the development of offline AI systems that can run directly on local machines while preserving user privacy.

### 2.2.1 Large Language Models (LLMs)

Large Language Models are AI systems trained on large text datasets to understand and generate natural language responses. Modern LLMs based on transformer architecture are capable of answering technical questions, explaining concepts, and supporting conversational interaction.

In this project, locally deployable LLMs are used to provide offline chatbot support for eSim-related workflows, simulation debugging, and electronics queries.

### 2.2.2 Retrieval-Augmented Generation (RAG)

Retrieval-Augmented Generation (RAG) improves chatbot responses by retrieving relevant information from external documentation before generating the final response.

In a RAG system, documents are converted into vector embeddings and stored in a vector database. When a user submits a query, the system retrieves semantically relevant document chunks and provides them as context to the language model.

In this project, RAG is used with eSim and NgSpice documentation to improve the accuracy and reliability of chatbot responses.

### 2.2.3 Local LLM Inference using Ollama

Ollama is an open-source framework used for running Large Language Models locally. It simplifies model management, downloading, and local inference through a lightweight API interface.

The framework allows the chatbot to operate completely offline while maintaining low latency and data privacy. Ollama is also used for serving embedding models required for the RAG pipeline.

### 2.2.4 Vision-Language Models (VLMs)

Vision-Language Models combine image understanding with natural language processing. These models can analyse images and generate descriptive or context-aware responses.

In this project, Vision-Language Models are used for analysing uploaded circuit schematic images and generating AI-based explanations regarding components and circuit functionality.

### 2.2.5 PaddleOCR for Circuit Text Recognition

PaddleOCR is an open-source Optical Character Recognition framework used for extracting text from images. It supports detection of labels, component names, and values from circuit schematics.

The OCR pipeline is used as a preprocessing stage before image analysis to improve understanding of uploaded circuit diagrams.

### 2.2.6 Offline Speech Recognition using faster-whisper

faster-whisper is an optimized implementation of OpenAI Whisper designed for efficient offline speech recognition. It provides accurate speech-to-text conversion with lower memory usage and faster inference.

The chatbot integrates faster-whisper to support offline voice-input interaction without requiring internet connectivity.

### 2.2.7 PyQt5 for GUI Integration

PyQt5 is a Python framework used for developing desktop graphical user interfaces. Since eSim itself is developed using PyQt5, the chatbot is integrated directly into the existing GUI environment to provide seamless interaction within the application.

### 2.2.8 HDLParse and HDL File Analysis

HDLParse is a Python library used for parsing Hardware Description Languages such as Verilog and VHDL. It extracts module definitions, ports, and structural information from HDL files.

During development, compatibility issues were identified with older versions of HDLParse in modern Python environments. The package was updated and integrated to ensure stable HDL parsing support within the chatbot workflow.

## 2.3 Research Gap and Need for the Proposed System

Most existing AI assistants for engineering applications are cloud-based and are not specifically designed for open-source EDA workflows such as eSim. These systems often lack support for offline execution, schematic analysis, and context-aware debugging assistance.

The existing eSim workflow also lacked an integrated intelligent assistant capable of understanding simulation errors, analysing circuit images, and supporting multimodal interaction.

The proposed system addresses these limitations by integrating an offline AI chatbot directly within eSim using local LLM inference, RAG-based retrieval, OCR-assisted image analysis, and offline speech recognition.

## 2.4 Summary

This chapter discussed the background technologies and concepts related to the development of the Offline AI Chatbot system for eSim. It covered the eSim ecosystem, Artificial Intelligence in engineering applications, Large Language Models, Retrieval-Augmented Generation, Ollama, Vision-Language Models, PaddleOCR, faster-whisper, PyQt5, and HDLParse. These technologies collectively form the foundation of the proposed system.

# Chapter 3

# System Architecture and Design

## 3.1 Overall System Architecture

The Offline AI Chatbot integrated within eSim is designed using a modular multi-layer architecture to provide intelligent assistance for circuit design, simulation, debugging, and documentation support. The system combines local Large Language Models, Retrieval-Augmented Generation (RAG), schematic image analysis, speech recognition, and netlist processing into a unified offline framework.

The architecture is divided into four major layers:

- **eSim GUI Layer:** Handles user interaction through the PyQt5-based eSim interface, dockable chatbot window, project explorer, and simulation console integration.
- **Chatbot Core Layer:** Responsible for query routing, context management, image handling, speech processing, and netlist analysis.
- **AI Processing Layer:** Contains Ollama-managed AI models, embedding models, OCR systems, and offline speech-recognition modules.
- **Data and Knowledge Layer:** Stores eSim documentation, vector databases, HDL/netlist files, project files, and SPICE error knowledge bases.

The modular structure improves scalability, maintainability, and seamless integration with the existing eSim workflow5.

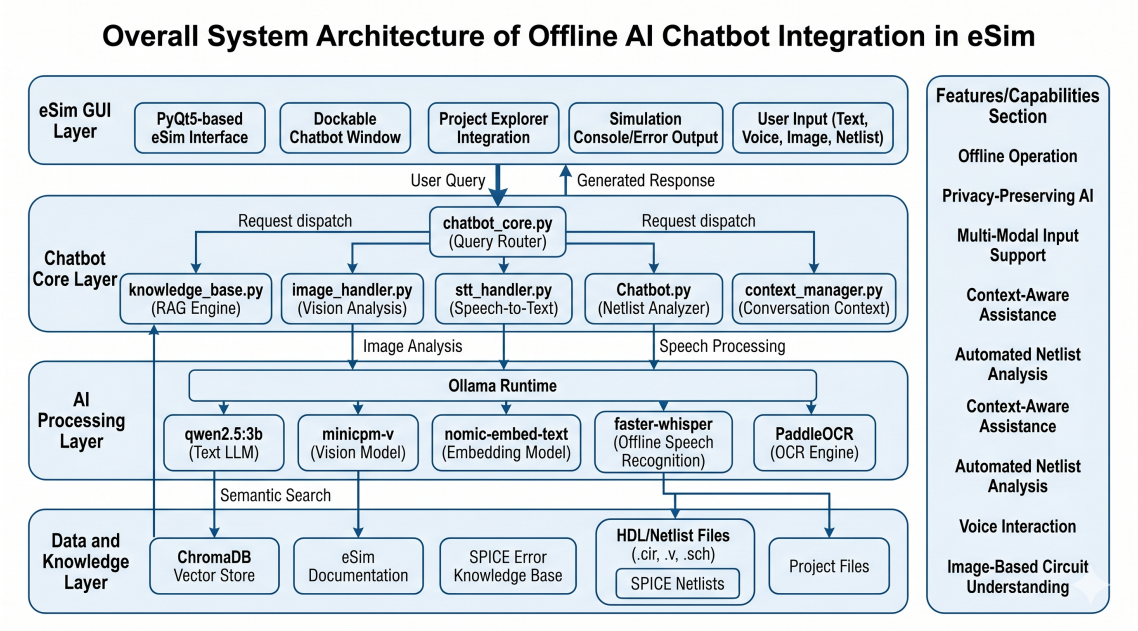


Figure 3.1: Overall System Architecture of Offline AI Chatbot Integration in eSim

## 3.2 Chatbot Workflow and Query Processing

The chatbot workflow begins when the user submits a query through text, voice input, image upload, or HDL/netlist files. The request is first received by the central routing module, `chatbot_core.py`, which classifies the query and forwards it to the appropriate processing module.

Text-based technical questions are routed through the RAG pipeline for semantic retrieval and contextual response generation. Voice queries are converted into text using offline speech-recognition models before further processing. Uploaded schematic images are analysed using OCR and Vision-Language Models, while HDL and SPICE netlists are processed through specialised parsing and debugging modules.

The processed information is combined with contextual knowledge and passed to the local language model for final response generation. The generated response is then displayed directly inside the eSim chatbot interface.

## 3.3 Retrieval-Augmented Generation (RAG) Pipeline

The Retrieval-Augmented Generation pipeline improves the factual accuracy and reliability of chatbot responses by retrieving relevant information from official eSim documentation before generating the final answer.

Initially, eSim manuals, tutorials, and technical documents are processed into smaller text chunks. These chunks are converted into vector embeddings using the `nomic-embed-text` embedding model and stored in the ChromaDB vector database.

When the user submits a query, the system performs semantic similarity search to retrieve the most relevant document chunks. The retrieved context is then combined

with the original query and passed to the local LLM through Ollama for response generation.

This approach significantly reduces hallucination and enables the chatbot to provide context-aware technical assistance aligned with official eSim documentation.

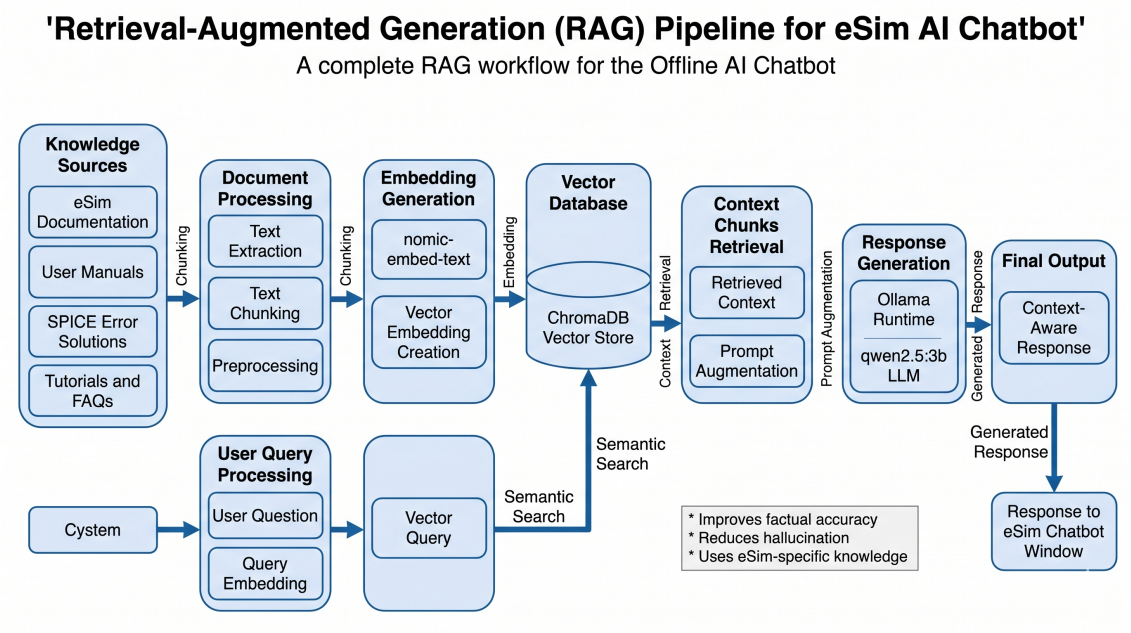


Figure 3.2: Retrieval-Augmented Generation (RAG) Pipeline

## 3.4 Schematic Image Analysis Pipeline

The chatbot system supports schematic image analysis using a multi-stage vision-processing pipeline. This feature enables users to upload circuit images and obtain AI-generated explanations and debugging assistance.

Initially, uploaded schematic images are preprocessed and passed through PaddleOCR for text extraction and component-label detection. The extracted information is then analysed using the `minicpm-v` Vision-Language Model to identify components, understand circuit topology, and analyse overall circuit functionality.

The interpreted information is forwarded to the language model for context-aware explanation and response generation.

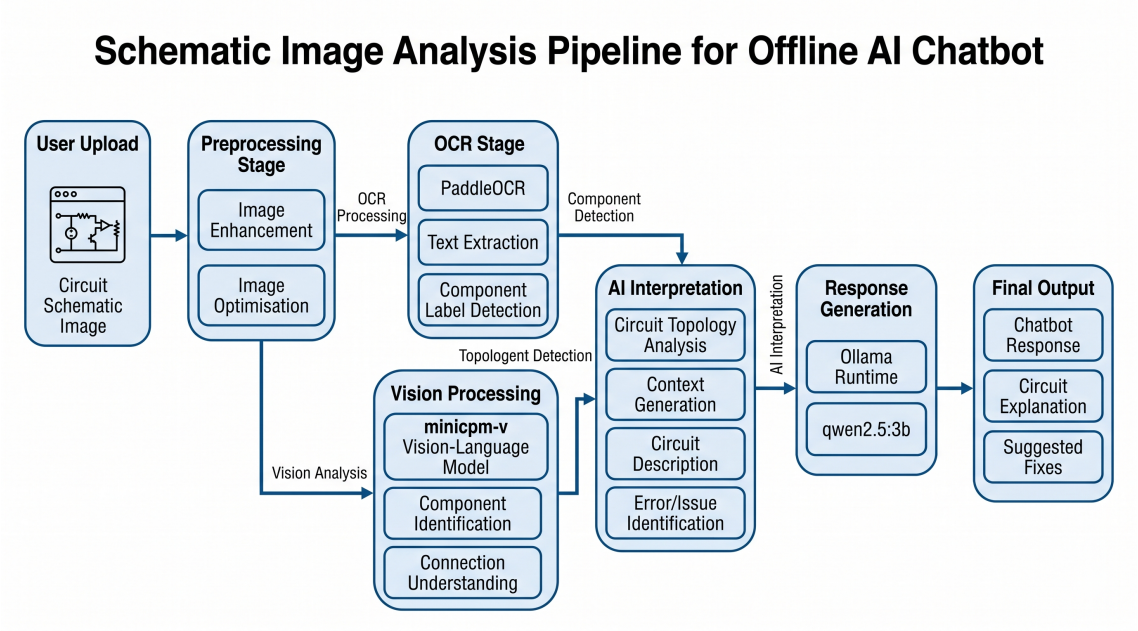


Figure 3.3: Schematic Image Analysis Pipeline

## 3.5 Integration with eSim GUI

The chatbot system is integrated directly into the existing PyQt5-based eSim graphical environment through a dockable chatbot interface. This integration allows users to interact with the assistant without leaving the eSim workspace.

The chatbot can access project files, schematic data, simulation outputs, and SPICE netlists directly from the active project environment. This enables the system to provide context-aware assistance related to the current workflow and simulation state.

The integration also supports real-time response generation and automated error analysis while maintaining a lightweight and responsive user interface.

## 3.6 Offline AI Processing using Ollama

The complete chatbot system operates fully offline using the Ollama framework for local AI inference.

The primary models used in the system are:

- `qwen2.5:3b` for chatbot response generation
- `minicpm-v` for schematic image understanding
- `nomic-embed-text` for vector embedding generation
- `faster-whisper` for offline speech recognition

Running these models locally ensures reduced latency, improved privacy, offline accessibility, and independence from cloud-based AI services.

# Chapter 4

# Implementation and Development Work

## 4.1 Initial Environment Setup and Analysis

The initial phase of the project involved setting up the eSim Copilot environment on Ubuntu 22.04 running under WSL2. During installation and testing, multiple dependency conflicts, compatibility issues, and runtime errors were identified which prevented successful execution of the chatbot system.

A detailed analysis of the installation workflow, package dependencies, and application startup behaviour was carried out to identify the root causes of these issues. The development work primarily focused on stabilizing the chatbot infrastructure, modernizing outdated packages, improving deployment automation, and ensuring reliable execution on modern Linux environments.

## 4.2 Dependency Resolution and Package Management

Several package conflicts and missing dependencies were identified in the existing `requirements.txt` configuration. These issues prevented proper execution of OCR modules, image-processing pipelines, and chatbot components.

### 4.2.1 PaddleOCR Dependency Fixes

During initialization of PaddleOCR, the following errors were observed:

```
[INIT] PaddleOCR init failed: No module named 'pyclipper'
[INIT] PaddleOCR init failed: No module named 'skimage'
```

The issue occurred because required dependencies were missing from the existing requirements configuration.

The following packages were added:

```
pyclipper
scikit-image
```

This fix enabled successful initialization of PaddleOCR and restored image-based OCR functionality.

### 4.2.2 OpenCV Compatibility Resolution

A version conflict between PaddleOCR and OpenCV caused repeated package installation failures.

Observed error:

```
paddleocr requires opencv-python<=4.6.0.66
```

The existing OpenCV package was replaced with a compatible headless version:

```
opencv-python-headless==4.6.0.66
```

Additionally, conflicting OpenCV packages were automatically removed inside `setup.sh`:

```
pip uninstall -y opencv-python \
opencv-contrib-python \
opencv-python-headless || true
```

This resolved OCR-related dependency conflicts successfully.

### 4.2.3 scipy Version Conflict Resolution

The `requirements.txt` file contained duplicate entries for `scipy`, causing unexpected package upgrades and dependency instability.

The duplicate unpinned entry was removed and only the required version was retained:

```
scipy==1.10.1
```

This stabilized package resolution during installation.

### 4.2.4 Removal of Unnecessary setuptools Pinning

The original project configuration pinned:

```
setuptools==65.5.0
```

This restriction was no longer required after modernization of HDLParse. The pinned dependency was removed to improve compatibility with modern Python environments.

## 4.3 HDLParse Modernization and Python 3 Compatibility

The original `hdlparse==1.0.4` package available on PyPI had not been maintained since 2017 and failed to install on modern Python versions.

Observed error:

```
error in hdlparse setup command:
use_2to3 is invalid
```

The issue occurred because the package relied on the deprecated `use_2to3` feature removed from newer versions of setuptools.

### 4.3.1 Python 2 to Python 3 Migration

To resolve the issue, the original HDLParse repository was forked and modernized for Python 3 compatibility.

The migration process included:

- Running the `2to3` migration tool on the source code
- Removing deprecated `use_2to3` configuration from `setup.py`
- Adding explicit Python version compatibility support
- Cleaning outdated build configurations

The conversion process was performed using:

```
python3.10 -m lib2to3 -w hdlparse/
```

The updated package configuration included:

```
python_requires='>=3.6'
```

### 4.3.2 Integration of Updated HDLParse Package

The outdated PyPI dependency:

```
hdlparse==1.0.4
```

was replaced with the updated GitHub fork:

```
hdlparse @ git+https://github.com/
nehadhumal-dev/hdlparse.git
```

This eliminated the need for legacy setuptools workarounds and enabled clean installation on modern Python environments.

## 4.4 setup.sh Automation and Deployment Improvements

The original installation workflow required multiple manual steps, including dependency installation, environment setup, model configuration, and knowledge-base initialization. These requirements made the onboarding process time-consuming and increased the likelihood of installation failures for new contributors.

To improve deployment reliability and simplify the setup experience, the installation script was extensively redesigned and enhanced. The updated `setup.sh` script automated several previously manual operations and introduced additional validation mechanisms to ensure a more reproducible development environment.

The enhanced script introduced the following capabilities:

- Automatic installation of required Ubuntu system packages.
- Creation and configuration of a dedicated Python virtual environment.
- Installation and validation of Python dependencies required by eSim Copilot.
- Automated installation and configuration of Ollama.
- Automatic downloading of language, vision, and embedding models.
- Installation of speech-recognition resources used by the chatbot.
- Automated generation and validation of the ChromaDB knowledge base.
- Detection of existing resources to avoid unnecessary downloads and reprocessing.
- Improved error handling, logging, and installation feedback.
- Simplified setup workflow for new contributors and developers.

These improvements significantly reduced manual intervention during installation and improved consistency across different development environments.

### 4.4.1 Ollama Auto-Installation

Previously, if Ollama was not installed, the setup process displayed a warning and skipped model configuration, resulting in an incomplete AI environment.

To eliminate this dependency on manual installation, automatic Ollama installation was integrated directly into the setup workflow using:

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

This ensured that the required local inference framework was available before proceeding with model configuration and chatbot initialization.

### 4.4.2 Ollama API Readiness Handling

During testing of fresh installations, model downloads occasionally failed because the Ollama service was not fully initialized immediately after installation.

Observed error:

```
Error: could not connect to ollama server
```

To address this issue, a readiness-check mechanism was introduced before model downloads were initiated.

```
for i in {1..10}; do
curl -s http://localhost:11434 \

> /dev/null && break
> sleep 3
> done
>
```

This validation ensured that the Ollama service was operational before attempting model downloads, improving installation reliability and preventing intermittent setup failures.

### 4.4.3 Automated Model and Resource Configuration

The setup workflow was further enhanced to automatically download and configure all resources required by eSim Copilot. This included language models, embedding models, speech-recognition resources, and Retrieval- Augmented Generation (RAG) dependencies.

Automating these tasks eliminated several manual configuration steps and ensured that a fully functional chatbot environment was available immediately after installation.

## 4.5 ChromaDB and RAG Initialization Fixes

During testing, the chatbot produced empty OCR responses and failed to generate contextual answers.

Observed logs:

```
[EMBED ERROR] Failed to connect to Ollama
WARNING: No valid chunks found
```

Root-cause analysis revealed that `ingest.py` was never executed during installation, leaving ChromaDB completely empty.

### 4.5.1 Automated ingest.py Execution

The setup script was modified to automatically initialize the RAG database after model installation.

```
if [ -f "src/ingest.py" ]; then
python src/ingest.py
fi
```

A guard condition was also added to prevent unnecessary re-ingestion when the database already existed.

This fix restored semantic retrieval and contextual chatbot responses.

### 4.5.2 Automated ingest.py Execution

The setup script was modified to automatically initialize the RAG database after model installation.

```
if [ -f "src/ingest.py" ]; then
python src/ingest.py
fi
```

A guard condition was also added to prevent unnecessary re-ingestion when the database already existed.

This fix restored semantic retrieval and contextual chatbot responses.

### 4.5.3 Knowledge Base Automation Improvements

The Retrieval-Augmented Generation (RAG) setup process was further improved to reduce manual configuration requirements.

The installation workflow was enhanced to automatically verify the existence of the ChromaDB database before executing the ingestion process.

The improvements included:

- Automatic detection of an existing knowledge base.
- Conditional execution of the ingestion pipeline.
- Prevention of unnecessary document reprocessing.
- Faster installation and setup times.

These changes improved installation reliability and simplified deployment for new contributors.

### 4.5.4 Netlist Contract File Handling

The chatbot failed to locate the netlist contract file because the file was generated in a different directory than expected.

Observed error:

```
Could not load netlist contract
```

The following file-copy step was added:

```
mkdir -p src/frontEnd/manuals
cp src/manuals/
esim_netlist_analysis_output_contract.txt \
src/frontEnd/manuals/
```

This restored proper netlist-analysis functionality.

## 4.6 GUI and Platform Compatibility Improvements

### 4.6.1 Wayland and Qt Platform Compatibility

The application initially froze completely on systems using the Wayland display server because Qt defaulted to the XCB backend.

To resolve this issue, automatic platform detection was added:

```
if [ -n "$WAYLAND_DISPLAY" ]; then
export QT_QPA_PLATFORM=wayland
else
export QT_QPA_PLATFORM=xcb
fi
```

This enabled stable execution on both Wayland and X11 environments.

## 4.7 Project Workspace and File Handling Fixes

During testing of eSim source code, project creation repeatedly failed with a `FileNotFoundError`.

Observed error:

```
/home/user/eSim-Workspace\n/.projectExplorer.txt
```

The issue occurred because the workspace path read from `workspace.txt` contained a trailing newline character.

The issue was resolved by modifying:

```
workspace = f.read()
```

to:

```
workspace = f.read().strip()
```

This fix restored correct project creation functionality.

## 4.8 Documentation and Developer Support Improvements

Several improvements were made to project documentation in order to simplify onboarding for future contributors.

The documentation work included:

- Updating installation instructions
- Adding troubleshooting steps
- Creating PR testing guidelines
- Documenting known issues and fixes
- Creating setup validation notes

These improvements reduced setup complexity and improved developer experience for future contributors.

The final setup successfully completed without manual intervention, confirming stability and reproducibility of the updated installation workflow. Furthermore, detailed installation workflows were documented for both automated and manual deployment methods, enabling reproducible setup across different development environments. Contributor guidelines were also enhanced to include repository synchronization, branch management, and pull request submission procedures. Additional effort was devoted to documenting chatbot-specific development workflows, RAG knowledge-base generation, and troubleshooting techniques to simplify future maintenance and feature development.

## 4.9 Repository Maintenance and Contributor Workflow Improvements

To improve onboarding and long-term maintainability, several repository-level improvements were implemented.

### 4.9.1 Chatbot Module Restoration

During testing of the chatbot branch, several chatbot-related files and supporting modules were found to be missing or improperly integrated.

The affected files were restored and reintegrated into the chatbot workflow. Import dependencies, initialization routines, and retrieval components were verified and corrected.

This restoration ensured successful chatbot startup and improved overall system reliability.

### 4.9.2 Developer Workflow Documentation

Additional documentation was created to help future contributors set up development environments and contribute more efficiently.

The documentation improvements included:

- Installation procedures.
- Automated setup instructions.
- Manual installation procedures.
- Branch management workflow.
- Pull request submission guidelines.
- Troubleshooting documentation.

These improvements simplified contributor onboarding and reduced setup-related issues.

## 4.10 Retrieval-Augmented Generation (RAG) Integration Improvements

During the final phase of development, additional work was carried out to improve the integration between the chatbot system and the Retrieval- Augmented Generation (RAG) pipeline.

The objective was to ensure that user queries were consistently enriched with relevant contextual information retrieved from the knowledge base before response generation.

### 4.10.1 Knowledge Base Retrieval Enhancements

Several improvements were implemented to strengthen document retrieval and knowledge-base utilization.

The improvements included:

- Validation of ChromaDB initialization before retrieval operations.
- Improved document-loading mechanisms for knowledge-base content.
- Better handling of missing or incomplete embeddings.
- Verification of successful retrieval before response generation.
- Improved integration between retrieval modules and chatbot response workflows.

These changes improved the reliability and consistency of context-aware responses generated by the chatbot.

### 4.10.2 Chatbot and RAG Workflow Integration

Additional testing revealed integration issues between the chatbot components and the retrieval pipeline. Several chatbot modules and supporting files were reviewed and reintegrated to ensure correct interaction with the RAG system.

The integration process involved:

- Restoring missing chatbot-related modules.
- Updating import dependencies and initialization routines.
- Verifying communication between retrieval components and chatbot interfaces.
- Testing end-to-end retrieval and response generation workflows.

These improvements ensured that retrieved knowledge was correctly passed to the language model, enabling more accurate and context-aware responses.

# Chapter 5

# Testing and Validation

## 5.1 Testing Environment

The complete testing process was performed on Ubuntu 22.04 running under Windows Subsystem for Linux 2 (WSL2). Testing was conducted using clean virtual environments to ensure reproducibility and stability of the installation workflow.

The testing environment included:

- Ubuntu 22.04 (WSL2)
- Python 3.10
- Conda and virtual-environment support
- Ollama local inference framework
- PyQt5-based eSim environment

The primary objective of testing was to verify stable installation, dependency compatibility, chatbot initialization, and successful integration with the eSim environment.

## 5.2 Dependency Installation Testing

Dependency testing was performed after updating the `requirements.txt` configuration.

The testing process verified:

- Successful installation of PaddleOCR dependencies
- OpenCV compatibility with OCR modules
- Removal of scipy version conflicts
- Correct installation of embedding and AI-related libraries

The updated dependency configuration successfully completed installation without package-resolution failures.

## 5.3 HDLParse Compatibility Testing

The updated HDLParse package was tested on modern Python environments to verify successful installation and execution.

Testing confirmed:

- Removal of `use_2to3` installation errors
- Successful Python 3 compatibility
- Correct package installation using pip
- Stable HDL parsing functionality

The modernized package installed successfully without requiring legacy setuptools downgrades.

## 5.4 setup.sh Automation Testing

The updated `setup.sh` script was tested multiple times using fresh installations.

Testing verified:

- Automatic environment validation
- Successful Ollama installation
- Correct model downloads
- Internet and Python-version detection
- Automatic execution of `ingest.py`
- Proper error handling and logging

The installation process completed successfully without requiring manual intervention.

## 5.5 ChromaDB and RAG Validation

The RAG pipeline was tested after automatic execution of `ingest.py`.

Testing confirmed:

- Successful ChromaDB initialization
- Proper document chunk generation
- Embedding creation using `nomic-embed-text`
- Semantic retrieval of documentation content
- Context-aware chatbot responses

The chatbot successfully generated responses using retrieved eSim documentation instead of producing empty outputs.

## 5.6 GUI and Platform Compatibility Testing

GUI testing was performed on systems using both Wayland and X11 display servers.

The testing process verified:

- Correct Qt platform detection
- Stable application startup
- Elimination of GUI freezing issues
- Proper chatbot window integration
- Responsive PyQt5 interface behaviour

The updated platform-detection logic enabled reliable execution on both display-server environments.

## 5.7 Project Creation Workflow Testing

Testing was performed after fixing the workspace path handling bug in eSim.

The following workflow was verified:

- Creation of new projects
- Generation of project directories
- Workspace file handling
- Project explorer initialization
- Correct path generation without newline issues

Project creation functionality worked successfully after applying the workspace path fix.

## 5.8 End-to-End System Validation

Final validation was performed using a completely fresh installation workflow.

The validation process included:

- Deleting the previous environment
- Cloning the repository again
- Running `bash setup.sh`
- Launching eSim Copilot
- Testing OCR functionality

- Testing RAG-based responses
- Testing chatbot startup and integration

The entire system executed successfully without installation failures or runtime crashes.

## 5.9 Testing Summary

The testing process confirmed that the implemented fixes successfully improved the stability, compatibility, and deployment workflow of the Offline AI Chatbot system integrated within eSim.

The updated setup process reduced installation complexity, improved cross-platform compatibility, and enabled reliable execution in modern Python and Linux environments.

# Chapter 6

# Developer Guide and Installation Manual

This chapter provides a step-by-step guide for developers to set up the eSim development environment, install dependencies, and run the eSim Copilot system.

## 6.1 Prerequisites

Before proceeding with the installation, ensure that the following software is installed on your system:

- Ubuntu 22.04 LTS or later
- Git
- Python 3.10 or later
- pip (Python Package Manager)
- Internet connection for downloading dependencies

## 6.2 Forking the Official Repository

To contribute to eSim, first create a personal fork of the official repository.

1. Visit the official eSim repository:
   “‘ `https://github.com/FOSSEE/eSim`
2. Click the **Fork** button in the top-right corner of the GitHub page.
3. Select your GitHub account as the destination for the fork.
4. GitHub will create a copy of the repository under your account. “‘

## 6.3 Cloning the Forked Repository

After creating the fork, clone your forked repository to your local machine.

```
git clone https://github.com/<your-username>/eSim.git
cd eSim
```

Replace `<your-username>` with your GitHub username.

## 6.4 Adding the Upstream Repository

Configure the original eSim repository as an upstream remote to keep your fork synchronized.

```
git remote add upstream https://github.com/FOSSEE/eSim.git
git remote -v
```

The second command verifies that both `origin` and `upstream` remotes have been added successfully.

## 6.5 Switching to the Chatbot Development Branch

The eSim Copilot development work is maintained in the `eSim-Chat-Bot-Semester-Long-Internshi` branch. After cloning the repository and configuring the upstream remote, switch to this branch before beginning development.

```
git fetch upstream
git checkout eSim-Chat-Bot-Semester-Long-Internship
git pull upstream eSim-Chat-Bot-Semester-Long-Internship
```

This ensures that the local repository contains the latest chatbot-related changes.

## 6.6 Creating a Personal Development Branch

Developers should create a separate branch for their work instead of directly modifying the chatbot branch.

```
git checkout -b chatbot-developer-branch
```

Example:

```
git checkout -b rag-improvements
```

Using a dedicated branch helps maintain code isolation and simplifies code review during the pull request process.

## 6.7 Automated Installation

To simplify the setup process, eSim provides an automated installation script named`setup_copilot_u` located in the scripts directory.

```
cd scripts
chmod +x setup_copilot_ubuntu.sh
./setup_copilot_ubuntu.sh
```

The script automatically performs the following tasks:

- Installs required Ubuntu system packages.
- Creates a Python virtual environment.
- Installs Python dependencies.
- Installs HDLParse and related packages.
- Installs PaddlePaddle.
- Installs Ollama.
- Downloads required language and embedding models.
- Downloads the Vosk speech recognition model.
- Generates the RAG knowledge base using ChromaDB.

## 6.8 Manual Installation

If the automated installation script cannot be executed, the setup can be performed manually by following each installation step individually.

### 6.8.1 Installing System Dependencies

```
sudo apt-get update
sudo apt-get install -y python3.10 python3.10-venv python3-pip
sudo apt-get install -y curl wget unzip
sudo apt-get install -y ngspice kicad
```

### 6.8.2 Creating a Virtual Environment

```
python3.10 -m venv .venv
source .venv/bin/activate
```

### 6.8.3 Installing Python Dependencies

```
pip install --upgrade pip wheel
pip install setuptools==57.5.0
pip install hdlparse==1.0.4 --no-build-isolation
pip install -r requirements.txt
pip install -r requirements-copilot.txt
```

### 6.8.4 Installing Ollama Models

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

### 6.8.5 Generating the Knowledge Base

```
cd src
python ingest.py
```

This command creates the ChromaDB knowledge base used by the Retrieval-Augmented Generation (RAG) pipeline.

## 6.9 Running eSim Copilot

After completing the installation, activate the virtual environment and navigate to the frontend directory.

```
cd eSim
source .venv/bin/activate
cd src/frontEnd
```

Launch the eSim application using:

```
python Application.py
```

Once the application starts successfully, the eSim graphical user interface (GUI) will be displayed. The integrated eSim Copilot can then be accessed from the application interface.

## 6.10 Verifying the RAG Knowledge Base

The eSim Copilot uses a Retrieval-Augmented Generation (RAG) pipeline backed by ChromaDB.

The knowledge base is automatically generated during execution of the setup script. The generated database is stored in:

```
eSim/src/chroma_db
```

If the database is not available, it can be created manually using:

```
cd eSim/src
source ../.venv/bin/activate
python ingest.py
```

Successful execution indicates that the documentation and knowledge sources have been processed and stored in the vector database.

## 6.11 Development Workflow

After installation, developers can begin implementing new features, bug fixes, or enhancements.

### 6.11.1 Synchronizing with the Latest Changes

Before starting development, synchronize the local repository with the latest updates from the upstream repository.

```
git fetch upstream
git checkout eSim-Chat-Bot-Semester-Long-Internship
git pull upstream eSim-Chat-Bot-Semester-Long-Internship
```

This minimizes the likelihood of merge conflicts and ensures development is based on the most recent code.

### 6.11.2 Implementing and Testing Changes

After creating a personal development branch, make the required code modifications.

The application should be tested locally before committing changes.

```
source .venv/bin/activate
cd src/frontEnd
python Application.py
```

Verify that the application launches correctly and that the implemented functionality behaves as expected.

### 6.11.3 Committing Changes

After completing development and testing, stage and commit the modified files.

```
git add .
git commit -m "Describe the implemented changes"
```

Example:

```
git commit -m "Improve RAG integration and restore chatbot modules"
```

### 6.11.4 Pushing Changes to GitHub

Push the development branch to the forked repository.

```
git push origin chatbot-developer-branch
```

Replace `chatbot-developer-branch` with the actual branch name used during development.

## 6.12 Creating a Pull Request

After pushing the changes, navigate to the forked repository on GitHub.

1. Open the forked repository.
2. Click **Compare & Pull Request**.
3. Provide a descriptive title.
4. Add a detailed explanation of the implemented changes.
5. Select `eSim-Chat-Bot-Semester-Long-Internship` as the target branch.
6. Submit the Pull Request for review.

After submission, project maintainers review the proposed changes. Additional modifications may be requested before the pull request is merged.

## 6.13 Troubleshooting

- Ensure that the Python virtual environment is activated before executing any Python commands.
  ‘‘‘
- Verify that all required dependencies have been installed successfully.
- Confirm that Ollama is running and the required models have been downloaded.
- Ensure that the ChromaDB knowledge base has been generated successfully.
- Synchronize the fork regularly to avoid merge conflicts.
- Review terminal logs and error messages for debugging information. ‘‘‘

## 6.14 Summary

This chapter presented the complete developer workflow for eSim Copilot, including repository setup, automated and manual installation procedures, development practices, execution of the application, and contribution through pull requests. Following these steps enables developers to efficiently set up the environment, implement new features, and contribute to the eSim project.

# Chapter 7

# Conclusion and Future Work

## 7.1 Conclusion

This work focused on improving the stability, compatibility, and deployment workflow of the Offline AI Chatbot integrated within the eSim environment. During the development and testing process, multiple dependency conflicts, installation failures, runtime issues, and platform-specific compatibility problems were identified and resolved.

Major improvements included modernization of the outdated HDLParse package for Python 3 compatibility, dependency resolution for PaddleOCR and OpenCV, automation of the installation workflow through an enhanced `setup.sh` script, automatic RAG database initialization, Wayland compatibility fixes, and resolution of workspace path handling issues affecting project creation.

The deployment workflow was significantly improved by automating Ollama installation, model setup, dependency validation, and environment configuration. These changes reduced manual setup complexity and improved reproducibility for new contributors and developers.

The implemented fixes also improved the overall reliability of the chatbot infrastructure, enabling stable OCR processing, semantic retrieval, chatbot initialization, and integration with the PyQt5-based eSim environment.

Overall, the project successfully enhanced the maintainability and usability of the Offline AI Chatbot system while providing a more stable and developer-friendly setup process for future development.

## 7.2 Future Work

Although the current system provides stable offline AI assistance within eSim, several enhancements can be implemented in future versions to further improve functionality and user experience.

Possible future improvements include:

- Integration of more advanced local language models for improved technical reasoning and faster response generation.

- Support for additional HDL formats such as SystemVerilog and VHDL.
- Enhanced schematic understanding using improved vision-language models and advanced component-connection analysis.
- Improved context-aware debugging support for NgSpice simulation errors and waveform analysis.
- Cross-platform optimization and testing for native Windows and macOS environments.
- Automated update and dependency-management systems for easier maintenance and deployment.
- Expansion of the RAG knowledge base using additional tutorials, documentation, and troubleshooting examples.
- Improved GUI interaction with richer chatbot visualizations and interactive debugging support.

The continued development of offline AI-assisted EDA tools can improve learning accessibility, simplify debugging workflows, and support open-source engineering education more effectively.

# Bibliography


[1] FOSSEE Team, IIT Bombay, *eSim: Open Source EDA Tool*. https://esim.fossee.in/

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

[3] ChromaDB, *Open-source Embedding Database*. https://www.trychroma.com/

[4] Baidu Inc., *PaddleOCR Toolkit*. https://github.com/PaddlePaddle/PaddleOCR

[5] SYSTRAN, *faster-whisper*. https://github.com/SYSTRAN/faster-whisper

[6] Alibaba Cloud, *Qwen2.5 Technical Report*, 2024.

[7] OpenBMB, *MiniCPM-V Multimodal Model*, 2024.

[8] KiCad Development Team, *KiCad Documentation*. https://www.kicad.org/documentation/

[9] Ngspice Development Team, *Ngspice User Manual*. https://ngspice.sourceforge.io/

[10] Python Software Foundation, *Python Documentation*. https://www.python.org/