Winter Internship 2024 Report

On

AI-Based Suggestion/Debugging Tool for eSim

Submitted by

Myo Thinzar Kyaw
B.E(Electronics and Communication)- Final Year
Myanmar Institute of Information Technology
Mandalay,Myanmar

Under the guidance of

Prof. Kannan M. Moudgalya
Chemical Engineering Department
IIT Bombay

March, 2025

Acknowledgements

We would like to express our sincere gratitude to the **FOSSEE** team for giving us the opportunity to contribute to the development of the open-source software eSim.

First, we would like to thank **Prof. Kannan M. Moudgalya** for his valuable guidance and support throughout the FOSSEE Winter Internship program. We also extend our thanks to the entire eSim team for their help and for providing the resources needed to complete this project. Special thanks to **Mrs. Usha Viswanathan** and **Mrs. Vineeta Parmar** for their constant support.

We are also very grateful to our mentor, **Mr. Sumanto Kar**, for his continuous support, cooperation, and willingness to share his knowledge, which made our learning experience much more enriching.

We would like to thank our fellow interns as well for making this journey both educational and enjoyable. It was a privilege to work with them under the guidance of such experienced mentors.

Finally, we would like to thank everyone who contributed directly or indirectly to the successful completion of our internship. We hope to apply the lessons learned from this experience to make a positive impact in the future.

Contents

Chapter 1

Introduction

1.1 eSim

FOSSEE (Free/Libre and Open Source Software for Education) encourages the adoption of open-source software in educational settings, with eSim being a leading project. eSim is a powerful open-source Electronic Design Automation (EDA) tool for circuit design, simulation, and PCB design.

1.2 AI Based Suggestion/Debugging Tool

The tool improves debugging in eSim, a free and open-source circuit simulation tool. Users often struggle with complex error messages and unconnected netlist components, making troubleshooting difficult. To address this, I developed an AI-powered tool that analyzes error logs to detect issues like missing connections or incompatible components. It will also parse simulation error logs to suggest fixes, such as correcting invalid parameters or resolving missing libraries. Additionally, an AI chat bot will provide real-time troubleshooting, offering tailored solutions.

1.3 Motivation for Developing the Tool

Debugging electronic circuits in eSim, a free and open-source EDA tool, can be complex and time-consuming, especially for students, educators, and researchers. Issues like unclear error messages, unconnected components, and simulation failures often hinder productivity and slow down the design process. This project aims to simplify debugging by integrating AI and machine learning for real-time error detection and correction, enabling users to focus on innovation rather than troubleshooting. Additionally, it supports the broader goal of promoting open-source tools in education and research, providing a cost-effective alternative to proprietary software.

1.4 Objectives

The objectives of AI Based Suggestion/Debugging Tool for eSim project are as follows:

- Develop an AI-powered tool to help eSim users identify and resolve common circuit design and simulation issues.

- Detect error in circuit, such as missing connections and incompatible components, through automated analysis.
- Analyze simulation error logs to provide actionable fixes for issues like invalid parameters or missing libraries.
- Implement an AI-powered chatbot for real-time troubleshooting, offering tailored guidance based on error logs.
- Design a user-friendly interface for seamless interaction and easy access to debugging suggestions.
- Enhance eSim's usability to promote open-source accessibility and encourage adoption in education and research as a cost-effective alternative to proprietary EDA tools.

Chapter 2

Problem Statement

Debugging circuit simulations in eSim can be a challenging and time-consuming task, as users must manually analyze complex error logs to identify issues. Common problems such as missing libraries, invalid parameter values, and incorrect netlist configurations can be difficult to diagnose, leading to delays and inefficiencies in the design process. Existing rule-based methods offer limited assistance, lacking adaptability to new or uncommon errors. To streamline debugging, an AI-powered tool is needed to automate error log analysis, provide intelligent troubleshooting suggestions, and optionally integrate with an AI chatbot for interactive assistance. By leveraging machine learning and natural language processing, this tool will enhance debugging efficiency, reduce manual effort, and improve the overall user experience for circuit designers.

The requirements for the system are as follows:

- **Error Log Analysis:**
  - Parse eSim error logs from ngspice simulations.
  - Classify the error types.
  - Suggest corrective actions to fix error.
- **AI-Powered Debugging:**
  - Use pattern recognition model to detect error
  - Give concise suggestion to resolve common circuit issues.
- **Interactive Chatbot Integration::**
  - Implement a interactive chat bot for circuit troubleshooting
  - Provide a learning assistance of users

The above requirements aim to provide an efficient and user-friendly tool for faster debugging, better user experience, learning assistance for beginners.

Chapter 3

System Architecture

3.1 Overview of System Components

This project features two primary windows within the eSim environment to address circuit simulation errors and provide interactive AI-driven support:

1. **AI Chatbot Window (Using Qwen 2.5 code 3b via *ollama*)**
2. **Debugging Tool Window (Using LSTM Model and Rule-Based Engine)**

3.1.1 AI Chatbot Window

- **Purpose**:
  - Offers an interactive, natural language interface for general troubleshooting and user queries.
  - Leverages a pre-trained Qwen 2.5 code 3b model (hosted locally via *ollama*) for conversational responses.
- **Key Features**:
  - **Context-Aware Assistance**: The chatbot provides on-demand suggestions or clarifications related to eSim usage and circuit design.
  - **Rapid Guidance**: Users can query the chatbot for common issues (e.g., simulation setup, parameter configuration) and receive step-by-step help.
  - **User-Friendly Interface**: A dedicated window inside eSim displays the chatbot conversation in a streamlined chat layout.

3.1.2 Debugging Tool Window

- **Purpose**:
  - Automates detailed error analysis for simulation logs generated by eSim (e.g., Ngspice outputs).
  - Generates precise debugging suggestions through an LSTM model and a rule-based engine.

- **LSTM-Based Error Analysis**:
  - **Error Pattern Recognition**: Tokenizes raw simulation logs and transforms them into vector embeddings, which are then fed into an LSTM model to classify complex and recurring error types.
  - **Automated Fixes**: The model maps recognized patterns (e.g., missing library references, invalid parameters) to corrective actions.
- **Rule-Based Engine for Ngspice Conversion**:
  - **Structured Checks**: Applies predefined rules to spot typical issues in KiCad-to-Ngspice netlist conversion (e.g., missing model statements, unsupported syntax).
  - **Direct Fix Suggestions**: For each recognized issue, the rule-based system provides recommended steps (e.g., add missing '.lib' file, correct netlist parameter).
- **User Interface**:
  - **Dedicated Debugging Window**: Displays parsed error logs, classification outcomes, and recommended fixes in a single view when user clicks debug button.
  - **Automatic Updates from ChatBot Window**: Whenever the user runs a new simulation, logs are automatically parsed to display fresh error diagnostics in real time from chat bot.

In this architecture, the **AI Chatbot Window** focuses on user interaction through natural language (utilizing the Qwen 2.5 code 3b model), while the **Debugging Tool Window** automates error classification and rule-based checks for Ngspice conversion. By separating these functionalities into two dedicated windows, users benefit from both an intelligent conversational assistant and a specialized diagnostic interface, significantly streamlining circuit simulation troubleshooting in eSim.

3.2 Workflow Diagram

The following diagram illustrates the high-level architecture of the system, showing how different components interact with each other.

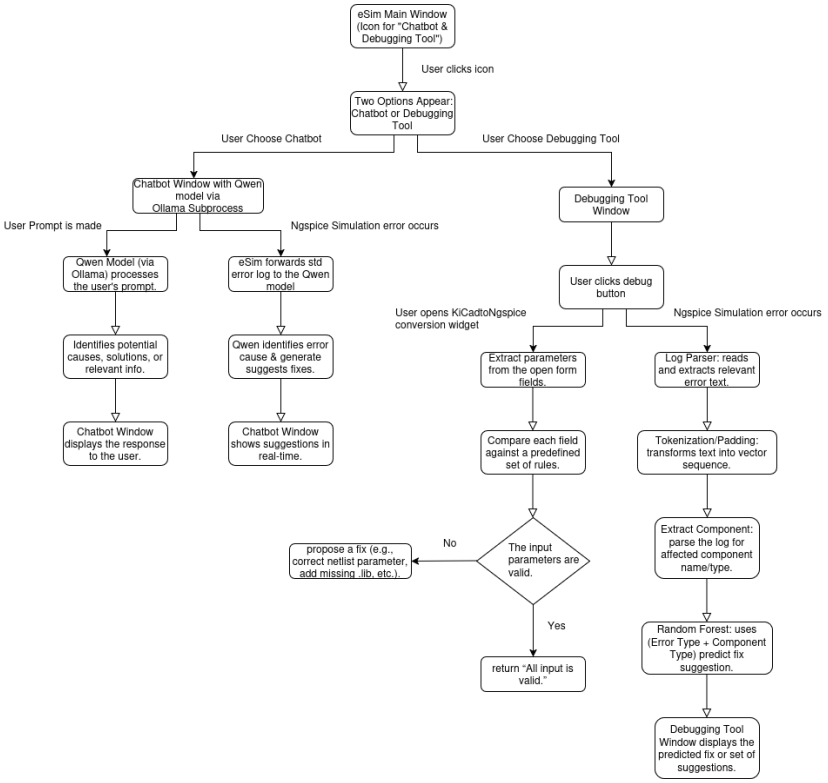

Figure 3.1: System Workflow

3.3 Data Flow and Interaction Between Components

The interaction between different components is structured in a way that ensures seamless data exchange and processing for circuit debugging. The flow of data occurs as follows:

1. A user launches **eSim** and encounters the icon for the combined "Chatbot & Debugging Tool."
2. When the icon is clicked, the user is presented with two options:
   - **Chatbot Window** (running Qwen 2.5 code3b via Ollama).
   - **Debugging Tool Window** (employing an LSTM model, a Random Forest, and rule-based checks).
3. If the user selects the **Chatbot Window**:
   - Any simulation error logs (from Ngspice) are forwarded to the Chatbot via **std error streams**.
   - The Qwen 2.5 model processes these logs or user-entered prompts to identify likely causes and generate fix suggestions.
   - Responses are passed back to the Chatbot Window for real-time display.
4. If the user selects the **Debugging Tool Window** and an Ngspice simulation error has occurred:
   - The **Log Parsing Engine** extracts relevant text from the error log.

- Parsed text is tokenized, padded, and fed into the **LSTM Model**, which classifies the error type.
- A **Random Forest Model** combines the classified error type and any detected component information to suggest a specific fix.
- The recommended fix is then displayed in the Debugging Tool Window.

5. If the user selects the **Debugging Tool Window** to validate KiCad → Ngspice inputs:
   - The tool extracts input values from the open widget in eSim.
   - These values are checked against a predefined **Rule-Based** system (e.g., verifying parameter ranges, required libraries).
   - If any rule is violated, a fix suggestion is presented; otherwise, it confirms "All inputs are valid."
6. Once the user has obtained **fix suggestions** (either from the Chatbot or Debugging Tool), they can apply corrections in eSim and **rerun the simulation** to see if the issue is resolved.
7. At any point, the user may **close** the Chatbot or Debugging Tool Window and return to the main eSim interface. The entire process can be repeated as needed until all errors are resolved.

3.4 User Workflow

The following diagram represents the step-by-step process of user interaction with the AI based Debugging Tool.

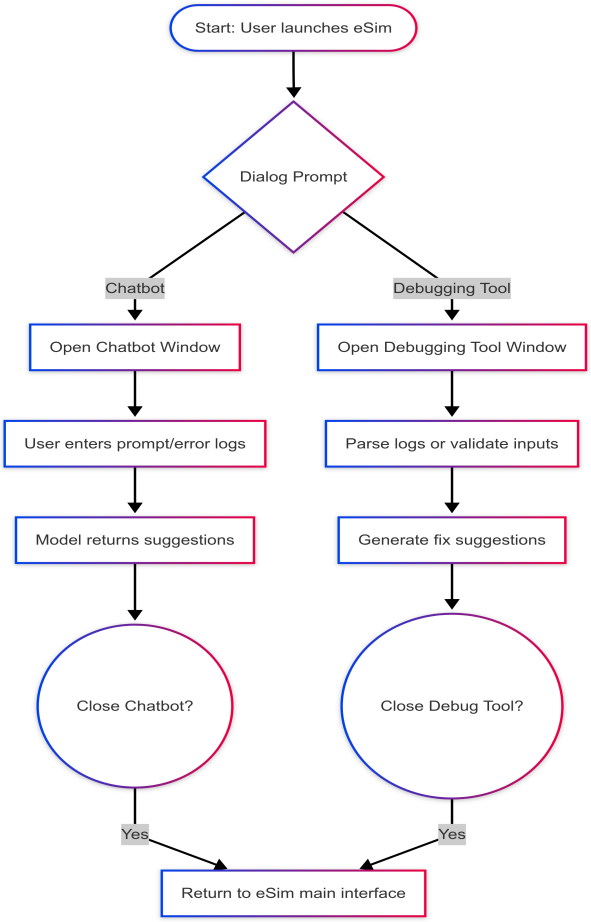

Figure 3.2: User Workflow

3.5 Core Concepts

3.5.1 Error Log Analysis

The project automates error log analysis by parsing eSim simulation logs, identifying error types (e.g., missing libraries, invalid parameters), and suggesting corrective actions using Python libraries and machine learning model for text classification.

3.5.2 Debugging Assistance

The tool integrates an AI chat bot (Qwen 2.5 coder:3B model) for interactive circuit troubleshooting, offering customized debugging suggestions, and includes a machine learning model trained on common circuit issues for additional learning-based recommendations.

3.6 Technologies Used

3.6.1 Python

Python is a high-level, interpreted programming language known for its simplicity, readability, and versatility. Python is widely used across various fields, including web develop-

ment, data science, artificial intelligence, machine learning, and automation.Key features of Python include its beginner-friendly syntax, cross-platform compatibility, and extensive library support, with popular libraries like NumPy, pandas, TensorFlow, Flask, and Django catering to diverse applications. Python is also known for its active community, which provides a wealth of resources, tutorials, and tools for develope

3.6.2 Machine Learning

Machine Learning (ML) is a type of artificial intelligence (AI) that allows computers to learn from data and make decisions without being directly programmed. It uses algorithms like supervised learning for tasks such as classification and regression, unsupervised learning for clustering, and reinforcement learning for decision-making. ML is used in applications like image recognition, language translation, and predictive analytics. Its key benefits are automation, scalability, and the ability to improve over time, though it requires quality data and significant computational power. Machine learning is a key driver of modern technology and innovation.

3.6.3 Natural Language Processing

Natural language processing (NLP) techniques such as text classification, tokenization, and sequential learning play a key role in analyzing error logs. Text classification helps categorize error messages into types (e.g., syntax, logic, runtime errors), enabling the tool to quickly identify the nature of the problem. Tokenization breaks down error logs into smaller units, such as keywords and error codes, making it easier to pinpoint specific issues. Sequential learning, using technique LSTM, allows the tool to analyze the flow of error messages over time, recognizing patterns and predicting future errors based on previous log sequences. Together, these NLP techniques enhance the tool's ability to understand and provide intelligent solutions to debugging challenges.

3.6.4 Deep Learning

Deep learning is a subset of machine learning that uses neural networks with many layers to analyze complex patterns in data. It mimics the human brain's ability to process information, allowing models to learn directly from raw data without needing manual feature extraction. Deep learning is useful in the AI-based debugging tool because it can automatically identify patterns in large volumes of unstructured error logs, improving error classification and prediction. It is used by training a neural network on historical error data, enabling the tool to classify errors, predict potential issues, and generate solutions based on past patterns.

3.6.5 Qwen 2.5 Coder 3B

Qwen 2.5 Coder 3B is an AI model designed for coding tasks, such as code generation, debugging, and understanding programming concepts. It can assist in automating coding tasks, generating code snippets, and offering solutions for common coding problems. Using this model can improve productivity and reduce manual errors in code development. You can integrate Qwen 2.5 Coder 3B into your project to generate code based on specific requirements, suggest fixes for bugs, or automate parts of the coding process, making it a valuable tool in enhancing your development workflow.

3.6.6 Ollama

Ollama is a platform that provides access to advanced AI models, allowing developers to integrate them into applications for tasks like natural language processing, code generation, and automation. It is useful for leveraging powerful AI capabilities without requiring deep expertise in machine learning, simplifying the integration of AI into software projects. By using Ollama,the tool can provide features such as conversational abilities,debugging assistance functionality.

Chapter 4

Implementation

4.1 Chatbot Window

The Chatbot Window is an integral part of the eSim application, offering users AI-driven assistance for circuit-related questions and simulation debugging. It operates through a locally hosted instance of the Qwen 2.5 code3b model via Ollama, ensuring fast response times and offline capability.

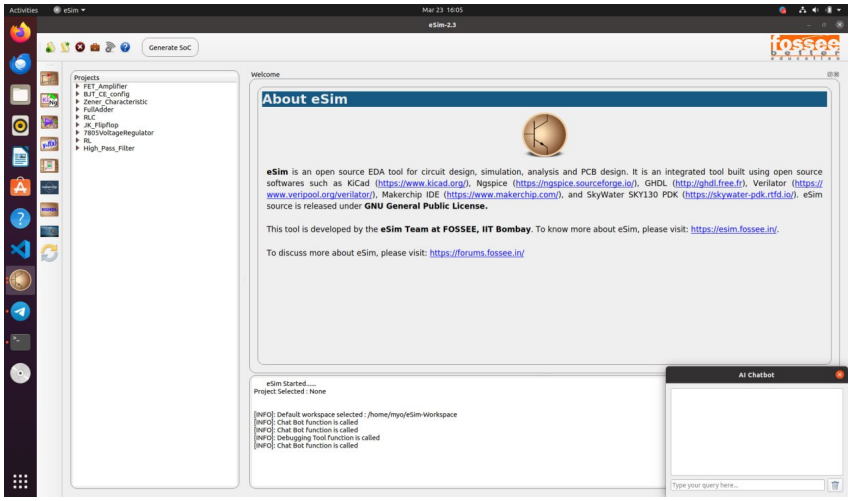

Figure 4.1: Chatbot Window integrated within eSim.

4.1.1 Chatbot Integration

When the user chooses the **Chatbot** option from the eSim main interface, a dedicated window (Figure 4.1) opens to facilitate an interactive troubleshooting session. This integration leverages **Ollama** to run the Qwen 2.5 code3b model locally. The essential steps in this integration are:

1. **Ollama Subprocess Setup**: Upon selecting the Chatbot, eSim spawns a local Ollama subprocess, ensuring that all AI-related computations occur on the user's own machine.

2. **Model Invocation**: The Qwen 2.5 code3b model files reside in the Ollama directory. Once the subprocess is active, user prompts (or error messages) are forwarded to the model.

3. **Response Handling**: Qwen processes the query, then returns a structured text response detailing potential solutions, error explanations, or follow-up questions.

4.1.2 User Interface Layout

The Chatbot Window provides a straightforward interface to streamline user interactions:

- **Conversation Log**: Displays the ongoing dialogue with the chatbot. Each user query and corresponding response from Qwen is shown in a scrollable panel.
- **Input Box**: A text field where users can type questions such as:
  - *"Why is my MOSFET not conducting?"*
  - *"How can I fix a missing library error in Ngspice?"*
- **Send Button**: Initiates the request. Alternatively, the user can press *Enter* to submit.
- **Close/Exit Button**: Allows the user to close the Chatbot Window at any time, returning to the main eSim interface.

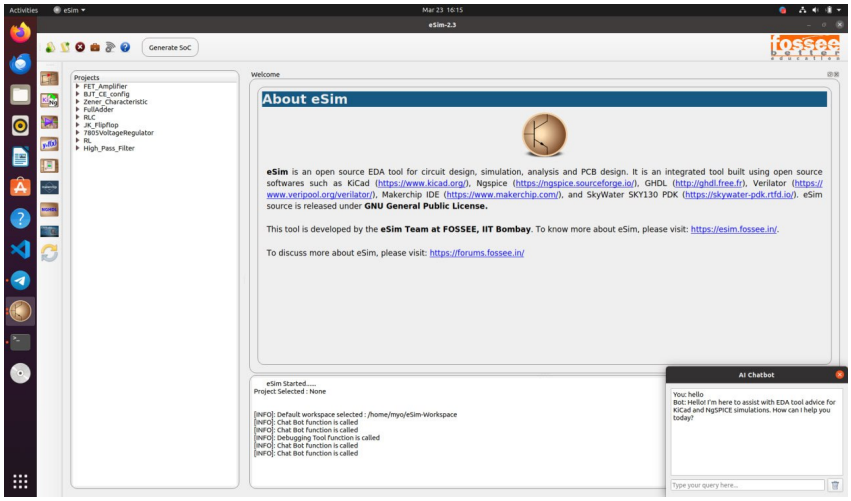

Figure 4.2: Key UI elements within the Chatbot Window.

4.1.3 Chatbot Workflow

Once the Chatbot Window is open, users can pose circuit-related questions or paste error logs. Internally, the following workflow ensures the seamless exchange of data:

1. **Prompt Submission**: The user types a question or error description into the Chatbot Window.
2. **Ollama Communication**: The prompt is passed as a request to the locally running Ollama subprocess.
3. **Qwen Processing**: The Qwen 2.5 code3b model interprets the input using its pre-trained knowledge and any provided error logs.

4. **Suggestion Generation**: Potential causes, suggested fixes, or clarifications are formulated and returned to the Chatbot Window.

5. **User Review & Follow-up**: The user reviews the response. If needed, they can continue the conversation for more details or close the window to apply fixes directly in eSim.

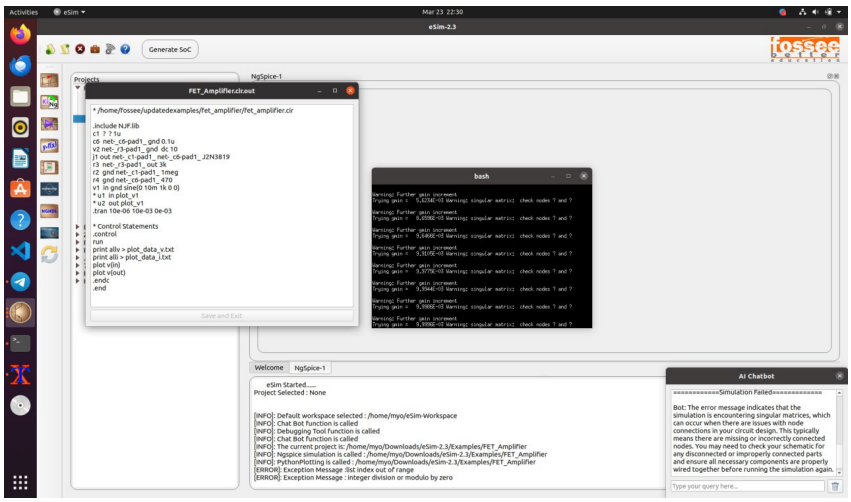

Figure 4.3: Example Chatbot conversation with error logs and suggested fixes.

4.1.4 Handling Circuit-Related Questions and Errors

The Chatbot is capable of:

- **Answering Circuit Theory Queries**: From basic resistor–capacitor questions to more advanced transistor biasing issues.
- **Interpreting Simulation Logs**: When Ngspice generates an error, the Chatbot can analyze the output text, detect keywords (e.g., "Convergence Problem," "No such device or model"), and respond with targeted advice.
- **Guiding Parameter Adjustments**: The Chatbot can suggest valid ranges or default values for common SPICE parameters if they are missing or incorrectly specified.
- **Suggesting Debug Steps**: For instance, recommending that the user verify library paths, check netlist syntax, or run a DC operating point analysis prior to a transient simulation.

4.1.5 Implementation Highlights

1. **Local LLM Runtime**: Leveraging Ollama ensures the large language model (Qwen 2.5 code3b) runs entirely on the local machine, maintaining data privacy.
2. **Error Log Forwarding**: Whenever a simulation error occurs, eSim can directly pipe the `stderr` output to the Chatbot. This reduces manual copy-paste steps.

3. **Real-time Feedback**: The Chatbot Window updates immediately upon receiving new responses, allowing a continuous conversation flow.

4. **Exit and Re-entry**: If users close the Chatbot Window, they can re-open it later for further assistance without losing major context (since Qwen is always running locally).

4.1.6 Benefits of Chatbot Integration

- **Faster Debugging Cycles**: Users get near-instant feedback on common errors, accelerating design iterations.
- **Reduced External Dependencies**: Running the model locally avoids reliance on internet-based APIs.
- **Comprehensive Knowledge Base**: Qwen 2.5 code3b contains domain-specific insights for circuit design, SPICE parameters, and troubleshooting conventions.
- **User-Friendly Interaction**: The natural language interface lowers the barrier for new designers who might not be familiar with advanced SPICE error codes.

4.2 Debugging Tool Window

In addition to the Chatbot functionality, the eSim application integrates a dedicated **Debugging Tool Window** for automated error analysis and rule-based validation of KiCad-to-Ngspice inputs. This tool aids users in quickly diagnosing simulation errors using machine learning (LSTM) and ensuring correct netlist parameters or library references through a rule-based check.

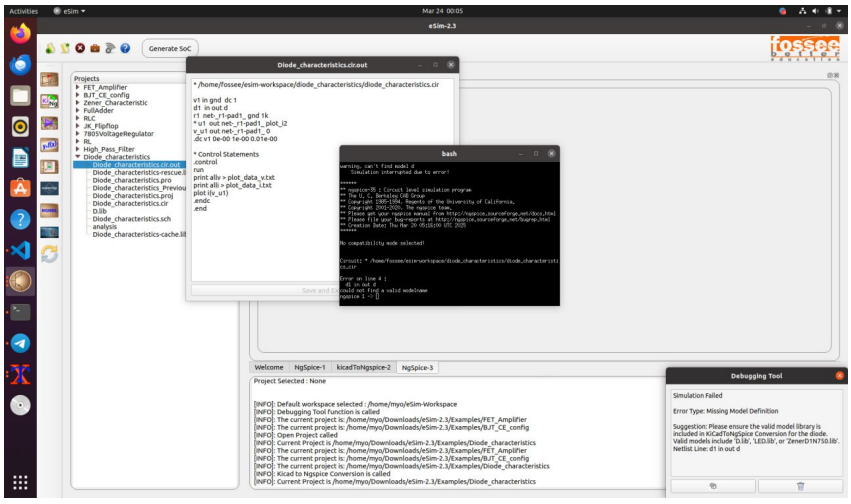

Figure 4.4: Debugging Tool Window in the eSim environment.

4.2.1 Error Analysis (LSTM Model)

When an Ngspice simulation produces errors, users can click on the *Debugging* button within debugging tool window to initiate an automated parsing and classification process:

1. **Log Parsing:** The tool extracts relevant text from the raw `stderr` or log files generated by Ngspice (e.g., missing model references, convergence issues).

2. **Tokenization and Padding:** Extracted text is tokenized and padded to form fixed-length sequences suitable for the LSTM.

3. **LSTM Classification:** A pre-trained LSTM model processes these sequences to determine the error category (e.g., missing library file, invalid parameter, convergence failure).

4. **Component Extraction:** The tool identifies which circuit component (if any) is implicated in the error (e.g., resistor R1, transistor Q2).

5. **Fix Suggestion (e.g., Random Forest):** Optionally, an additional model such as a Random Forest can incorporate the classified error type and component data to predict a specific fix (e.g., "Add missing .lib file," "Increase Vds limit," etc.).

Interface and User Experience

Upon completing the analysis, the tool presents a comprehensive summary in the **Debugging Tool Window**:

- **Parsed Error Summary**: Highlights the critical lines or keywords from the Ngspice log.
- **Classified Error Type**: Informs the user of the probable cause (e.g., "Library Not Found").
- **Suggested Fix**: Lists one or more recommended steps, such as adding a missing library path or correcting a netlist parameter.
- **Apply Fix / Re-run Simulation**: Allows users to quickly modify the circuit or netlist and re-run the simulation.

4.2.2 Rule-Based Validation (KiCad-to-Ngspice Conversion)

The Debugging Tool also supports a **Rule-Based Validation** feature to ensure correct user input when translating designs from KiCad to Ngspice.

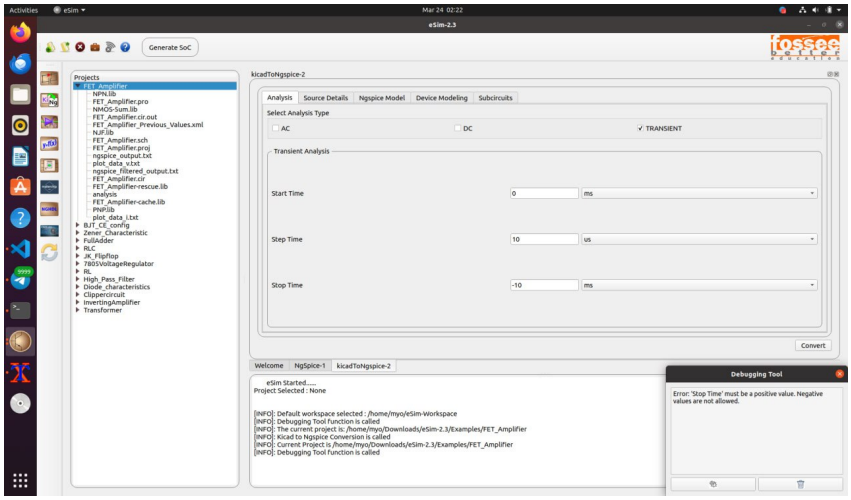

Figure 4.5: Rule-based validation interface for KiCad-to-Ngspice conversions.

Validation Workflow

1. **User Input Extraction**: The tool retrieves relevant parameters and settings from the KiCad-to-Ngspice conversion dialog (e.g., source values, model files, simulation type).
2. **Rule Checking**: A predefined set of rules then checks the extracted parameters, such as:
   - Required library paths.
   - STOP,STEP,START time values for transient analysis.
   - Proper source name for dc analysis.
3. **Validation Results**:
   - If **invalid**, the tool highlights any issues and suggests specific corrections (e.g., "Add .lib path for transistor models").
   - If **valid**, it confirms that all parameters fall within expected ranges.

Interface Elements

As shown in Figure 4.5, the Debugging Tool Window provides:

- **Validation Messages**: Shows a concise report, either "All inputs valid" or error messages with corrective tips.
- **Quick-Fix Suggestions**: Suggests potential solutions, such as "Include valid library path for transistor33" or "Increase stop time so that it's greater than start time."

4.2.3 Implementation Highlights

1. **Automatic Launch**: When the user clicks the *Debug* button after an Ngspice simulation error, the Debugging Tool automatically parses logs and runs LSTM classification.

2. **Hybrid Approach**: By combining machine learning (LSTM, Random Forest) with explicit rule-based logic, the tool covers both nuanced error patterns (e.g., convergence issues) and straightforward input constraints.

3. **Real-time Updates**: If the user modifies parameters in the KiCad-to-Ngspice dialog, the Debugging Tool can re-validate instantly, providing a rapid feedback loop.

4. **Seamless Integration**: The Debugging Tool Window resides within the eSim environment, so users can switch between design, simulation, and debugging without leaving the application.

4.2.4 Benefits of the Debugging Tool

- **Accelerated Error Resolution**: Users get immediate, data-driven feedback on simulation issues without manual log inspection.
- **Reduced Configuration Mistakes**: Rule-based checks help enforce best practices in netlist formatting and library references.
- **Enhanced Workflow Efficiency**: Automatic classification saves time otherwise spent scouring error messages or searching for known solutions.
- **Easier Onboarding**: Novice users benefit from explicit suggestions and direct guidance on parameter constraints.

Chapter 5

Conclusion & Future Enhancements

5.1 Summary of Findings

The AI-based suggestion and debugging tool for eSim simulations offers a powerful solution for automating the error analysis and troubleshooting process. By parsing error logs generated from simulations, the tool identifies common issues and provides corrective actions, significantly reducing the time and effort required for manual debugging. It integrates machine learning models to enhance error identification and incorporates an AI chatbot for interactive, real-time support. This approach not only helps users resolve problems more efficiently but also makes debugging more accessible, especially for those with limited experience.

- **Error Log Analysis:** Automates error log analysis and identifies common issues.
- **Debugging Assistance:** Provides actionable corrective suggestions to resolve errors.
- **Learning-Based Debugging Suggestions:** Utilizes machine learning model for error classification and pattern recognition.
- **Chatbot Integration:** Integrates an AI chatbot for interactive, real-time troubleshooting.
- **Time Efficiency and User Experience Improvement:** Improves productivity by streamlining the debugging process.

The system successfully streamlines the debugging process, providing users with accurate error identification and actionable solutions, making troubleshooting faster and more accessible.

5.2 Limitations & Challenges

Despite the successful implementation, several challenges and limitations were encountered during development:

- **Error Pattern Complexity:** Some errors may be too complex or rare for the current machine learning models to identify accurately, limiting the tool's effectiveness in certain scenarios.

- **Data Quality and Training:** The tool's performance relies on the quality of the dataset used for training. Incomplete or biased data could result in inaccurate suggestions.
- **User Interaction and Feedback:** Ensuring that the AI chatbot provides helpful and relevant suggestions can be challenging, especially if the user provides unclear or ambiguous input.

5.3 Possible Improvements & Future Work

To enhance the functionality and performance of the eSim Debugging Tool, the following improvements and future work are proposed:

5.3.1 Expand Error Types in Training Data

The model may only recognize a limited set of error types based on the training data available. To enhance the model's ability to handle a broader range of issues, additional error types should be incorporated into the training dataset. These could include more intricate or less common errors that occur in real-world simulations, such as those caused by advanced circuit components, compatibility issues with other simulation tools, or unique configuration problems.

- Regularly analyze user-reported errors, identify patterns, and add them to the dataset.
- Collaborate with simulation experts to ensure a diverse range of scenarios is considered for training.

5.3.2 User Feedback Integration

Implement a mechanism to collect user feedback on the accuracy and usefulness of suggestions. This could be done through ratings (e.g., thumbs up/down), comments, or user annotations about the context of the error.

- Users could be prompted to rate the quality of suggestions after applying them. Additionally, allow users to submit custom error cases or scenarios that the model may not have addressed.
- Feedback would then be processed to refine the model's future predictions and recommendations.

5.3.3 Relearning and Continuous Model Improvement

The model may become static and less effective over time if it is not updated with new data and trends.

- Implement a dynamic learning pipeline where the tool can be periodically retrained using new logs and feedback. This could be done on a scheduled basis (e.g., monthly or quarterly) to ensure the model stays up-to-date.
- An automated system for collecting new logs and re-training the model would streamline this process.

5.4 Conclusion

This project streamlines the debugging process for circuit designers, significantly reducing manual effort and saving valuable time. By providing precise error detection and actionable recommendations, it enhances the reliability and accuracy of circuit simulations. Furthermore, the project contributes to the FOSSEE community by offering an open-source tool that supports both educational and professional applications, fostering accessibility and innovation in circuit design.

Bibliography

[1] FOSSEE eSim Project. *eSim Resources*. Available at: `https://esim.fossee.in/resources`

[2] Python Software Foundation. *The Python Programming Language*. Available at: `https://www.python.org/`

[3] Ollama *Local AI Model Runner*. Available at: `https://ollama.com/search`

[4] Qwen2.5-coder. *GenAI model for code generation, code reasoning and code fixing*. Available at: `https://ollama.com/library/qwen2.5-coder`

[5] Tensorflow *Open-source machine learning framework*. Available at: `https://www.tensorflow.org/tutorials`

[6] open source spice simulator for electric and electronic circuits. *Ngspice Documentation*. Available at: `https://ngspice.sourceforge.io/docs/ngspice-manual.pdf`