eSim Semester Long Internship Autumn 2025

Project Report

On

AI Chatbot Integration in eSim

Submitted by:

Nicholas Coutinho

Department of Artificial Intelligence and Data Science

KJ Somaiya School of Engineering

Under the guidance of:

Prof. Prabhu Ramachandran

Principal Investigator

Department of Aerospace Engineering

Indian Institute of Technology Bombay

January 29, 2026

Acknowledgement

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.

I am deeply grateful to my mentor, **Mr. Sumanto Kar**, for his consistent guidance, technical expertise, and mentorship throughout the course of this project. His insights, constructive feedback, and systematic approach were crucial in addressing challenges and ensuring the successful development and integration of the AI chatbot in eSim.

I extend my sincere appreciation to my internal mentors, **Mr. Varad Patil**, **Mr. Aditya Minocha**, 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.

This internship provided valuable exposure to software development, artificial intelligence, and electronic design automation within a structured open-source environment. I also thank the FOSSEE team for their support and coordination, which facilitated smooth progress and the successful completion of the project.

Contents

Chapter 1

Introduction

1.1 Background of the FOSSEE–eSim Environment

The Free/Libre and Open Source Software for Education (FOSSEE) project is based at the Indian Institute of Technology Bombay (IITB) and is part of the National Mission on Education through Information and Communication Technology (NMEICT), Ministry of Education (MoE), Government of India. The initiative is aimed at promoting the adoption of open-source software within engineering education, research, and technological development. By encouraging the use of freely accessible and community-supported tools, FOSSEE seeks to reduce reliance on proprietary platforms and to provide sustainable alternatives that support academic and institutional requirements. As part of this effort, a range of domain-specific software solutions have been developed to facilitate learning, experimentation, and innovation across multiple science and engineering disciplines.

One of the major outcomes of this initiative is eSim, an Electronic Design Automation (EDA) platform developed to support circuit design, simulation, and analysis. eSim provides an integrated environment that combines schematic capture with simulation backends and waveform-based analysis tools. By leveraging multiple open-source components within a unified workflow, the platform enables users to design and evaluate analog, digital, and mixed-signal circuits. Furthermore, eSim maintains compatibility with widely accepted simulation methodologies, allowing it to function as a practical alternative to commercial circuit design systems.

eSim has been widely adopted in academic institutions for laboratory instruction, independent learning, and research-oriented circuit development. Its open-source architecture offers transparency, extensibility, and flexibility in terms of tool customization and accessibility. However, this openness also implies that users are often required to engage more directly with circuit modelling principles, simulator constraints, netlist-level conventions, and tool-specific operational guidelines. As a result, while eSim provides a powerful educational and design platform, it can present a steep learning curve for new and intermediate users, particularly during debugging and simulation error resolution. A representative view of the eSim workspace is presented in Figure 1.1.

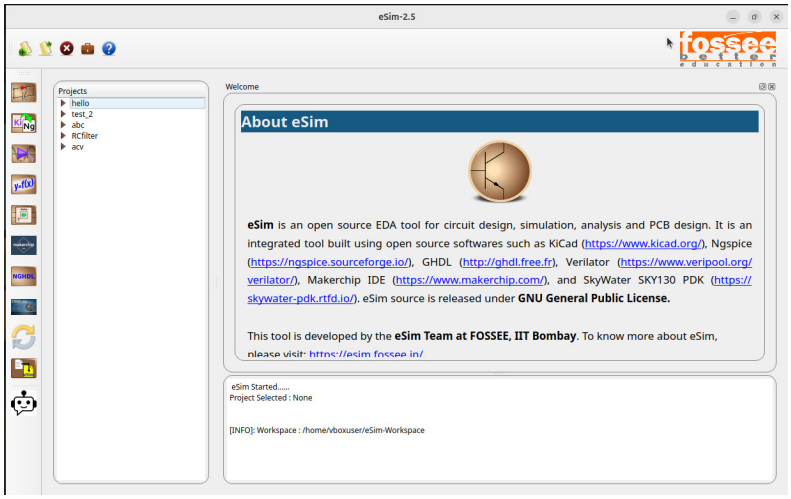

Figure 1.1: Graphical interface of the eSim platform

1.2 Rationale for Intelligent Assistance in Circuit Simulation

Circuit simulation tools operate on formally defined mathematical representations, including systems of differential and algebraic equations derived from circuit topologies and device models. Although such frameworks enable accurate prediction of circuit behavior, they also generate diagnostic warnings and error outputs that are often low-level, domain-specific, and syntactically concise. Users are therefore expected to interpret these messages and relate them back to schematic-level design constraints, which can be difficult without strong familiarity with simulator semantics.

The complexity of circuit simulation workflows introduces a significant usability barrier, particularly during iterative design and debugging phases. Minor inconsistencies such as incorrect net connectivity, missing model declarations, invalid parameter ranges, or incomplete component definitions can prevent convergence or result in simulation failure.

In many cases, identifying the root cause requires manual inspection of netlists, rule constraints, and simulator feedback, leading to increased overhead and reduced efficiency. These challenges highlight the need for structured assistance that can reduce diagnostic effort and support more systematic troubleshooting practices.

Embedding intelligent support mechanisms directly within the simulation environment can address these limitations by providing contextual interpretation of errors and design-level guidance. By correlating user queries with schematic context, simulation logs, and known failure patterns, such assistance can facilitate more effective resolution strategies. This improves accessibility for beginners while also enabling more efficient interaction with complex circuit design tasks, thereby strengthening both usability and the overall learning experience.

1.3 Aim and Scope of the Project

The primary aim of this project was to develop an AI chatbot to assist users in understanding and resolving issues encountered during circuit design and simulation in eSim. The chatbot is intended to serve as an interactive support mechanism that provides guidance when users experience difficulties in interpreting simulation results or responding to system-generated errors. By offering contextual explanations and recommendations, it complements existing documentation and enhances the overall troubleshooting process.

The scope of the project encompasses several essential capabilities required for effective assistance within the circuit simulation environment. These include natural language query processing, enabling users to communicate problems and requests in a conversational manner, as well as schematic and netlist analysis to extract meaningful circuit-level information. The system focuses on identifying common simulation errors, such as connectivity faults, missing elements, or configuration inconsistencies, and subsequently generates corrective suggestions to support users in resolving such issues. Furthermore, the chatbot is designed to accommodate both textual and visual inputs, thereby allowing flexible modes of interaction.

A key design objective of the proposed system is offline operability. All processing, retrieval, and inference tasks are executed locally, ensuring that the chatbot remains functional even in environments with limited or no internet access. This feature is particularly beneficial in academic laboratories and standalone setups where network connectivity may be constrained. Additionally, the chatbot functions as an advisory layer that enhances usability and user support without introducing any modifications to the underlying eSim

simulation engine, thereby preserving the integrity of the existing software framework.

Chapter 2

Functional Capabilities of the eSim Copilot

2.1 Offline Query Assistance

Offline query assistance is a fundamental capability of the eSim Copilot. The system is designed to process user queries locally through preloaded language models and a structured knowledge base derived from relevant documentation, simulation guidelines, and domain-specific troubleshooting resources. This architecture enables the chatbot to provide meaningful support without relying on external services or continuous network connectivity.

The offline design ensures consistent access to assistance regardless of network availability. In academic laboratories, institutional environments, or controlled setups where internet access may be restricted, offline operation allows uninterrupted usage during experiments, coursework, and self-learning sessions. This strengthens the reliability and accessibility of the system, particularly in educational contexts where continuous technical guidance is valuable.

Local processing also improves system responsiveness, predictability, and user control. Since all computations are performed directly on the user's machine, the chatbot avoids latency introduced by remote servers and ensures that circuit-related information and user interactions remain within the local execution environment, thereby enhancing both privacy and operational stability.

2.2 Circuit Debugging Support

The eSim Copilot provides structured debugging assistance by identifying common schematic-level and simulation-related issues encountered during circuit development. It analyzes user queries to detect references to known error conditions and associates them with predefined diagnostic categories such as connectivity faults, missing model declarations, incorrect component specifications, or convergence-related simulation failures.

Based on this classification, the chatbot generates targeted recommendations that guide users toward corrective actions. These suggestions are aligned with established eSim and NgSpice practices, ensuring that the guidance remains consistent with the simulator's operational constraints and expected workflows. An example interaction demonstrating debugging assistance is shown in Figure 2.2.

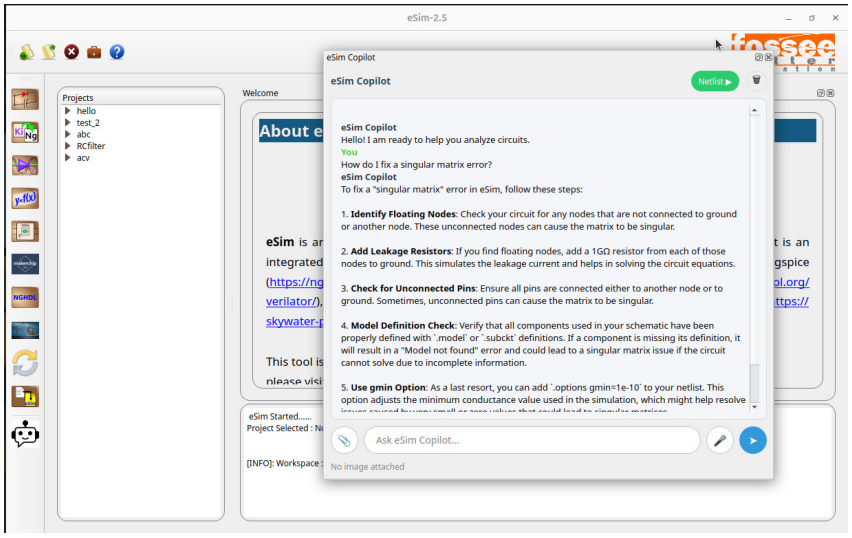

Figure 2.2: Copilot-generated guidance for resolving simulation errors

In addition to suggesting fixes, the system emphasizes conceptual clarity by explaining why an issue occurred. This approach encourages systematic debugging practices, supports deeper understanding of simulation behavior, and reduces the likelihood of repeated errors during iterative circuit design.

2.3 Schematic Interpretation

Schematic interpretation enables the chatbot to reason about circuit structure, component relationships, and node connectivity. By analyzing schematic representations, the system can infer how components interact electrically and how signals propagate through the designed circuit topology.

This capability allows the chatbot to respond effectively to queries related to circuit configuration and functional intent. It assists users in verifying whether a schematic aligns with expected behavior and highlights regions that may require closer attention or revision. Figure 2.3 illustrates the assistant's interpretation of schematic-level circuit structure.

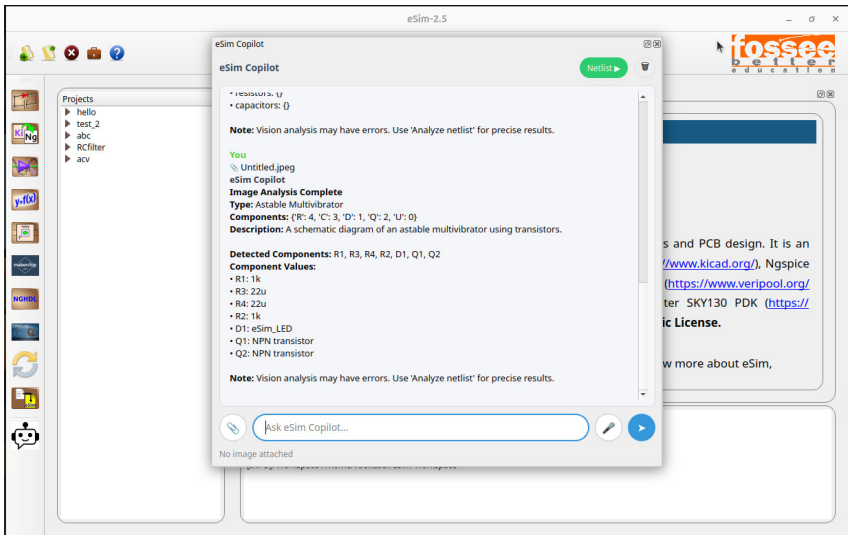

Figure 2.3: Example of schematic-level circuit interpretation by the Copilot

Furthermore, schematic interpretation supports early detection of inconsistencies at the design stage. By identifying potential issues before simulation execution, the system helps users reduce debugging overhead and improves overall design efficiency.

2.4 Netlist Evaluation Features

Netlists serve as the formal textual input to SPICE-based simulators and represent schematic designs in a structured format suitable for simulation execution. Although essential for accurate simulation, netlists are often difficult for users to interpret directly due to their compact syntax, directive-driven structure, and low-level abstraction.

The eSim Copilot evaluates netlists by examining component declarations, simulation directives, parameter values, and structural constraints. It identifies potential problems such as missing specifications, improper references, unsupported statements, or conflicting instructions that may prevent successful simulation or affect numerical stability. The integrated netlist evaluation capability is demonstrated in Figure 2.4.

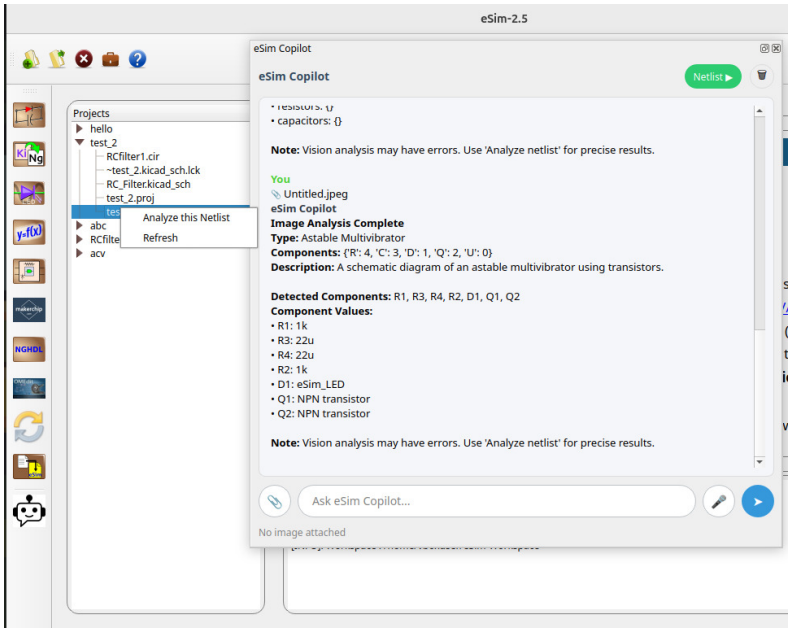

Figure 2.4: Netlist evaluation option integrated within the eSim Copilot interface

By providing structured explanations of netlist elements and their role within the simulation workflow, the system helps users better understand the correspondence between schematic design decisions and simulator-level execution behavior, thereby improving interpretability and troubleshooting effectiveness.

2.5 Multimodal Interaction Support

The eSim Copilot supports multimodal interaction by accepting both textual and visual inputs during user engagement. Users can submit schematic images alongside natural language queries, enabling the system to incorporate visual circuit context into its analysis and reasoning process.

This capability is particularly useful in scenarios where textual descriptions are insufficient to accurately convey circuit topology, component placement, or connectivity. Visual input allows the chatbot to interpret structural relationships more effectively and provide responses that are better aligned with the user's specific design configuration.

Multimodal support enhances accessibility and flexibility by accommodating different user workflows and interaction preferences. By enabling richer communication between users and the assistant, the system contributes to a more intuitive and effective troubleshooting experience within the circuit simulation environment.

Chapter 3

Problem Definition

3.1 Complexity of Circuit Design Workflows

Circuit design workflows consist of multiple interdependent stages, including schematic development, component selection and parameterization, netlist generation, and simulation execution. Each of these stages requires careful adherence to design conventions, consistency in parameter specification, and compliance with tool-specific constraints imposed by the simulation environment. Even minor inconsistencies introduced during schematic construction may propagate through later stages, affecting the correctness and stability of simulation outcomes.

As circuit designs increase in scale and functional complexity, interactions between components, subcircuits, and system-level blocks become progressively more difficult to trace. Dependencies between analog and digital elements, feedback networks, and hierarchical structures can introduce subtle design faults that may not be immediately visible during schematic creation. Consequently, such issues are often detected only when simulation is attempted, at which point identifying their origin becomes significantly more challenging due to indirect and low-level diagnostic feedback.

In addition to technical complexity, circuit design workflows frequently require users to shift between multiple representations such as graphical schematics, textual netlists, simulation logs, and waveform outputs. Managing these transitions increases cognitive effort, particularly for learners who are still developing familiarity with modeling conventions. This complexity heightens the likelihood of oversight and highlights the need for structured support throughout the design and debugging process.

3.2 Common User Errors in Simulation

Users frequently encounter simulation failures that arise from incomplete, inconsistent, or incorrect design specifications. Common examples include missing ground references, unconnected or floating nodes, incompatible component values, and undefined or unsupported device models. Such issues may prevent successful simulation execution, lead to convergence errors, or result in outputs that do not reflect intended circuit behavior.

Although simulation engines provide diagnostic warnings and error messages when failures occur, these outputs often emphasize internal failure conditions rather than practical corrective guidance. Effective interpretation of such feedback requires familiarity with SPICE syntax, circuit modeling assumptions, and numerical solver constraints, which may not be readily accessible to new or intermediate users operating in educational or laboratory contexts.

Understanding these recurring error patterns is therefore essential for designing effective assistance mechanisms. By recognizing common sources of simulation failure and their underlying causes, support systems can more directly relate simulator feedback to meaningful corrective actions, thereby improving troubleshooting efficiency and user comprehension.

3.3 Gaps in Existing Tool Assistance

Although eSim and related simulation tools provide reference documentation, user manuals, and basic diagnostic outputs, these resources are largely static and externally oriented. Users are typically expected to consult documentation independently, interpret error logs manually, and determine corrective steps through personal reasoning and experience. This creates a disconnect between simulator feedback and accessible resolution strategies during practical circuit development.

The absence of interactive and context-aware assistance significantly limits the effectiveness of existing support mechanisms. When simulation errors arise, users must independently identify relevant information, interpret technical messages, and apply fixes through repeated trial and error. This process can be time-consuming, particularly for beginners, and often discourages systematic experimentation and deeper learning.

This gap highlights the need for intelligent support that is integrated directly within the

circuit design and simulation workflow. Context-aware assistance can reduce cognitive overhead by improving interpretability of errors, guiding users toward resolution strategies informed by design context, and supporting more efficient troubleshooting practices. Such integration strengthens usability and contributes to improved learning outcomes within open-source circuit simulation environments.

Chapter 4

Overall System Architecture

4.1 Architectural Overview

The overall system architecture of the eSim Copilot is designed around a centralized control mechanism responsible for managing user interaction and coordinating processing across specialized analytical modules. This controller serves as the main entry point of the system, ensuring that user inputs are handled in a structured and consistent manner. User queries may be submitted through the interface in multiple formats, including natural language text or schematic images, thereby enabling flexible interaction modes suited to different troubleshooting scenarios. Once input is received, the controller performs initial preprocessing operations such as query normalization, intent detection, and context extraction, before determining which analytical components should be activated.

The architecture integrates several complementary capabilities within a unified assistance framework, including natural language understanding, documentation-based knowledge retrieval, rule-driven fault diagnosis, and circuit-aware interpretation of schematic or netlist context. This combination allows the chatbot to provide support across a wide spectrum of circuit design and simulation tasks. For instance, the system can respond to conceptual questions regarding circuit behavior, assist in resolving simulation failures through structured diagnostics, and help verify schematic configurations through design-level reasoning. The modular organization ensures that each analytical capability operates within clearly defined functional boundaries, while still contributing collaboratively toward generating coherent, context-aware responses for the user.

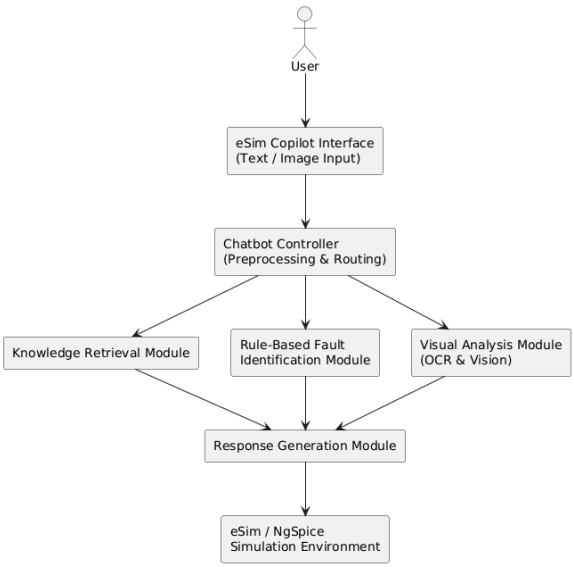

Figure 4.1: Modular processing pipeline of the eSim Copilot architecture

As illustrated in Figure 4.1, query processing follows a structured pipeline in which input is routed through specialized modules, and the outputs are consolidated into a final advisory response. This organization ensures systematic handling of diverse user requests while maintaining clarity in component responsibilities.

4.2 Component-Level Organization

The eSim Copilot is organized into distinct functional components, each responsible for a specific role within the assistance pipeline. These include modules for knowledge retrieval, rule-based fault identification, visual schematic interpretation, and response formulation. Each component encapsulates its internal processing logic, allowing the system to maintain separation of concerns and reduce unnecessary dependencies across modules.

This modular organization supports maintainability and extensibility by isolating complexity within well-defined units. Individual modules can be improved or refined independently as enhanced models, additional diagnostic rules, or expanded circuit reasoning capabilities become available. Such a design also simplifies testing and evaluation, since intermediate outputs at each stage can be examined systematically.

By structuring the chatbot into independent analytical components, the system achieves greater transparency, reliability, and flexibility in addressing circuit simulation challenges.

4.3 Inter-Module Communication Flow

Inter-module communication is facilitated through structured data exchanges that preserve contextual information throughout the processing pipeline. Intermediate representations, such as extracted query intent, identified diagnostic categories, retrieved documentation segments, or schematic-level observations, are passed between modules in a controlled and standardized format.

This communication strategy reduces redundant computation by allowing results produced by one component to be reused by subsequent stages. For example, once a query is categorized as an error condition, downstream modules can focus specifically on generating targeted explanations and corrective guidance. Such coordination enables multiple analytical perspectives to contribute meaningfully toward a unified response.

Visual inputs are incorporated into this communication flow through a schematic selection interface, which serves as the entry point for image-based analysis within the pipeline. The schematic image selection step supporting visual input is illustrated in Figure 4.3.

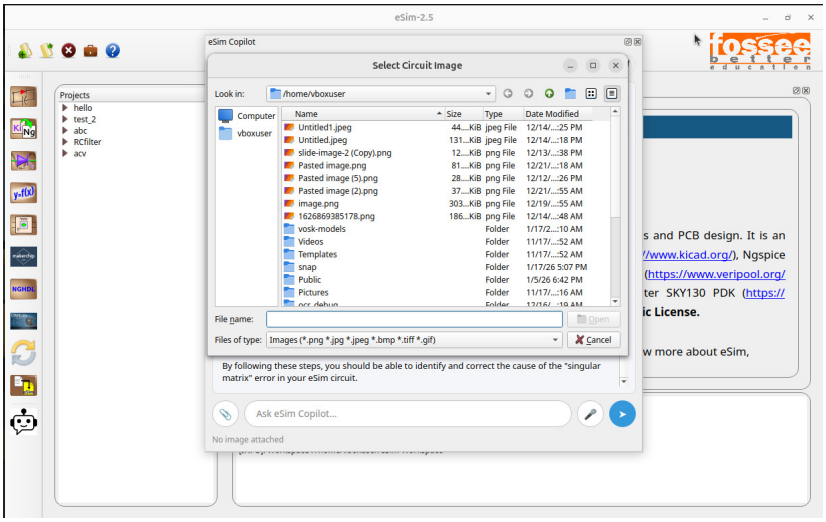

Figure 4.3: Schematic image selection step enabling visual input for circuit analysis

A well-defined exchange mechanism is essential for maintaining coherence, particularly in troubleshooting scenarios that require combined linguistic understanding, circuit context, and simulator-specific reasoning.

4.4 Interaction with eSim and NgSpice

The eSim Copilot interfaces with the eSim environment by interpreting schematic information, netlists, and simulation outputs generated through the NgSpice backend. These

artifacts serve as analytical inputs that allow the chatbot to reason about circuit structure, simulation constraints, and common execution failures encountered during design evaluation.

Importantly, the chatbot operates exclusively as an advisory layer and does not modify circuit files, alter simulation execution, or interfere with the internal behavior of eSim or NgSpice. By maintaining this non-intrusive integration approach, the architecture preserves compatibility with established workflows while enhancing usability through interpretive assistance.

This design ensures that users benefit from contextual guidance and structured support while retaining full control over circuit development and simulation processes within the standard toolchain.

Chapter 5

Chatbot Control Flow and Processing Pipeline

5.1 User Input Preprocessing

User input preprocessing forms the initial stage of the chatbot control flow, where raw user input is normalized into a consistent and analyzable format. Since user queries may vary widely in structure, terminology, and completeness, this stage performs essential preparation to reduce linguistic variability and ensure robust downstream processing. Typical operations include the removal of formatting artifacts, correction of incomplete or malformed expressions, and standardization of textual representation for reliable interpretation. This step is particularly important in conversational environments, where users may provide fragmented descriptions, informal phrasing, or partially specified circuit-related requests.

In addition, preprocessing incorporates validation steps to confirm that the input conforms to expected interaction patterns. This includes filtering irrelevant tokens, handling typographical inconsistencies, and ensuring that the query can be mapped to meaningful circuit or simulation context. Such normalization improves the accuracy of subsequent categorization and analytical stages, allowing the system to interpret user intent more effectively. Moreover, preprocessing establishes a structured input representation that supports consistent handling of both general informational questions and technical debugging requests, thereby improving overall system reliability and response quality. The end-to-end control flow for query processing is summarized in Figure 5.1.

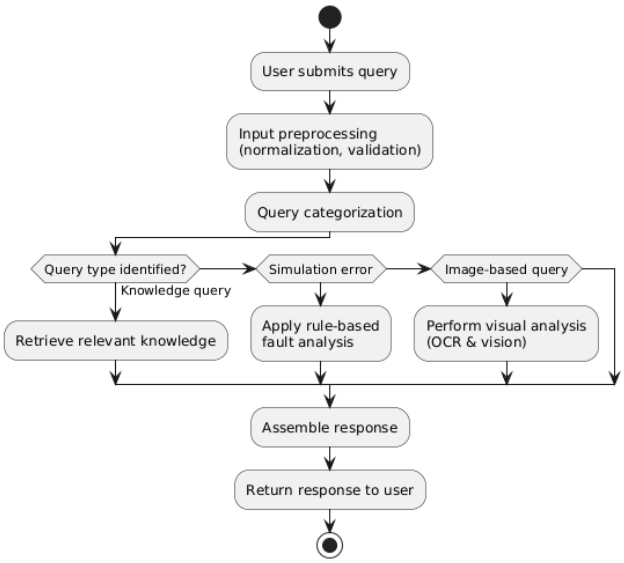

Figure 5.1: Logical flow of query handling in the eSim Copilot

Furthermore, the preprocessing stage detects embedded constructs such as SPICE netlists, simulator error outputs, or schematic image references that require specialized handling in later stages. Early identification of these elements ensures that downstream modules receive input that is both contextually meaningful and appropriately structured for circuit-aware analysis.

5.2 Query Categorization Mechanism

The query categorization mechanism determines the intent and analytical nature of the user's request by examining linguistic cues, key terms, and contextual indicators. Rather than treating all user inputs uniformly, the system assigns each query to a predefined category that reflects its processing requirements. This stage functions as a decision layer that guides the chatbot toward the most suitable reasoning pathway.

Categorization enables the system to distinguish between informational queries, simulation-related issues, schematic analysis requests, netlist evaluation tasks, and follow-up interactions. This structured classification reduces interpretive ambiguity and ensures that responses are generated in a context-aware manner rather than through generic processing.

By mapping each query to an intent category, the system improves both efficiency and response relevance. Categorization also supports scalable development, as new categories and analytical handlers can be incorporated over time without altering the overall pipeline structure.

5.3 Request Routing Strategy

Following categorization, the request routing strategy directs the query to the most suitable processing pathway within the system. Each query category is associated with a specialized handler designed to address its specific analytical needs, such as knowledge retrieval, rule-based debugging, schematic interpretation, or netlist-level evaluation.

Routing enforces a clear separation of concerns by ensuring that only the relevant modules are activated for a given request. This avoids unnecessary computation and allows the system to maintain predictable behavior across diverse interaction scenarios. For instance, a conceptual query may primarily engage the retrieval module, whereas a simulation failure may trigger diagnostic reasoning based on known error patterns.

This modular routing strategy enhances maintainability and extensibility by allowing individual processing pathways to evolve independently. New handlers can be introduced as the system expands, without disrupting existing workflows or reducing architectural clarity.

5.4 Response Assembly Process

The response assembly process integrates outputs from multiple processing modules into a coherent and user-facing response. Depending on the query type, this may involve combining retrieved documentation segments, diagnostic insights, schematic-level observations, or netlist interpretations into a unified explanation.

During response assembly, the system refines the generated output to ensure clarity, logical structure, and alignment with user intent. Technical content is presented in an interpretable form, balancing completeness with conciseness so that responses remain actionable rather than overly verbose. The system also prioritizes the most relevant corrective or explanatory elements to ensure that guidance is directly applicable to the user's circuit design context.

By consolidating module-level results into structured support, this stage ensures that final responses are coherent, informative, and practically useful. As a result, the chatbot provides effective assistance within the eSim simulation workflow while maintaining consistency across diverse troubleshooting and learning scenarios.

Chapter 6

Knowledge Retrieval Framework

6.1 Structure of the Knowledge Repository

The knowledge repository serves as a structured collection of reference material relevant to circuit simulation, NgSpice diagnostics, and eSim usage. It functions as the primary informational foundation from which the chatbot retrieves grounded technical guidance during user interaction. The repository is designed to store domain knowledge in an organized and searchable form, enabling accurate retrieval of support content aligned with specific circuit design and troubleshooting scenarios.

Rather than treating documentation as monolithic text, the repository maintains logically separated content units that preserve contextual continuity while improving retrieval granularity. Each unit corresponds to a focused technical segment, such as an error interpretation, simulation directive explanation, or schematic configuration guideline. This organization allows the system to retrieve only the most relevant informational fragments, improving response specificity and minimizing the inclusion of unrelated context.

By maintaining the repository in a segmented and indexed format, the system supports efficient access to domain knowledge even under offline execution constraints. This design ensures that chatbot responses remain accurate, context-aware, and consistently aligned with established simulator conventions and verified documentation sources. The logical organization adopted for the knowledge repository is illustrated in Figure 6.1.

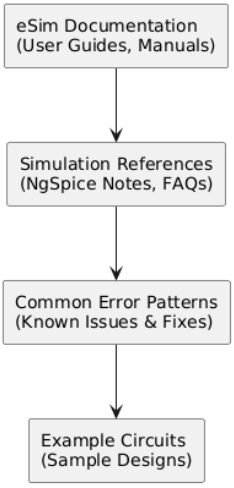

Figure 6.1: Logical organization of reference material used by the chatbot

6.2 Document Ingestion Methodology

Document ingestion defines the systematic procedure through which reference material is incorporated into the chatbot's offline knowledge repository. Since the eSim Copilot relies on curated documentation to provide grounded technical guidance, incoming sources must be processed and structured carefully to support reliable retrieval during user interaction.

The reference corpus includes eSim user manuals, NgSpice diagnostic documentation, simulation directive guides, and frequently encountered troubleshooting cases. These materials are first examined for relevance to circuit simulation workflows to ensure that stored content remains domain-specific and aligned with practical debugging requirements.

Before inclusion in the repository, documents undergo preprocessing operations such as text cleaning, removal of formatting artifacts, elimination of duplicated sections, and filtering of non-technical content. This normalization step ensures consistency across heterogeneous sources and reduces noise that could otherwise degrade retrieval precision.

Following normalization, the content is segmented into semantically meaningful knowledge units. Each unit represents a coherent technical concept, such as an error interpretation, model constraint, convergence guideline, or schematic configuration requirement. This segmentation balances contextual completeness with retrieval efficiency, enabling the assistant to return focused supporting fragments instead of entire documents.

Through controlled ingestion and structured organization, the repository becomes a re-

liable offline foundation that supports accurate, context-aware assistance across a wide range of circuit simulation troubleshooting scenarios.

6.3 Context Selection Logic

Context selection logic determines how the chatbot identifies the most relevant knowledge fragments from the repository in response to a user query. Because the repository contains a large number of segmented technical entries, selecting the most appropriate supporting context is essential for generating concise and accurate diagnostic responses.

Rather than retrieving broad document blocks, the system evaluates semantic alignment between the user's query and stored knowledge units. This enables relevance assessment beyond surface-level keyword matching, allowing the assistant to interpret conceptually similar queries even when users express problems using informal, incomplete, or varied technical language.

For example, the same fault may be described through phrases such as "simulation not running," "floating node error," or "singular matrix problem." Context selection mechanisms ensure that these expressions are mapped to the same underlying diagnostic knowledge. By prioritizing semantically aligned segments, the system maintains focus on the user's intended troubleshooting objective.

This relevance-driven strategy prevents information overload and reduces distraction from unrelated documentation. Retrieving only the most suitable fragments improves response clarity, enhances interpretability, and ensures that contextual support directly contributes to query resolution.

Overall, context selection functions as the primary link between user intent and grounded simulator guidance, thereby strengthening both usability and response reliability within an offline support environment.

6.4 Controlled Knowledge Usage Strategy

To maintain technical reliability, the eSim Copilot enforces controlled usage of retrieved knowledge during response formulation. Retrieved repository entries are treated as grounding references that constrain the assistant's explanations, rather than serving as unrestricted sources for open-ended content generation.

This grounding strategy ensures that chatbot outputs remain consistent with verified documentation, established NgSpice solver constraints, and recognized circuit simulation troubleshooting practices. In simulation environments, where incorrect guidance may lead to repeated execution failures or misinterpretation of results, such anchoring is essential for dependable support.

During response construction, the assistant synthesizes explanations by combining retrieved technical fragments with structured reasoning, while remaining bounded by the scope and validity of stored material. This reduces the risk of speculative or unsupported recommendations and improves consistency across repeated interactions involving similar fault conditions.

Controlled knowledge integration also enhances user trust, since responses remain reference-backed, technically defensible, and aligned with documented simulator conventions. By regulating how retrieved knowledge influences final output, the system strengthens interpretability, prevents hallucinated corrective suggestions, and supports reliable offline troubleshooting across diverse circuit design scenarios.

Chapter 7

Semantic Representation and Matching

7.1 Generation of Vector Representations

Vector representations are generated to encode the semantic meaning of both user queries and stored knowledge entries. These representations transform textual information into numerical embeddings that capture contextual meaning in a form suitable for mathematical comparison. By converting language into dense vector spaces, the system is able to operate on semantic relationships rather than relying solely on surface-level lexical overlap.

Unlike keyword-based approaches, embedding-based representations preserve conceptual similarity even when different wording or phrasing is used. For example, user queries expressing the same troubleshooting intent may employ varied terminology, yet remain semantically equivalent. Vector representations therefore allow the chatbot to interpret user intent more flexibly, supporting robust understanding across diverse natural language inputs.

This capability is particularly important in circuit simulation assistance, where users may describe technical issues informally or incompletely. Embedding-based encoding provides a stable mechanism for mapping such inputs into meaningful representations, enabling downstream retrieval and matching modules to function effectively.

7.2 Similarity Computation Technique

Similarity computation provides a quantitative basis for determining relevance between user intent and available contextual information stored within the knowledge repository.

Rather than relying on explicit keyword matching, the system evaluates semantic alignment between the query embedding and candidate knowledge embeddings. This enables more reliable relevance assessment even when textual expressions differ significantly.

Semantic similarity scoring underpins the retrieval process by prioritizing knowledge segments that are conceptually aligned with the user's request. As a result, the chatbot is able to select context that directly supports query resolution, while avoiding distraction from loosely related material. This improves both interpretability and efficiency of the response generation process.

This mechanism underpins both context retrieval and conversational continuity by enabling consistent relevance assessment. By prioritizing semantically aligned information, the system maintains focus on user intent while avoiding distraction from loosely related content.

In addition to retrieval, similarity computation also supports conversational continuity. By maintaining semantic consistency across interactions, the system can correctly interpret follow-up queries that refer implicitly to previous issues, thereby strengthening coherence in multi-turn troubleshooting conversations. A conceptual view of semantic similarity computation is shown in Figure 7.2.

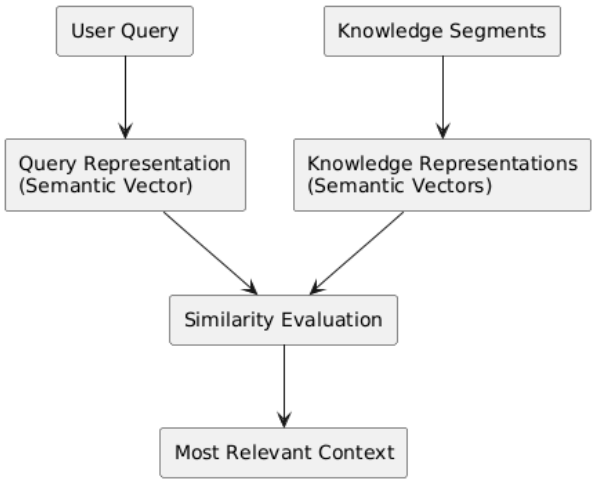

Figure 7.2: Conceptual illustration of semantic similarity matching

7.3 Detection of Topic Discontinuity

Topic discontinuity detection identifies shifts in user intent across consecutive chatbot interactions. By comparing the semantic similarity between successive query embeddings,

the system determines whether previously retained context remains applicable or whether the user has transitioned to a new topic.

When a significant topic shift is detected, prior context is disregarded to prevent incorrect assumptions from influencing the response. This is particularly important in technical support scenarios, where unrelated troubleshooting issues may arise sequentially within the same session. Topic discontinuity detection therefore ensures that responses remain aligned with the current query rather than being biased by outdated conversational context.

By dynamically managing contextual relevance, this mechanism improves response accuracy and prevents the propagation of irrelevant or misleading guidance during extended interactions.

7.4 Impact on Response Relevance

Semantic representation and matching directly influence response relevance by guiding how supporting context is selected, prioritized, and applied during response formulation. Effective semantic alignment ensures that system outputs reflect the intent of the current query rather than superficial similarity based only on shared keywords.

By decoupling relevance determination from exact phrasing, the system becomes robust to ambiguous, concise, or variably expressed inputs. This contributes to consistent response quality, improves interpretability of retrieved knowledge, and reduces the likelihood of misclassification or misinterpretation.

Overall, semantic matching strengthens the chatbot's ability to provide accurate, context-aware support within the circuit simulation workflow, ensuring that responses remain focused, meaningful, and aligned with user expectations.

Chapter 8

Rule-Based Fault Identification System

8.1 Error Pattern Recognition

Error pattern recognition focuses on identifying recurring characteristics within user queries, simulation warnings, and error outputs generated during circuit simulation. The system analyzes these inputs for recognizable indicators such as commonly observed phrasing, diagnostic keywords, simulator-generated codes, and structural irregularities that are frequently associated with known fault scenarios in eSim and NgSpice workflows.

By detecting such recurring patterns, the chatbot can move beyond surface-level interpretation and infer likely underlying issues that may not be explicitly stated by the user. For instance, errors related to missing ground references, floating nodes, or invalid component definitions often produce characteristic warning signatures that can be systematically recognized. This structured recognition process forms the foundation for consistent and explainable fault diagnosis, enabling the system to respond based on established fault trends rather than treating each error as an isolated symptom.

Pattern recognition also improves efficiency by allowing frequent simulation failures to be identified quickly, reducing the effort required for manual inspection of logs or netlists. As a result, users receive faster and more reliable diagnostic support during troubleshooting.

8.2 Fault Classification Logic

Fault classification organizes recognized issues into well-defined diagnostic categories according to their source and nature. Typical classifications include connectivity-related

faults, parameter misconfigurations, unsupported component usage, incorrect netlist directives, and violations of simulation constraints imposed by the underlying SPICE solver.

This classification logic provides a structured decision framework that guides subsequent diagnostic reasoning. By grouping similar error patterns under consistent categories, the system ensures uniform handling of repeated fault types and reduces the likelihood of inconsistent or contradictory responses across interactions. For example, connectivity faults are addressed through node-level correction guidance, whereas convergence-related faults may require parameter tuning or model adjustments.

In addition, classification enhances interpretability by providing users with clearer insight into the type of issue encountered. Instead of presenting errors as opaque simulator failures, the system frames them within meaningful diagnostic classes that support systematic learning and correction.

8.3 Severity Assessment Criteria

Severity assessment evaluates the extent to which an identified fault affects simulation execution, numerical stability, and the reliability of generated results. Faults are assessed based on their impact within the simulation pipeline, ranging from critical errors that prevent execution entirely to non-fatal issues that may allow simulation completion but compromise output correctness or produce warning-level diagnostic messages.

This assessment mechanism enables prioritization of corrective guidance by distinguishing between blocking faults and secondary concerns. For example, missing ground references, undefined device models, or invalid netlist directives typically halt simulation immediately and therefore require urgent resolution. In contrast, issues such as minor parameter inconsistencies, tolerance-related warnings, or marginal convergence instability may not fully prevent execution but can still affect waveform accuracy or result interpretability.

Severity-aware handling ensures that the chatbot directs user attention toward the most critical problems first, reducing wasted effort on less significant messages while core execution faults remain unresolved. By ordering corrective suggestions according to diagnostic urgency, the system supports a more systematic troubleshooting process and improves overall user efficiency during circuit development.

Incorporating severity criteria also enhances clarity in multi-error scenarios, where several

warnings and errors may appear simultaneously. Rather than presenting all faults equally, the assistant provides structured guidance that reflects the practical impact of each issue, thereby improving debugging effectiveness and ensuring that simulation reliability is restored in a stepwise and prioritized manner.

8.4 Corrective Action Mapping

Corrective action mapping defines structured links between identified fault categories and appropriate resolution steps. These mappings are derived from established circuit simulation practices, documented eSim and NgSpice conventions, and widely accepted troubleshooting guidelines. By grounding recommendations in recognized diagnostic standards, the system ensures that corrective suggestions remain technically reliable and consistent across recurring error scenarios.

Through targeted and actionable guidance, the assistant reduces ambiguity during fault resolution and supports more systematic debugging. Recommended actions are framed according to the specific fault type, enabling users to address simulation issues efficiently without relying on extensive manual inspection or repeated trial-and-error efforts.

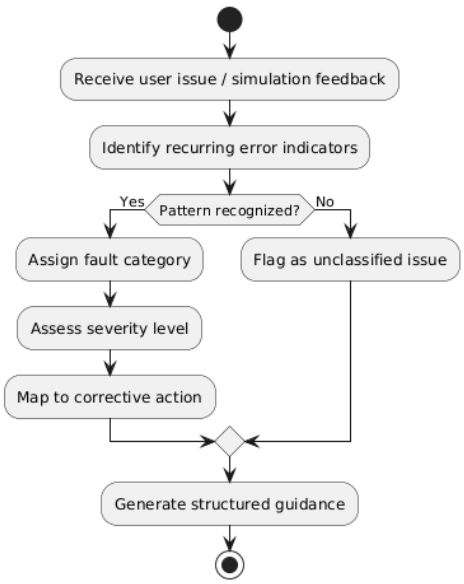

Figure 8.4: Conceptual flow of fault identification and corrective guidance generation

Chapter 9

Static Schematic and Netlist Analysis

9.1 Visual Schematic Understanding Module

The visual schematic understanding module provides the foundation for circuit-level interpretation from schematic images. It prepares circuit diagrams for subsequent analysis through a set of image-level preprocessing operations that improve consistency and robustness. Image size optimization reduces resolution and file size while preserving critical symbolic and structural features required for reliable interpretation.

This preprocessing stage enhances computational efficiency and ensures stable performance during vision-based analysis, particularly when handling large, high-density schematics containing numerous components and interconnections. Standardizing image properties such as scale, contrast, and aspect ratio minimizes variability across different schematic inputs and reduces the likelihood of downstream recognition errors.

By producing normalized visual representations, the module establishes a stable input format for later stages of text recognition, component extraction, and topology inference.

9.2 Text Recognition from Schematics

Text recognition focuses on extracting alphanumeric information embedded within schematic images, including component labels, reference identifiers, node annotations, and parameter values. This symbolic information serves as an essential complement to graphical interpretation, as schematic meaning is often distributed across both visual symbols and textual references.

Recognized text elements are normalized to resolve inconsistencies in formatting and are associated with their corresponding visual regions. This enables consistent mapping between reference identifiers (such as R1, C2, or VDD) and their associated circuit elements. Such integration supports accurate interpretation of user queries that reference specific components, values, or design constraints.

Text recognition therefore strengthens the assistant's ability to generate circuit-aware responses that reflect both schematic structure and symbolic context.

9.3 Vision-Based Circuit Interpretation

Vision-based circuit interpretation analyzes graphical features such as component symbols, connecting wires, junction points, and spatial relationships in order to infer circuit topology. Through this analysis, the system develops a high-level understanding of circuit organization independent of explicit textual descriptions.

This interpretation process enables reasoning about functional connectivity, signal propagation paths, and relationships between circuit blocks that may not be directly available from text alone. As a result, the chatbot can respond effectively to design-oriented queries that require contextual awareness of schematic layout and interconnection structure.

By transforming raw visual input into a structured topological understanding, the assistant supports deeper interaction with circuit designs beyond surface-level component identification.

9.4 Extraction of Components and Parameters

Component and parameter extraction identifies individual circuit elements within the schematic and associates them with relevant attributes such as component type, nominal values, reference identifiers, and inferred connectivity information. Extracted elements are converted into a structured internal representation that can be used for reasoning, validation, and guided troubleshooting.

This structured representation enables the system to answer design-specific queries with greater precision. For example, the chatbot can explain the role of a particular transistor stage, verify whether component values match expected operating conditions, or highlight

missing specifications. Importantly, this analysis can be performed without requiring full simulation execution, allowing early-stage assistance during schematic development.

By relying on component-level extraction, the chatbot improves its capacity to provide targeted diagnostic feedback and circuit explanations.

9.5 Visual Error Indication

Visual error indication highlights regions within the schematic that exhibit potential design inconsistencies or irregularities identified during image-based analysis. These indicators may correspond to missing connections, floating nodes, ambiguous wiring, incorrect symbol usage, or incomplete circuit structure.

By directing user attention toward specific areas of interest, this mechanism accelerates fault localization within complex circuit schematics. Visual cues reduce the cognitive effort required to manually trace large designs, thereby enabling faster inspection, correction, and refinement.

Such schematic-level feedback enhances usability, particularly for learners and intermediate users who may struggle to detect subtle connectivity or configuration issues. Visual error indication therefore complements textual diagnostic explanations by providing intuitive and spatially grounded troubleshooting support.

Chapter 10

Multimodal Interaction Management

10.1 Text Interaction Handling

Text interaction handling manages conversational input submitted in natural language and serves as the primary communication interface between the user and the eSim Copilot. Incoming text queries are interpreted to identify user intent, extract circuit-relevant terminology, and preserve a structured internal representation suitable for subsequent analytical processing. This stage ensures that free-form user language can be translated into a form that supports reliable reasoning and response generation.

This mechanism enables natural and intuitive interaction without compromising analytical rigor. By separating linguistic interpretation from downstream diagnostic or retrieval processes, the system ensures that conversational flexibility does not interfere with the accuracy and consistency of technical assistance. Text handling therefore provides the foundation for supporting both conceptual questions and simulation-oriented troubleshooting requests within a unified dialogue framework.

In addition, text interaction management supports multi-turn communication by retaining query context when appropriate. This ensures that follow-up questions can be interpreted correctly, thereby improving conversational coherence and usability during extended troubleshooting sessions.

10.2 Image Interaction Handling

Image interaction handling coordinates the processing of visual inputs associated with user queries, such as schematic diagrams, circuit snapshots, or annotated design representations. Visual analysis is performed in conjunction with the surrounding textual context to ensure that schematic interpretation remains aligned with the user's intent and the

ongoing interaction state.

This capability extends interaction beyond text-only interfaces and allows users to reference circuit structure directly through visual input. Such support is particularly valuable when connectivity, topology, or spatial design relationships cannot be expressed easily through natural language alone. By integrating visual processing into the interaction pipeline, the system enables richer and more expressive query formulation without increasing user complexity.

Furthermore, image interaction handling ensures that visual inputs are normalized and interpreted in a structured manner before being incorporated into circuit reasoning modules. This supports consistent extraction of components, parameters, and connectivity features required for accurate multimodal assistance.

10.3 Context Linking Across Modalities

Context linking across modalities unifies information derived from both textual and visual inputs into a coherent interaction state. The system correlates extracted visual features, such as detected components or circuit regions, with relevant textual references provided by the user. This alignment ensures that the assistant maintains a consistent understanding of the circuit design under discussion across different input forms.

This unified context management enables more accurate and context-aware responses by preventing fragmented interpretation between modalities. For example, a user may reference a particular transistor label in text while also providing a schematic image containing that element. By reconciling both sources of information within a single framework, the system improves grounding and response relevance.

Effective modality linking also supports seamless transitions between different interaction modes, allowing users to alternate between textual descriptions and schematic-based clarification without disrupting the continuity of troubleshooting or circuit explanation.

10.4 Resolution of Image-Based Follow-Ups

Resolution of image-based follow-up queries addresses subsequent user interactions that refer to previously analyzed visual inputs. Rather than requiring complete reprocessing of the original schematic for each follow-up question, the system selectively reuses relevant

extracted representations such as component lists, inferred connectivity, or structural observations.

This approach supports conversational continuity while minimizing redundant computation. By leveraging retained visual context, the chatbot can respond efficiently to incremental queries, such as requests for clarification about a specific circuit block or component role. At the same time, appropriate contextual boundaries are maintained to avoid incorrect carryover when the user shifts to a different topic or schematic.

Overall, follow-up resolution ensures that multimodal interactions remain consistent, computationally efficient, and aligned with prior analysis, thereby enhancing the usability of the assistant during extended design and debugging sessions.

Chapter 11

Netlist Interpretation and Refinement

11.1 Netlist Input Structuring

Netlist input structuring serves as the foundational step for systematic netlist analysis and refinement. At this stage, user-provided netlists are examined to verify conformance with expected formatting conventions and syntactic structure. Core elements such as component declarations, node references, model specifications, and simulation directives are identified and separated to enable focused examination of each structural component.

This structuring process allows the system to distinguish descriptive circuit content from executable simulator instructions. By organizing the netlist into logical segments, such as element definitions, control statements, and parameter assignments, the assistant reduces interpretive ambiguity and prepares the input for reliable analysis. This is particularly important when netlists are incomplete, reordered, densely formatted, or generated through intermediate schematic conversion.

Through systematic structuring, the chatbot establishes a stable internal representation of the netlist that supports subsequent interpretation, validation, and refinement guidance.

11.2 Prompt Conditioning for Netlists

Prompt conditioning involves embedding the structured netlist information within a constrained analytical prompt suitable for controlled processing. Rather than forwarding raw netlist content directly into unrestricted generation, the system frames the information to emphasize explicitly observable structure, declared parameters, and simulator-relevant constraints.

This conditioning approach ensures that interpretation remains grounded in the provided netlist content and prevents unsupported assumptions during analysis. By enforcing clear boundaries on what may be inferred, the chatbot improves consistency, technical reliability, and safety when generating netlist-related diagnostic responses.

Prompt conditioning also enables the assistant to focus on actionable issues such as missing parameters, invalid node connections, undefined models, or conflicting directives, thereby increasing the practical usefulness of refinement feedback.

11.3 Output Sanitization

Output sanitization focuses on refining analytical responses prior to presentation to the user. Generated outputs are examined to remove extraneous tokens, intermediate reasoning artifacts, or formatting elements that do not contribute directly to user understanding or corrective action.

Sanitization ensures that responses remain concise, readable, and aligned with the intended diagnostic objective. This step is essential in technical assistance systems, where overly verbose or unstructured explanations can overwhelm users and obscure the key corrective guidance.

By filtering out non-essential content and emphasizing simulator-relevant insights, output sanitization enhances usability and ensures that users receive focused recommendations rather than internal processing detail.

11.4 Presentation of Analytical Results

The presentation stage transforms sanitized analytical output into structured, user-facing explanations. Rather than exposing raw simulator-level interpretations, results are organized logically to emphasize clarity, interpretability, and relevance. This ensures that users can understand identified issues without requiring deep expertise in SPICE syntax or solver-level diagnostics.

Analytical findings are presented in a manner that highlights structural inconsistencies, missing specifications, unsupported constructs, or refinement opportunities within the netlist. Responses are framed to guide users toward appropriate corrective steps while

remaining accessible to both novice and intermediate circuit designers.

This approach supports informed decision-making during circuit improvement by providing feedback that is precise, actionable, and aligned with established simulation practices. Figure 11.4 illustrates the structured output format used for presenting netlist analysis.

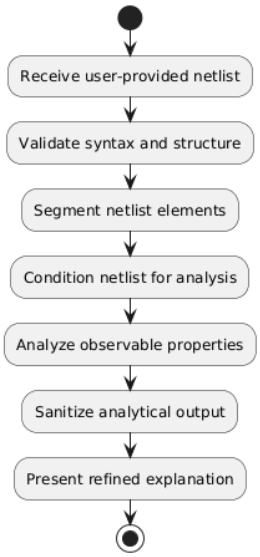

Figure 11.4: Structured presentation of netlist analysis outcomes

Chapter 12

Context Retention Mechanism

12.1 Temporary Conversation Memory

Temporary conversation memory enables the system to retain relevant interaction history during an active user session. This memory captures essential conversational elements such as recent user queries, system responses, identified fault categories, and resolved references that are required to sustain coherent dialogue across multiple turns. By maintaining this short-term interaction state, the chatbot can interpret follow-up questions more accurately and avoid treating each query as an entirely independent request.

The memory mechanism is intentionally constrained in both duration and scope. Retention is limited to a bounded window of recent exchanges, ensuring that preserved context remains relevant to the ongoing troubleshooting or explanation task. This design prevents unintended influence from outdated or unrelated interactions and reduces the likelihood of context drift over extended conversations.

By restricting memory to session-level continuity, the system maintains stability and predictability in conversational behavior while still supporting meaningful multi-turn technical assistance.

12.2 Reference Resolution in Follow-Up Queries

Reference resolution addresses follow-up queries that rely on information introduced earlier in the conversation. Users often employ linguistic constructs such as pronouns, implicit references, abbreviated expressions, or component shorthand when continuing a dialogue. The system examines these cues to determine the most appropriate contextual targets within the retained interaction state.

Accurate reference resolution enables users to engage incrementally without repeatedly restating circuit details or simulation conditions. For example, a follow-up question such as "Why did this happen?" can be correctly linked to the most recent diagnostic explanation or identified fault. This capability is particularly valuable during multi-step problem-solving workflows, where troubleshooting requires iterative refinement and clarification.

Through systematic reference tracking, the chatbot improves conversational efficiency and supports more precise interpretation of user intent across consecutive exchanges.

12.3 Benefits of Context Preservation

Context preservation enhances interaction quality by allowing related queries to be processed as part of a cohesive dialogue rather than as isolated requests. This continuity reduces redundancy in user input and supports more natural conversational progression, particularly when users are debugging complex circuits or exploring multiple aspects of a schematic design.

By retaining relevant short-term context while remaining responsive to topic discontinuity, the system maintains an effective balance between continuity and adaptability. This balance improves response relevance, reduces ambiguity in interpretation, and prevents inappropriate reuse of outdated information when user intent shifts.

Overall, context preservation contributes to a smoother and more intuitive user experience, enabling the chatbot to provide consistent guidance across extended troubleshooting sessions. The reuse of short-term conversational context is illustrated in Figure 12.3.

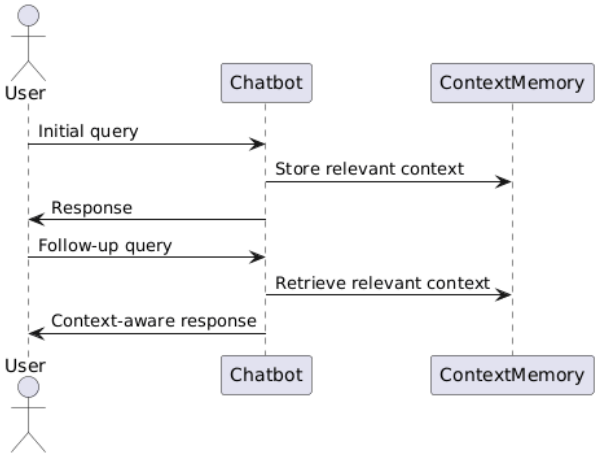

Figure 12.3: Illustration of context reuse across consecutive user interactions

Chapter 13

Implementation Stack and Dependencies

13.1 Programming Language and Runtime

The system is implemented using Python due to its expressiveness, extensive library ecosystem, and suitability for rapid development of modular intelligent systems. Python enables concise representation of complex analytical logic while maintaining readability and clarity, which is essential for iterative development, debugging, and long-term maintainability of research-oriented software projects.

The runtime environment supports modular integration of independent components responsible for language processing, vision-based schematic analysis, and rule-driven diagnostic reasoning. Python's compatibility with scientific computing libraries, local file handling utilities, and system-level operations allows the assistant to manage multi-stage analytical workflows within a unified execution framework.

This implementation choice facilitates efficient development of an offline-capable assistant by providing strong support for machine learning integration, structured knowledge management, and interaction with the eSim simulation environment.

13.2 AI and Language Processing Tools

Language processing capabilities are implemented using locally deployable language models and embedding-based semantic similarity mechanisms. These tools enable the system to perform natural language understanding, intent-sensitive context retrieval, and controlled response formulation without dependence on cloud-based services.

Semantic embeddings support robust interpretation of user queries by capturing conceptual similarity beyond surface-level keyword matching. This allows the assistant to retrieve relevant troubleshooting material and generate context-aware explanations even when users express issues using varied technical phrasing.

Local deployment ensures consistent availability in offline environments and provides predictable performance characteristics across constrained systems. This design choice aligns with the broader objective of supporting usage in academic laboratories or restricted computational settings while maintaining reliability, user privacy, and independence from external infrastructure.

13.3 Vision and OCR Libraries

Vision and optical character recognition capabilities are supported through dedicated libraries designed for image analysis and text extraction. These libraries enable the system to process schematic images, detect symbolic circuit elements, and extract textual annotations such as component labels, reference identifiers, and parameter values embedded within visual representations.

The integration of vision and OCR components allows schematic-level information to be incorporated into analytical workflows alongside textual inputs. This multimodal capability strengthens circuit understanding and supports richer interaction without relying on external image-processing services, thereby preserving full offline operability.

Such integration is particularly valuable in educational and laboratory environments, where schematic interpretation plays a central role in circuit debugging and learning.

13.4 System-Level Dependencies

System-level dependencies include supporting libraries and utilities required for numerical computation, graphical processing, and interaction with the operating system. These dependencies ensure stable execution across supported platforms and provide essential infrastructure for tasks such as embedding computation, file handling, and local resource management.

Careful selection and management of dependencies minimizes configuration complexity

for end users. By relying on widely supported and well-documented open-source libraries, the system maintains portability while reducing maintenance overhead and mitigating compatibility issues during deployment. This approach strengthens usability and ensures reliable operation within constrained academic setups.

Chapter 14

Application Scenarios

14.1 Diagnosis of Grounding and Connectivity Issues

One key application scenario involves diagnosing grounding and connectivity-related issues in circuit designs. The system assists users in identifying missing reference nodes, unintended floating connections, or improperly linked components that commonly result in simulation failures or convergence instability.

By correlating reported symptoms with known connectivity fault patterns, the chatbot provides targeted guidance that directs users toward corrective actions. This assistance reduces the need for repeated trial-and-error debugging and improves reliability during early design stages, particularly for users with limited experience in circuit simulation workflows.

Such support is especially valuable in laboratory settings, where common connectivity mistakes frequently hinder successful execution and learning progression.

14.2 Interpretation of Circuit Diagrams

The system supports interpretation of circuit diagrams by analyzing structural relationships between components, interconnections, and functional sub-blocks. This capability enables users to verify design intent and understand circuit organization without requiring extensive manual tracing of complex schematics.

Diagram interpretation is particularly useful in educational contexts and preliminary design review tasks, where users may seek clarification regarding signal flow, the functional

role of components, or expected circuit behavior before proceeding to simulation.

By providing schematic-level explanations, the assistant enhances conceptual understanding and supports more effective circuit development practices.

14.3 Automated Simulation Error Guidance

Automated guidance assists users in understanding simulation-related errors by correlating diagnostic messages with known fault categories and design inconsistencies. This guidance focuses on interpretation and explanation rather than direct modification of simulator execution.

By translating technical simulator outputs into structured and comprehensible feedback, the system reduces reliance on external documentation and improves user comprehension of simulation constraints, solver limitations, and common configuration mistakes.

This application scenario strengthens the assistant's role as an advisory support layer that improves debugging efficiency and reduces user frustration during simulation failure resolution.

14.4 Guided User Assistance Workflows

The system supports guided assistance workflows that lead users through problem resolution in a structured and incremental manner. Guidance is adapted dynamically to user inputs, enabling stepwise troubleshooting rather than isolated responses.

This approach encourages systematic reasoning by helping users understand both the origin of identified issues and the rationale behind corrective actions. Guided workflows support learning by reinforcing best practices in circuit debugging and simulation preparation.

Overall, these workflows contribute to improved usability and more effective interaction during multi-step troubleshooting sessions.

Chapter 15

System Constraints

15.1 Visual Recognition Limitations

Visual recognition capabilities are constrained by factors such as image quality, resolution, schematic complexity, and symbol clarity. Overlapping graphical elements, low-contrast annotations, ambiguous wiring, or unconventional diagram styles may reduce extraction accuracy, particularly when schematics contain dense component placement or non-standard formatting.

These limitations necessitate cautious interpretation of visual analysis results. The system therefore presents visual feedback as supportive guidance rather than definitive conclusions, encouraging users to verify highlighted areas through manual inspection. Vision-based reasoning is intended to assist troubleshooting rather than replace expert schematic review in complex design cases.

15.2 Knowledge Availability Boundaries

The system's responses are bounded by the scope and coverage of its curated knowledge repository. Queries that fall outside available documentation, stored troubleshooting patterns, or supported simulator references may not yield detailed guidance, particularly in cases involving highly specialized components or uncommon simulation scenarios.

This limitation highlights the importance of maintaining domain-relevant and carefully curated reference sources. It also reinforces the system's role as an assistive tool that enhances user understanding rather than serving as a comprehensive substitute for expert-level circuit design knowledge.

15.3 Resource Constraints in Offline Execution

Offline execution introduces constraints related to memory usage, computational capacity, and response latency. Resource-intensive operations such as semantic retrieval, vision-based schematic interpretation, and multi-stage reasoning must be managed carefully to ensure responsiveness on systems with limited hardware resources.

These constraints influence architectural and design decisions by emphasizing efficiency, modularity, and controlled processing over exhaustive computation. As a result, the system prioritizes lightweight analysis and targeted assistance, ensuring practical usability in constrained academic environments where high-performance infrastructure may not always be available.

Chapter 16

Scope for Enhancement

16.1 Advanced Circuit Reasoning

Future enhancements may include advanced reasoning capabilities that account for circuit behavior across varying operating conditions, nonlinear device characteristics, component tolerances, and design parameter variations. Such reasoning would enable deeper analytical support beyond static interpretation by incorporating functional relationships and expected electrical behavior under different simulation contexts.

Advanced circuit reasoning could further assist users in design optimization, validation, and comparative evaluation of alternative configurations. This would allow the system to provide more insightful feedback regarding stability, operating regions, biasing constraints, and performance trade-offs in complex analog and mixed-signal circuits. Incorporating behavioral awareness would also strengthen the assistant's ability to support design-level decision-making rather than only troubleshooting.

16.2 Knowledge Base Expansion

Expanding the knowledge repository to include additional documentation, troubleshooting examples, and domain-specific references would significantly improve system coverage and versatility. Broader inclusion of practical case studies, frequently encountered simulation scenarios, and extended device model references would enable support for a wider range of circuit architectures and user queries.

Careful curation remains essential to ensure accuracy, relevance, and long-term reliability. Mechanisms for maintaining up-to-date resources, filtering outdated or inconsistent material, and validating reference quality would be necessary to preserve trustworthiness as the repository continues to grow.

16.3 Interaction Quality Improvements

Enhancing interaction quality may involve refining conversational flow, improving response clarity, and adapting explanations to different user expertise levels. Improved interaction design could support more intuitive usage by offering structured guidance, clearer phrasing, and more context-sensitive assistance during troubleshooting and circuit exploration.

Such improvements would benefit both novice users seeking step-by-step instructional support and experienced users requiring concise technical clarification. Incorporating stronger follow-up handling, adaptive response depth, and lightweight personalization could further improve usability in both educational and practical engineering environments.

16.4 Performance Optimization Opportunities

Performance optimization may focus on reducing response latency, improving resource utilization, and streamlining multi-stage processing pipelines. These optimizations would support smoother operation, particularly in constrained offline environments where computational resources are limited.

Continued profiling and iterative refinement could enable scalable deployment without compromising system stability. Future work may also explore lightweight model variants, more efficient retrieval indexing strategies, and optimized vision processing pipelines to enhance responsiveness across diverse hardware platforms.

Chapter 17

Conclusion

17.1 Summary of Contributions

This project presents an intelligent assistant that integrates natural language understanding, circuit analysis, and vision-based schematic interpretation within the eSim environment. The system demonstrates how structured assistance can enhance usability while preserving the integrity and reliability of existing simulation workflows. By functioning as an advisory layer, the chatbot supports users in interpreting simulator feedback, understanding circuit representations, and resolving design-related issues more effectively.

By combining semantic retrieval, rule-based fault identification, multimodal processing, and offline execution within a unified framework, the project contributes a practical approach to improving user support in open-source EDA platforms. The modular architecture further strengthens the applicability of the system in academic and constrained computational environments.

17.2 Observed Impact on eSim Usage

The introduction of contextual and guided assistance has the potential to reduce debugging time and improve user confidence during circuit design and simulation. By translating technical simulation outputs into more interpretable explanations and corrective guidance, the system enables users to progress through troubleshooting tasks with greater clarity and reduced reliance on external resources.

The assistant supports both learning-oriented use cases, such as laboratory experimentation, and practical engineering workflows where efficient error resolution is essential. This impact underscores the value of intelligent support systems in lowering barriers to effective use of complex simulation environments and encouraging broader adoption of

open-source tools.

17.3 Closing Remarks

This work demonstrates the feasibility of integrating intelligent assistance into circuit simulation environments without altering core simulation engines or disrupting established workflows. By emphasizing usability, reliability, and offline accessibility, the project contributes toward improving the effectiveness of open-source engineering software in educational and research settings.

The approach outlined in this report highlights a practical pathway for enhancing accessibility, user experience, and learning outcomes within the open-source EDA ecosystem. Future extensions may further strengthen reasoning depth, expand knowledge coverage, and improve interaction quality, supporting continued advancement of intelligent tools for circuit design assistance.

Bibliography

[1] FOSSEE Project, IIT Bombay. Available: https://fossee.in

[2] eSim – Open Source EDA Tool for Circuit Design and Simulation. Available: https://esim.fossee.in

[3] NgSpice Simulator Official Documentation. Available: https://ngspice.sourceforge.io

[4] T. Mikolov et al., *Efficient Estimation of Word Representations in Vector Space*, 2013. Available: https://arxiv.org/abs/1301.3781

[5] P. Lewis et al., *Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks*, NeurIPS 2020. Available: https://arxiv.org/abs/2005.11401

[6] Python Software Foundation. Available: https://www.python.org