technische universität
dortmund

Final report

Project Group WS 2023-24 - AI-Based SI/PI/EMC-Compliant PCB Design Implementation of ML/AI Modules to Support Interference-Resistant Design of Microelectronic Systems

Sahil Rajpurkar, Atif Penkar, Bekhzodkhon Khamdamov, Soupal Paul

Supervisors:
Dr.-Ing. W. John
M.Sc. Emre Ecik
M.Sc. Julian Withöft
M.Sc. N. Ghafarian Shoaee
Prof. Dr.-Ing. Jürgen Götze
External Industrial EDA Supervisor:
Dipl.-Inf. Ralf Brüning

Technische Universität Dortmund
Elektrotechnik and Informationstechnik (Arbeitsgebiet Datentechnik)
Otto-Hahn-Straße 4 – 44221 Dortmund

Contents

Chapter 1

Introduction

The rapid advancement of natural language processing (NLP) algorithms and neural networks has significantly accelerated the integration of artificial intelligence (AI) in chatbot systems. In our project, these systems are leveraged to address complex queries related to signal integrity and power integrity in electronic design automation (EDA), enhancing user interaction and streamlining the problem-solving process in this specialized domain.

In recent years, chatbots have been developed to address a wide range of needs, including providing essential services in educational institutions. University chatbots, powered by AI, assist students by offering information about curricula, admissions, schedules, and grades, and even provide additional services like worship schedules and weather forecasts. These chatbots, often built using deep learning models such as Long Short-Term Memory (LSTM) integrated with the Rasa framework, exemplify how AI can replicate human intelligence through specific training schemes. Platforms like Facebook facilitate the deployment of these chatbots, leveraging their vast user base to ensure wide accessibility [8].

To mitigate new challenges and expedite design processes, intelligent chatbot systems are poised to play a pivotal role in the future of PCB design applications. Leveraging NLP techniques, such as RASA NLU (Natural Language Understanding), developers can interact with chatbots to access domain-specific knowledge and guidance tailored to their design tasks. By interpreting natural language queries and utilizing trained models, chatbots empower developers to navigate the intricacies of SI, PI, and EMC in PCB design.

Furthermore, the development of such a chatbot system addresses the increasing need for smart, AI-powered tools in educational and professional settings. For instance, smart systems for universities can greatly benefit from AI chatbots that provide instant access to information on curriculum, admissions, and schedules. This integration not only enhances user experience but also streamlines administrative processes, especially during situations that limit face-to-face interactions.

By leveraging the power of AI and NLP, the proposed chatbot system aims to bridge the gap between complex EDA tasks and user-friendly interfaces. This report delves into

the design and implementation of the chatbot, highlighting its functional framework and the principle of RASA NLU. It also presents an analysis of the system's performance and discusses its potential impact on the future of EDA software and intelligent user interfaces.

1.1 Objective

The primary objective of this project is to develop an AI-powered chatbot system tailored to the specific needs of PCB designers, particularly in the domains of SI, PI, and EMC. By leveraging advanced NLP techniques and integrating with existing EDA software environments, the chatbot aims to provide comprehensive support for developers throughout the design process.

Key objectives include:

- To define a functional framework for the chatbot system, incorporating RASA NLU and neural network methodologies.
- To implement principles of entity extraction, intent recognition, and keyword mapping to enhance chatbot functionality.
- To facilitate seamless interaction between developers and the chatbot, enabling efficient access to domain-specific knowledge and guidance.

Through these objectives, the project aims to demonstrate the feasibility and efficacy of AI-powered chatbot systems in enhancing the PCB design process.

1.2 Overview

This report presents a comprehensive exploration of AI and machine learning (ML) applications in the domain of microelectronic system design, particularly focusing on Printed Circuit Board (PCB) development. The investigation is structured into several key areas:

1.2.1 Graphical User Interface (GUI)

An examination of the GUI components essential for integrating AI/ML capabilities into the PCB design workflow. This includes the development of a ChatBot interface for facilitating natural language interaction, as well as dedicated modules for Signal Integrity (SI), Power Integrity (PI), and Transmission Line Analysis.

1.2.2 Intent-Based Model

A detailed exploration of the intent-based model, outlining its methodology, underlying model architecture, evaluation metrics, and testing procedures. This section aims to elucidate how AI/ML techniques can be leveraged to enhance PCB design processes by understanding developer intents and providing relevant guidance. The Intent-Based Model can

be developed for multiple topics and sub-domains, in the given project we have divided the topics into the following domains, as shown in Figure 1.1

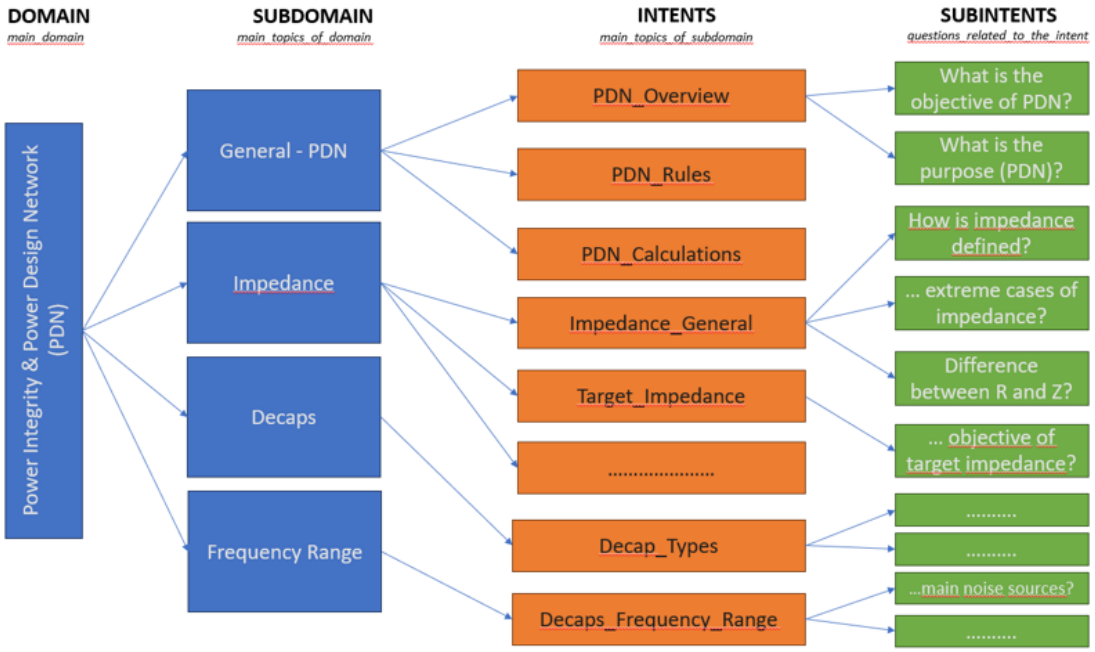

Figure 1.1: Sub Domains

1.2.3 LLM-Based Model

An in-depth investigation into the LLM (Large Language Model)-based approach, elucidating its methodology, evaluation criteria, GUI testing protocols, and concluding remarks. This segment offers insights into the efficacy of utilizing advanced language models for addressing complex design challenges in microelectronics.

1.2.4 Fine Tuning

A comprehensive overview of the experimental results obtained from fine-tuning methods, training approaches, response time analysis, and quality assessment of answers. Additionally, this section outlines potential avenues for future research and development in leveraging AI/ML techniques to further enhance PCB design methodologies.

Through these components, this report aims to provide a holistic understanding of how AI and ML methodologies can be effectively integrated into microelectronic system design workflows to optimize PCB development processes and mitigate design challenges.

Chapter 2

Literature

Chatbots, or conversational agents, have gained significant popularity in recent years due to advancements in natural language processing (NLP) and artificial intelligence (AI) technologies. These systems are designed to engage in human-like conversations and provide automated responses to user queries, making them valuable tools for various applications. Developing a chatbot involves two key challenges: effectively representing a sequence as a feature vector for analysis and designing a model capable of accurate and quick data identification. RASA supports these requirements by providing a robust framework that includes natural language understanding (NLU) and dialogue management components.

RASA's architecture typically employs an encoder-decoder structure to process user inputs and generate responses.Printed Circuit Board (PCB) design is a critical aspect of modern electronics manufacturing, requiring a high degree of precision and expertise. Traditionally, PCB design has relied on manual processes and the experience of skilled engineers. However, the increasing complexity of electronic systems and the demand for faster development cycles have necessitated the integration of advanced technologies to streamline the design process. Signal integrity and power integrity are critical concerns in the design of high-speed and high-frequency electronic systems. SI deals with ensuring that signals transmitted within a system maintain their fidelity without distortion or interference, while PI focuses on ensuring a stable and reliable power distribution network within the system. Traditional methods for addressing SI and PI issues often involve manual analysis, simulations, and design iterations, which can be time-consuming and resource-intensive.

Chatbots, powered by natural language processing (NLP) and artificial intelligence (AI), could potentially streamline and automate certain aspects of this process. The project explores the integration of RASA NLU, a popular NLU framework, with Llama Models for developing a chatbot system. The proposed a system which aims to automatically learn and answer questions specific to Singal Integrity and Power Integrity for PCB development. Chatbots are a major advancement in PCB design, with the potential to streamline design processes, enhance knowledge management, and boost collaboration. Despite existing

challenges, the ongoing development of AI and NLP technologies is set to integrate chatbots into PCB design comprehensively. Utilizing chatbots, designers can work more efficiently, minimize errors, and speed up the creation of innovative electronic systems.

Chapter 3

GUI

The Graphical User Interface (GUI) plays a crucial role in facilitating the interaction between designers and the PCB design software. This section provides an overview of the importance of a user-friendly GUI in microelectronic system design and explains how it contributes to enhancing the overall design process.

3.1 ChatBot

In this section, we explore the integration of a ChatBot system hosted on a server, specifically designed to assist users in Signal Integrity (SI), Power Integrity (PI), and Transmission Line analysis. Utilizing the Intent-Based Model and a Large Language Model (LLM), this ChatBot responds to user queries and guides in navigating topics related to SI, PI, and transmission line analysis. Additionally, by seamlessly integrating with simulation tools like LTspice [1] and ECADSTAR [12], users gain easy access to perform basic simulations directly within the chat interface

3.1.1 Overview

The ChatBot GUI serves as a central hub for users to interact with the design software. It features a clean and intuitive layout, facilitating efficient communication and problem-solving. The interface is divided into several sections, each catering to specific tasks and functionalities.

3.1.2 Components

- **Side Panel**: Located on the left side of the interface, the side panel provides quick access to essential features and functions. It includes buttons for switching between different modes that are used in the field of SI, PI, and LTspice simulation. Additionally, it incorporates logos of collaborating entities, enhancing brand visibility and partnership acknowledgment.

- **Chat Display**: Positioned prominently within the main interface, the chat display presents a chronological view of the conversation between the user and the ChatBot. Messages exchanged between the user and the ChatBot are displayed here, along with any accompanying images, formulas, or diagrams.

- **User Input**: Situated below the chat display, the user input section comprises a text input field and a send button. Users can type their queries or commands into the text input field and press the send button to submit their message to the ChatBot. This streamlined interaction mechanism ensures smooth communication between the user and the software.

- **Overlay**: The overlay serves as a visual indicator of certain states or actions within the interface. It is displayed as a semi-transparent layer over the main content area, providing focus and context for specific operations, such as modal dialogs or pop-up windows.

3.1.3 Functionality

- **Message Exchange**: Users can engage in natural language conversations with the ChatBot by typing their queries or commands into the message input field. The ChatBot processes these messages and responds accordingly, providing relevant information, guidance, or assistance based on the user's input.

- **Dynamic Content Updates**: The GUI dynamically updates its content based on user interactions and selections. For instance, when the user switches between SI and PI modes, the relevant content and options are displayed accordingly, ensuring a personalized and context-aware user experience.

- **Integration with LTspice**: The GUI seamlessly integrates with LTspice, a powerful simulation tool for analyzing transmission lines. Users can input simulation parameters directly into the interface and generate plots to visualize the results, enhancing the design exploration and validation process.

3.1.4 Visual Representation

The layout and key components of the ChatBot GUI are illustrated in Fig. 3.1.

3.1.5 User Interaction

Users can interact with the ChatBot GUI in the following manner:

1. Type messages or commands into the message input field and press send.

2. Receive responses from the ChatBot in the chat display area.

Figure 3.1: ChatBot GUI

3. Select the desired mode (SI, PI, LTspice) from the side panel.

4. Interact with dynamic content and options based on the selected mode.

3.1.6 Run Commands

To launch the ChatBot system, the following commands are used:

1. **rasa run -m models –enable-api –cors "*"**: This command starts the Rasa server with the specified model files (‘-m models‘) and enables the API (‘–enable-api‘). The ‘–cors "*"‘ flag allows cross-origin resource sharing (CORS) for web applications, enabling communication between the ChatBot frontend and backend.

2. **rasa run actions –cors "*"**: This command starts the Rasa action server, which handles custom actions defined in the ChatBot’s domain. Custom actions include fetching data from external APIs, performing database operations, or executing any other backend logic required to respond to user queries [5].

3. **python LTspice_API.py**: This command runs a Python script (‘LTspice_API.py‘) responsible for integrating the ChatBot with LTspice, a simulation tool for analyzing transmission lines. The script facilitates communication between the ChatBot interface and LTspice, allowing users to input simulation parameters and visualize the results directly within the ChatBot environment.

These commands must be executed in the respective directories where the ChatBot system and LTspice integration scripts are located. Once executed, the ChatBot system

will be up and running, ready to interact with users and assist them with Signal Integrity (SI), Power Integrity (PI), and Electromagnetic Compatibility (EMC) analysis during the PCB design process.

3.2 SI - Signal Integrity

The SI (Signal Integrity) window within the GUI serves as an interface for conducting simulations crucial for analyzing signal integrity within the circuit. It enables users to simulate both differential pair and single-ended circuits, providing valuable insights into signal behavior using software like LTspice and ECADSTAR. This interactive approach allows users to clarify doubts and simulate designs before implementing them in real-life scenarios.

In particular, the SI group utilizes the DHSTL topology for both differential and single-ended circuits within these simulations. They collect simulation results across a range of different values, which are then used to train AI models. By simulating various configurations and parameters, the group gathers a comprehensive set of data. This data is essential for training AI models to predict signal integrity issues more accurately. Through this process, the simulation tools not only aid in immediate design verification but also contribute to the development of advanced AI-driven analysis techniques for signal integrity.

Additionally, recent research by Zabinski et al. [11] has demonstrated the applications of optimization routines in signal integrity analysis, providing further insights into the optimization techniques utilized in signal integrity studies.

3.2.1 Differential Pair Circuit Simulation

Differential signaling, renowned for its resilience against noise and cross-talk, finds extensive application in high-speed digital circuits. The SI window facilitates the simulation of differential pair circuits, comprising two parallel conductors conveying equal and opposite signals to minimize electromagnetic interference and enhance noise immunity.

The SI window offers functionality for simulating a differential pair circuit, as shown in Figure 3.2

Users input essential parameters, including resistor values ($R8$, $R9$) in ohms, characteristic impedance values ($DL1$, $DL2$) of the parallel transmission line in ohms, and their respective lengths ($DL1$, $DL2$) in millimeters. This data is processed by ECADSTAR software to generate an eye diagram, providing a visual representation of signal performance.

3.2.2 Single-ended Circuit Simulation

Despite their simpler design compared to differential pairs, single-ended circuits remain prevalent in various applications. Users can simulate single-ended circuits within the SI window, inputting parameters such as resistor values ($R1$) in ohms and characteristic

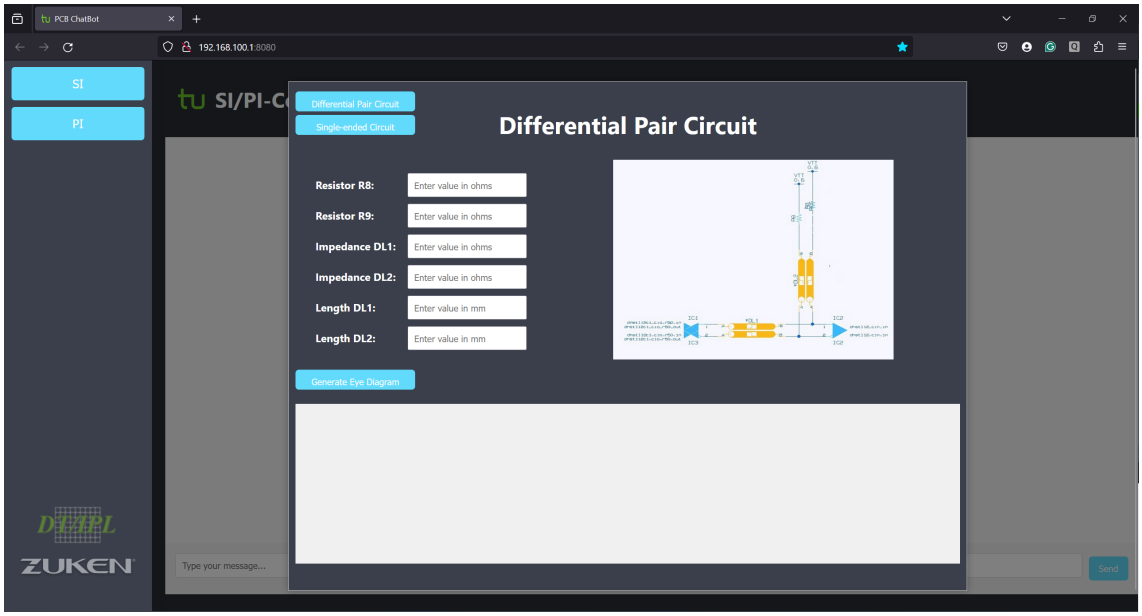

Figure 3.2: Differential Pair Circuit Simulation Window

impedance values ($TL1$, $TL2$) of the transmission line in ohms, along with their respective lengths ($TL1$, $TL2$) in millimeters. ECADSTAR software leverages this information to generate an eye diagram, offering insights into signal behavior and integrity (see Figure 3.3).

3.2.3 Eye Diagram Visualization

Ecadstar software is capable of generating the eye diagram, as shown in Figure 3.4. This visual representation aids in comprehending signal integrity, highlighting important characteristics such as jitters, noise margin, and timing margins. The graph is given in voltage (V) versus Time (ns). It enables engineers to assess the robustness of the design and identify areas for improvement.

However, our initial attempt to generate the eye diagram directly from eCADstar encountered a limitation: the software lacks an independent API specifically for this purpose. As of now, this functionality is not available.

Due to this limitation, we shifted our approach to utilizing LTspice software, which offers an independent API for generating graphs relevant to signal integrity analysis, particularly in the domain of transmission line analysis, as discussed in the last section, "Transmission Line Analysis."

While LTspice currently serves as our solution, we remain open to revisiting eCADstar's capabilities in the future, should the software provide suitable APIs for integration.

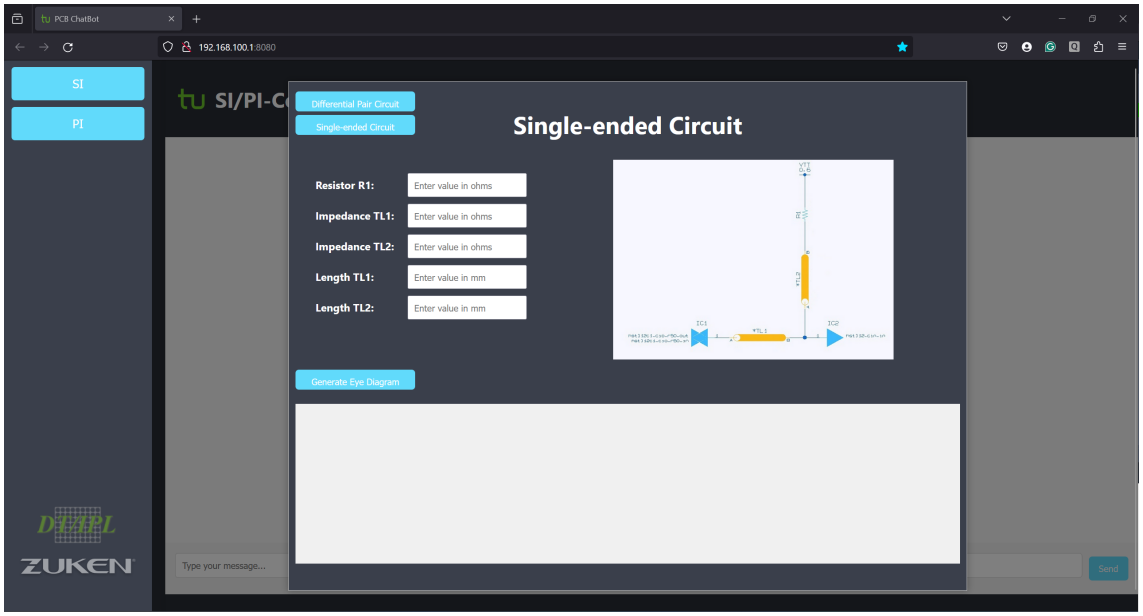

Figure 3.3: Single-ended Circuit Simulation Window

3.3 PI - Power Integrity

The Power Integrity (PI) simulation window integrated into our chatbot's graphical user interface (GUI) is designed to perform detailed power integrity analyses using the ECADSTAR platform. This feature provides users with a comprehensive tool to simulate and evaluate the frequency impedance response of their electronic designs, ensuring optimal performance and reliability of their power delivery networks (PDNs).

3.3.1 Key Components

The PI simulation window is intuitively designed to facilitate easy navigation and operation. It includes the following key components:

- **Input Fields:** Users can specify parameters such as simulation frequency range and impedance range.
- **Control Buttons:** To start the simulation with the given user inputs and view the results.
- **Display Area:** A graphical display area where the impedance versus frequency response is plotted in real-time after the simulation and also display the best possible decap location.

3.3.2 Simulation

Currently, we have a predefined square board with 12 decoupling capacitors and 2 ICs as shown in Figure 3.6. The PCB design file is imported to ECADSTAR. User first sets the

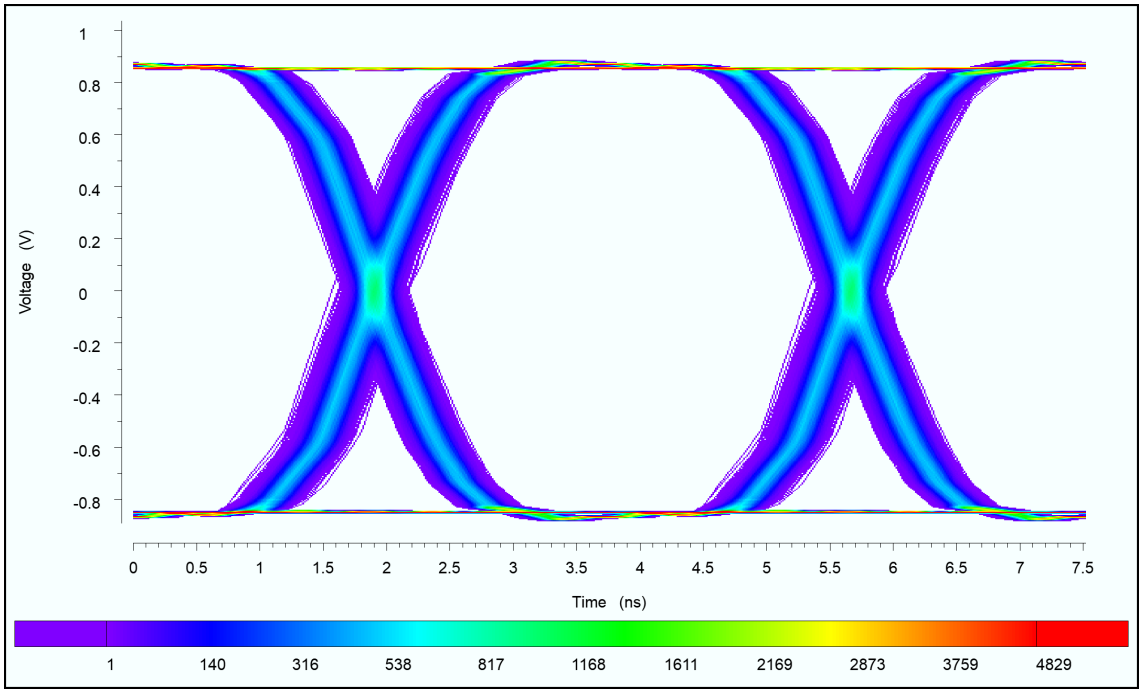

Figure 3.4: Eye Diagram for Signal Integrity Analysis

input values like frequency maximun and minimun range and silimarly for impedence,then initiate the simulation with the 'frquency-response' button in the GUI. The PI agent processes the input data to calculate the impedance profile of the PDN across the specified frequency range by running several iterations.It also predicts the best possible position and number of decaps that could be placed on the board.

3.3.3 Simulation Result

After the simulation is complete the user can view the result using the view-result button. The GUI displays the given board with possible decaps location.It also plots the impedance response over the frequency spectrum for both of the ICs as shown in figure 3.7. Peaks in the impedance plot indicate potential resonances, which may affect the stability and performance of the power delivery system.

The PI simulation feature in our chatbot's GUI represents a significant advancement in providing accessible and effective power integrity analysis tools. By leveraging ECADSTAR's robust simulation capabilities within a user-friendly interface, we empower designers to achieve superior power integrity in their electronic systems, ensuring reliable and high-performance products.

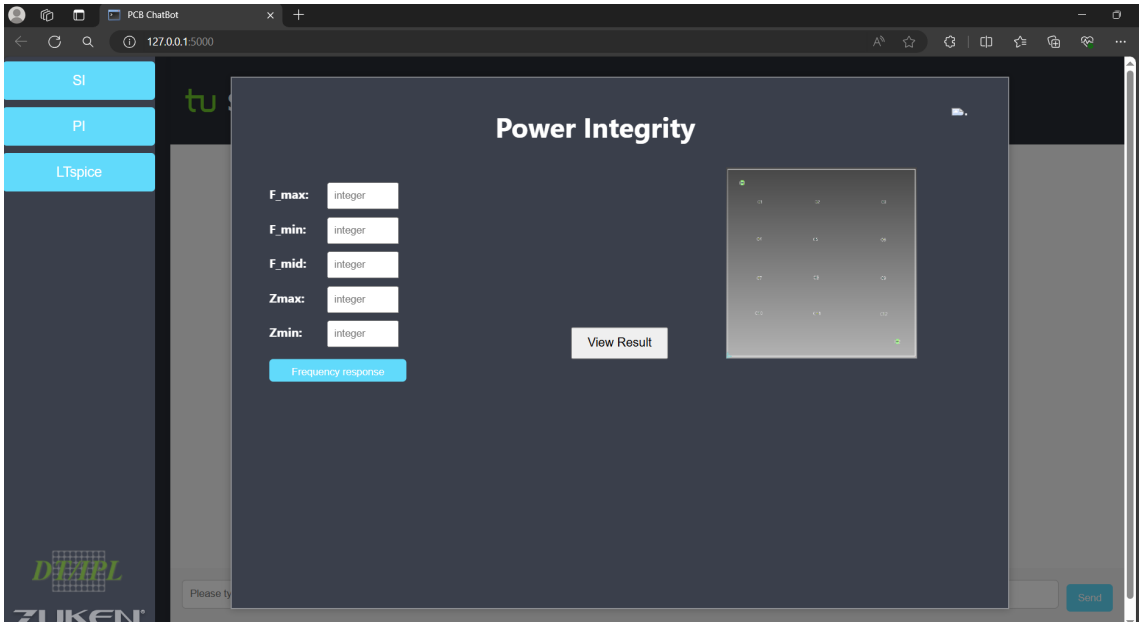

Figure 3.5: GUI for Power Integrity

3.4 Transmission Line Analysis

Transmission lines are vital components in high-frequency circuits, enabling the efficient transfer of signals while minimizing signal distortion. Users can analyze signal behavior along transmission lines through simulations using the LT-spice software within the GUI. Here, star-topology and daisy-chain-topology simulations have been integrated, allowing users to easily visualize the effects by inputting the required values.

3.4.1 Star-Topology Analysis

In a star topology, the signal is distributed from a central point to multiple branches, resembling a star shape. This topology is often used in scenarios where a central signal needs to be transmitted to multiple endpoints with minimal reflection and signal loss. Figure 3.8 illustrates a typical star-topology configuration.

The simulation of a star topology can be performed using the Star-topology window in GUI. The inputs for this simulation are:

- $R1$: Resistance in ohms for the first branch
- R: Resistance in ohms for the central node
- $Z1$: Characteristic impedance in ohms for the first branch
- Z: Characteristic impedance in ohms for the central node
- $L1$: Length in millimeters for the first branch

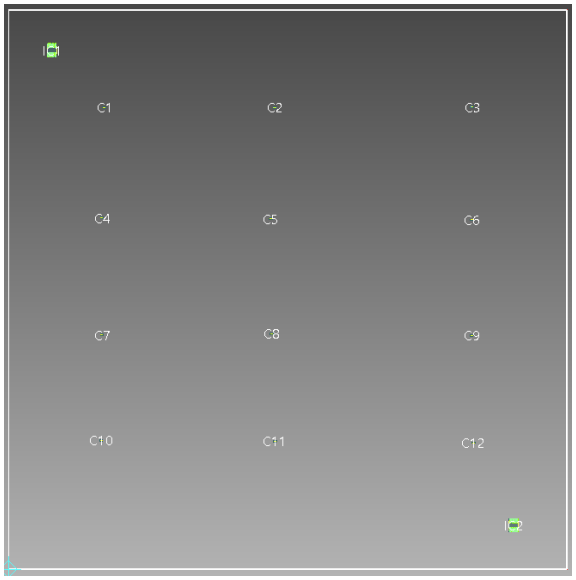

Figure 3.6: Predefined square board with 10 decoupling capacitors and 2 ICs

- L: Length in millimeters for the central node
- $DielectricConstant$: Value for the dielectric constant

Figure 3.9 illustrates the user interface for the star-topology simulation.

The output of the star-topology simulation provides a plot of voltage versus time, showing how the voltage travels over time. Figure 3.10 illustrates an example output of the star-topology simulation.

Users can also open the Star-Topology Analysis window directly by sending the command "Open the Star-Topology window" or similar commands such as "Launch Star-Topology window" or "Activate Star-Topology window" to the chatbot, enhancing the user experience and streamlining the workflow.

3.4.2 Daisy-Chain Analysis

In a daisy-chain topology, the signal is passed from one device to the next in a linear sequence. This topology is commonly used in scenarios where the signal needs to be transmitted through multiple devices or stages, making it important to manage reflections and signal integrity at each connection point. The simulation of daisy-chain topology can be performed using the LT-spice software. Figure 3.11 illustrates a typical daisy-chain topology configuration.

The simulation of daisy-chain topology can be performed using the DaisyChain-topology window in GUI. The inputs for this simulation are:

- RT: Termination resistance in ohms
- $L1$: Length in millimeters for the first segment

Figure 3.7: PI Simulation result

- $L2$: Length in millimeters for the second segment
- $L3$: Length in millimeters for the third segment
- $L4$: Length in millimeters for the fourth segment
- $L5$: Length in millimeters for the fifth segment
- Z: Characteristic impedance in ohms for the transmission line
- $DielectricConstant$: Value for the dielectric constant

Figure 3.12 illustrates the user interface for daisy-chain topology simulation.

The output of the daisy-chain topology simulation provides a plot of voltage versus time, showing how the voltage travels through the circuit over time. Figure 3.13 illustrates an example output of the daisy-chain topology simulation.

Users can also open the Daisy-Chain Topology Analysis window directly by sending the command "Open the DaisyChain-Topology window" or similar commands such as "Launch DaisyChain-Topology window" or "Activate DaisyChain-Topology window" to the chatbot, improving accessibility and efficiency in the analysis process.

3.5 Conclusion

The ChatBot GUI offers a user-friendly and interactive platform for designers to address SI/PI/EMC challenges during the PCB design process. By leveraging natural language

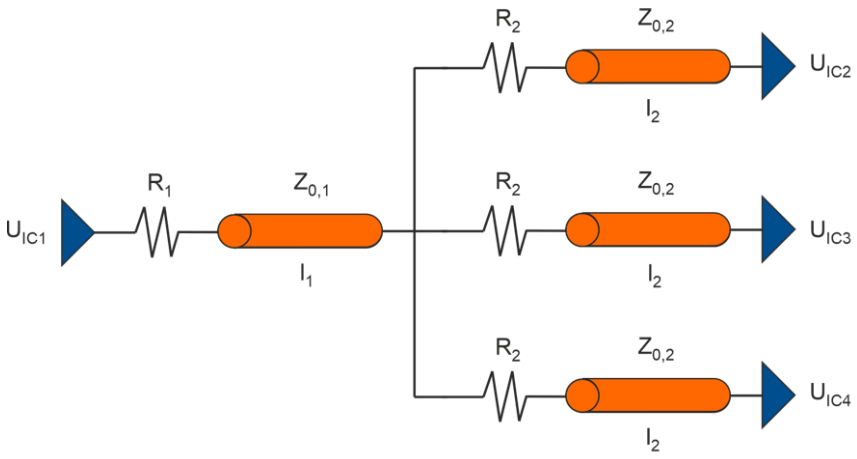

Figure 3.8: Star-Topology Configuration

processing and dynamic content updates, the GUI enhances productivity, fosters collaboration, and accelerates design iterations, ultimately leading to optimized microelectronic system designs.

By analyzing the voltage waveforms, engineers can identify signal integrity issues such as impedance mismatches, reflections, and distortion. This information is invaluable for refining transmission line designs and ensuring dependable signal transmission in high-speed circuits.

Integrating ECADSTAR and LTspice simulations within the GUI empowers engineers to conduct comprehensive analyses of signal integrity and transmission line behavior. This integration facilitates informed design decisions and optimizations for high-performance electronic systems.

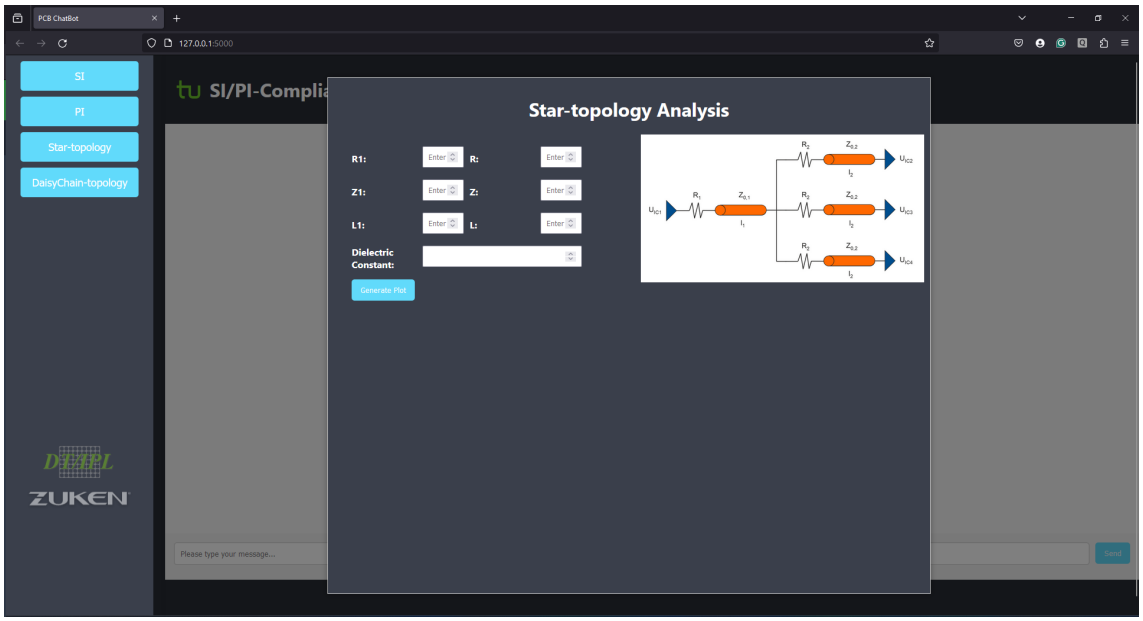

Figure 3.9: Overview of the Star-Topology Window

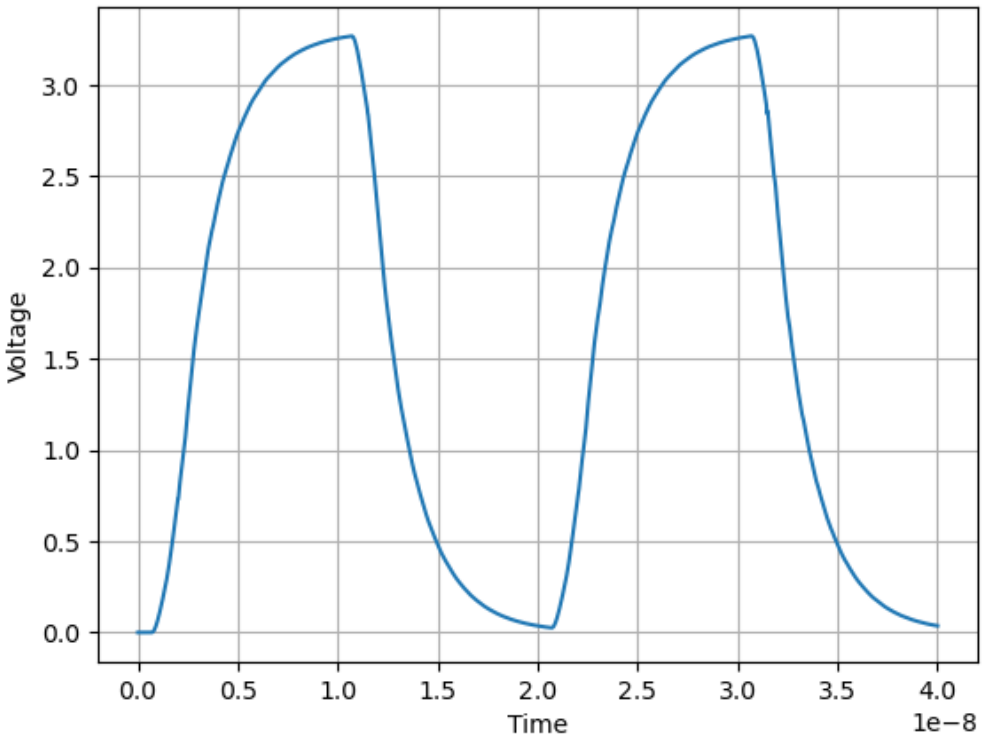

Figure 3.10: Output of the Star-Topology Simulation

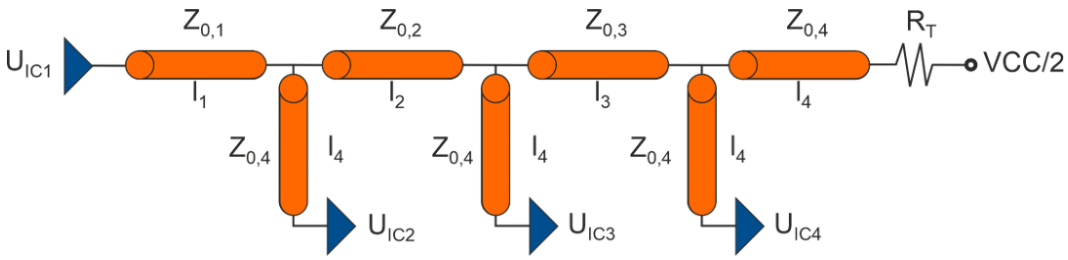

Figure 3.11: Daisy-Chain Topology Configuration

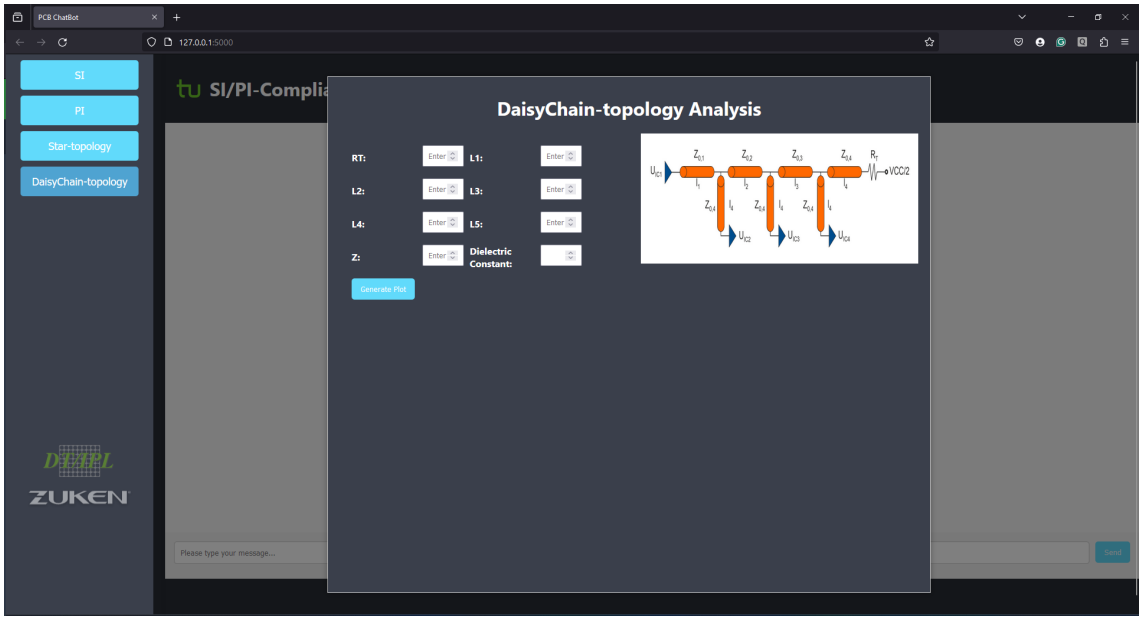

Figure 3.12: Overview of the Daisy-Chain Topology Window

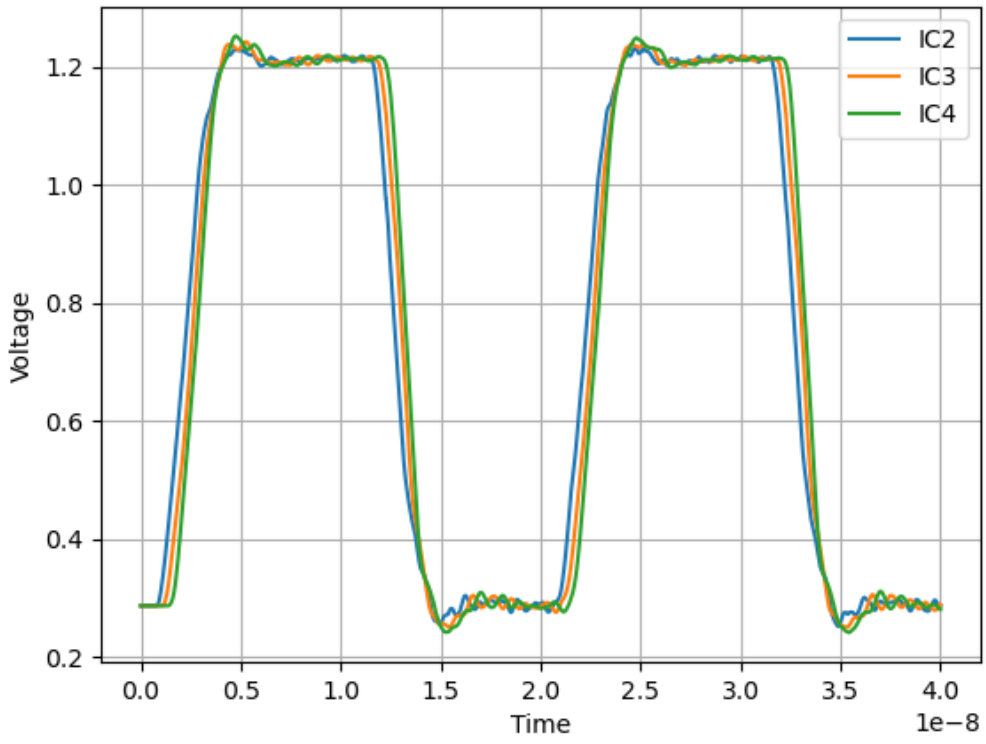

Figure 3.13: Output of the Daisy-Chain Topology Simulation

Chapter 4

Intent Based Model

4.1 Methodology

The first level of the model is based on the Rasa framework which is based on understanding the intent behind the messages given by the user and providing an appropriate response based on already stored responses. It is based on an NLU (natural language understanding) that classifies the questions provided by the user into predefined categories and then it takes relevant actions.

The workflow is divided into the following steps:

1. **Collection of test data**: Gathering of data as per the required domains (in our case Signal Integrity and Power Integrity) and classifying them into intents.

2. **Intent Classification**: The gathered data is used to train a machine learning model based on NLP (Natural Language Processing) techniques, to recognize the intents of the messages given by the user.

3. **Entity Extraction**: The Chatbot also extracts relevant entities from the user message. Entities represent specific pieces of information that the bot needs to understand or act upon.

4. **Dialogue Management**: Once the intent and entities are identified, the bot needs to determine how to respond or act based on this information. Dialogue management involves using predefined rules or machine learning algorithms to generate appropriate responses or trigger relevant actions. This may involve accessing external APIs, databases, or other systems to retrieve information or perform tasks.

5. **Feedback Loop**: As the user interacts with the bot, it learns and optimizes its performance.

4.1.1 Collection of test data and sources

The data used for training within the Chatbot has been accumulated from many different books, online sources, and research articles. It was mainly manual scraping of data to get answers which are relevant for the task at hand. In the current version, as mentioned before, we have only developed the bot for the sub-domain "Decaps". In which we have configured 156 intents for initial training of the bot. For the intents which basically are used to understand the user query and then provide an appropriate response as per the configured domain. For this we have manually written the first question and then the duplicated questions were generated using ChatGPT.

4.1.2 NLU Pipeline

The NLU (Natural Language Understanding) pipeline comprises the steps used to identify the user input and extract meaning from it. The workflow of a typical Rasa NLU pipeline is shown in Figure 4.1

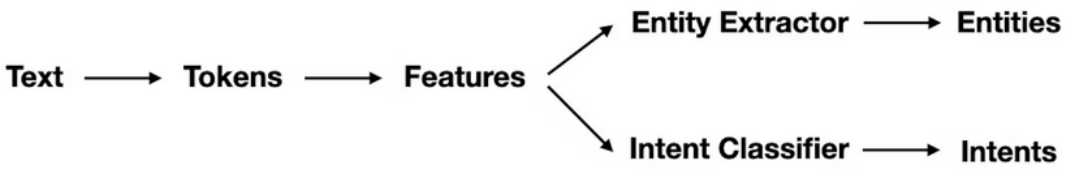

Figure 4.1: Rasa NLU Pipeline

1. **Tokens**: When a user provides an input, the first step is to split it into smaller chunks, where each chunk is called a token. This classification is done before featurization using machine-learning models.

2. **Features**: Once tokens are created, each token is assigned numeric features which the machine learning model uses. These are classified into Sparse and Dense features.

3. **Intent Classification**: After the features are assigned to the tokens it is passed to an intent classification model.
   Rasa provides their DIET (Dual Intent and Entity Transformer) model which allows for both intent and entity classification. It leverages the power of transformers and embeddings to process natural language inputs efficiently. It is easily provides features to use pre-trained embeddings. ADD REF HERE

4. **Entity Classification**: Not all entities are required to be used with the DIET model, some basic entities can also use other simpler models for quicker classification.

4.1.3 Dialogue Management

Dialogue management is the component of a conversational AI system responsible for determining the flow of conversation and deciding how the system should respond to user inputs based on the current context of the dialogue. The bot has 3 different policies to decide which action to take at each step:

1. **RulePolicy**: It is based on rules described by the developer. It will predict response based on predefined rules in "rules.yml".
2. **MemoizationPolicy**: It checks if the current conversation is related to any of the defined stories in "stories.yml".
3. **TEDPolicy**: This uses machine learning to detect the next best response.

4.2 Working Model

Until now we have discussed the basic working principles of the Rasa intent-based bot. Now let us describe the model which has been created for the PCB Design domain. The model is currently developed for the sub-domain "Decaps", we have configured multiple intents that pertain to this domain with reference to SI and PI-based PCB design. But it can also be extended to different domains as shown in Figure 1.1 in the Introduction section.
The database includes intents, a total of 156, which are related to specific topics, each intent is defined using multiple variations of questions or inputs that the user can provide (generated using ChatGPT) REF. It also includes predefined responses, generated manually using various research papers, PDFs, and online sources REF to these intents which are to be displayed when the user gives the respective input. These intents are mapped to the responses using stories which define the correlation. Actions can be configured for multiple applications. The domain, nlu, stories, and rules are YAML scripts. The action file is a python script which issued to call external APIs, notebooks, scripts etc.

4.2.1 Intents

We have configured various intents in the "nlu.yml" file. In the current ChatBot, we have used one single subdomain to show the working and legibility. A sample intent is shown below:

```
- intent: ask_decap_definition
 examples: |
   - What is a Decoupling Capacitor?
   - Explain Decoupling Capacitors to me.
   - Can you tell me about Decoupling Capacitors?
```

```
- Describe the purpose of Decoupling Capacitors.
- I want to understand Decoupling Capacitors better.
- What does a Decoupling Capacitor do?
- How do Decoupling Capacitors work?
- Why are Decoupling Capacitors used in electronic circuits?
- Give me an overview of the role of Decoupling Capacitors.
- When should I use Decoupling Capacitors in my circuit?
- What happens if I don't use Decoupling Capacitors?
```

The more variants of the intent we have the better the performance, as then the intent classification model will have more variations to learn from and better understand the user input.

4.2.2 Domain

For every defined intent there should be a response mapped to it. These are defined in the "domain.yml" file. Currently these responses have been extracted from research papers, internet articles and books on the given topics. A sample domain is shown below:

```
utter_ask_decap_functions:
 - text: "Decoupling capacitors have the following functions:
   \n1. Eliminate High Frequencies
   \n2. Supply DC Power to Active Devices
   \n3. Voltage Stabilization"
```

It also stores the labels of all the intents which have been defined by the developer. Normally for each intent there should be a response defined.

4.2.3 Stories

Stories are used to map intents with responses and also train the dialogue management model. These define the responses the bot will provide when a specific input is received from the user. A sample story is shown below:

```
- story: ask_decap_definition1
 steps:
 - intent: ask_decap_definition
 - action: utter_ask_decap_definition
```

Stories can be defined in complex ways so that a conversational chain can be created wherein the context of the conversation can be maintained. It allows the user to add in details to get more information on a specific topic that has been started.

4.2.4 Actions

Actions are defined to run a custom code. In our model, we have configured an action to call the LLM model when the confidence of a response in the intent-based model is less. The user input is then fed to the LLM model which has various resources linked to it. The model then searches the response in these resources. More details about the functioning of the LLM model will be discussed in Chapter 5. In our defined action we have. COMPLETE LATER

4.3 Conclusion

In conclusion, the Rasa intent-based model uses a manually developed database to provide responses to user inputs. It is limited by the fact that the data should already be present in the database for the Chatbot to provide an appropriate response to the query. We can configure conversation chains and define custom actions to enhance the performance of the Chatbot. The Chatbot is configured optimally to respond with also formulas (defined in LaTeX) and also images stored in an online cloud server.

Chapter 5

LLM Based Model

5.1 Methodology

The next level of the model is based on an LLM (Large Language Model) for queries which can not be resolved using the Intent-based model. LLMs are systems which use millions or billions of parameters to understand and process human language. These are normally pre-trained with humongous datasets and can be tuned for specific applications. The working of an LLM is based on the following steps:

1. **Architecture**: Transformers rely on attention mechanisms to weigh the significance of different words in a sentence, allowing the model to capture dependencies and relationships between words more effectively.

2. **Pre-training**: Before being fine-tuned for specific tasks, LLMs undergo pre-training on vast amounts of text data. During pre-training, the model learns to predict the next word in a sequence given the preceding context. This process helps the model develop a rich understanding of language patterns and semantics.

3. **Fine-tuning**: After pre-training, LLMs can be fine-tuned on specific tasks by providing them with labelled examples. For example, if the task is sentiment analysis, the model would be trained on a dataset where each text sample is labelled with its corresponding sentiment (positive, negative, or neutral). Fine-tuning adapts the model to perform well on the target task.

4. **Inference**: Once trained, LLMs can generate text or perform various language-related tasks. During inference, the model processes input text and generates output based on its learned knowledge and patterns. This could involve completing a sentence, summarizing a document, translating languages, answering questions, or any other task it has been trained for.

5. **Continual learning**: LLMs can also be continually updated with new data to stay relevant and accurate over time. This ongoing learning process helps them adapt to evolving language usage and new information.

5.2 Working Model

In our version of the model we have configured it to take data from pdf sources and store them as small chunks, these chunks are then converted into Embeddings which are then used to search for answers in the stored vectors depending on the user query.
Many open source models such as LLaMA 2, Mistral, BERT etc. can be used for this purpose, for our purpose we have used LLaMA 2 model which has been developed by Meta. We chose this because LLaMA 2 is one of the most resource efficient LLM therefore it can run on variety of hardware platforms. It is also possible to train LLaMA 2 model for a specific use-case, in our case we use the chat version of the model. Although we have chosen this model it is quite easy to use a different model directly from the HuggingFace database. Additionally, a conversation chain has been created to keep context of the conversation allowing the user to ask follow-up questions.

5.2.1 Packages and Libraries

A wide range of libraries are needed to implement the specified functionality. The application of each of them is listed below:

1. **langchain**: It is used to develop LLM-based applications and integrate other powerful packages, to develop a highly functional chatbot application. It includes all necessary libraries including, HuggingFaceEmbeddings, Document Loaders, Text Splitters, Conversational Chain, Buffer Store, etc.

2. **bitsandbytes**: The bitsandbytes package is a lightweight Python wrapper around CUDA custom functions, in particular 8-bit optimizers, matrix multiplication and quantization functions.

3. **pypdf**: This package is used to work with PDFs. It allows to edit, merge, or transform pages of pdf files. It is used to read data from PDF sources.

4. **sentence-transformers**: This is a framework which allows efficient computation of dense vector representations.

5. **chromadb**: Used as a database to store documents, embeddings etc. In our case, the embedded chunks are stored here and then retrieved as required.

5.2.2 Workflow

The model used here is divided into multiple stages, allowing the user to access data from PDF sources and create an interactive Chatbot to create conversational chains.

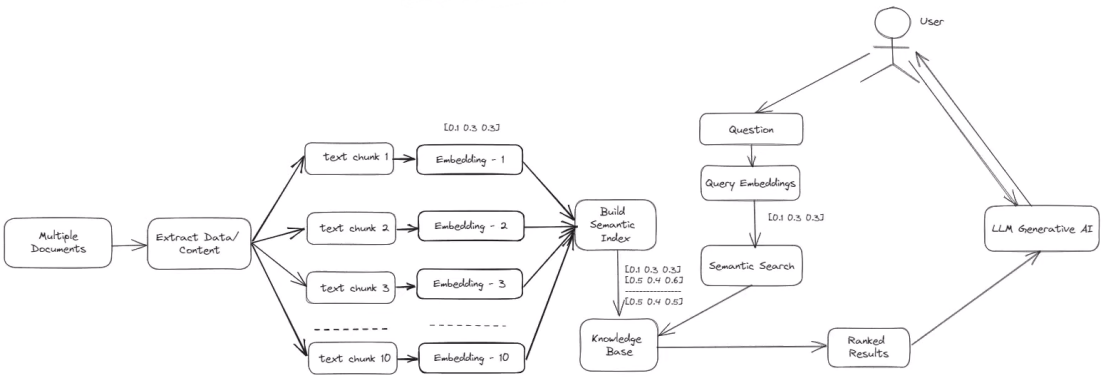

Figure 5.1: LLM Workflow

We have the following steps which allow this to happen:

1. **Extraction of data**: After selecting the required PDF sources, the first step is to extract the data as a text file. In our model, we have provisions to extract data not only from PDFs but also DOC files and TXT files. This extracted text is then stored as a large vector.

2. **Creating chunks of data**: LLM have restriction of token size, therefore it is not possible to feed the entire text vector to the model. Therefore we have to create small chunks of text which can then be processed to be used by the LLM. The langchain package includes a document_splitter function which allows us to create chunks of size defined by the user.

3. **Embeddings**: Each text chunk is transformed into a numerical representation called an embedding. Embeddings capture the semantic meaning of the text, making it easier for the model to understand and compare. Embeddings are vectors in a space where similar texts have similar vector representations. We have used "all-MiniLM-L6-v2" embeddings from HuggingFace, it is a Sentence-Transformers type of embedding that maps sentences to a 384 dimension vector space.

4. **Semantic Index**: The embeddings are used to create a semantic index, which organizes and indexes the text chunks based on their semantic content. This index allows for efficient searching and retrieval of relevant information.

5. **Knowledge Base**: The semantic index is stored in a knowledge base. This knowledge base acts as a repository where all the processed and indexed data are kept. In

our model, we have used ChromaDB to store all these embeddings, as it is optimized to store a large-scale of data and makes it easier for semantic search applications.

6. **User Query**: The user query is sent both to the LLM model and also to the Knowledge base. On the Knowledge base side, the question is first converted into an embedding which is similar to the one created earlier and then this is matched with the database using Semantic search to find the most appropriate result. The top-ranked results which closely resemble the query passed are then sent to the LLM, where the model takes the best result and then generates a natural response.

5.2.3 LLaMA2 Model

In the given use case we have used the LLaMA2 Model developed by Meta. It is available in multiple parameter sizes, we have initially tested the performance using a 7 billion parameter model as it can easily run on a local computer. The advantage of LLMs is that these can be trained for specific use cases, and multiple such pre-trained models are readily available to download from the HuggingFace database. We have used the LLaMA 7B-chat model, which has been tuned for conversational applications. Models with more parameters can be used to improve performance and response time but require higher computing power.

5.3 Conclusion

Therefore, we have developed a fully functioning LLM-based chatbot which can extract data from PDFs and provide responses to user queries and also create a conversational link.

Chapter 6

Fine tuning of LLM

6.1 Introduction

LLM finetuning of the base model allows users to train the model for the specific task with labeled data, which is called supervised fine tuning (SFT). OpenAI finetuning recommendations [4] show us that for a specific task fine-tuned, much smaller LLM (GPT 3.5 Turbo) model can outperform the bigger one (GPT-4). In our example we want to show the fine-tuned model performance on local server, which is relevant for a lot of companies who want to reform their knowledge systems based on LLMs, as this approach allows hundred percent security of internal data usage. For this task we have chosen Llama 3, which has been released on April 2024and the most capable openly available LLM to date[3].

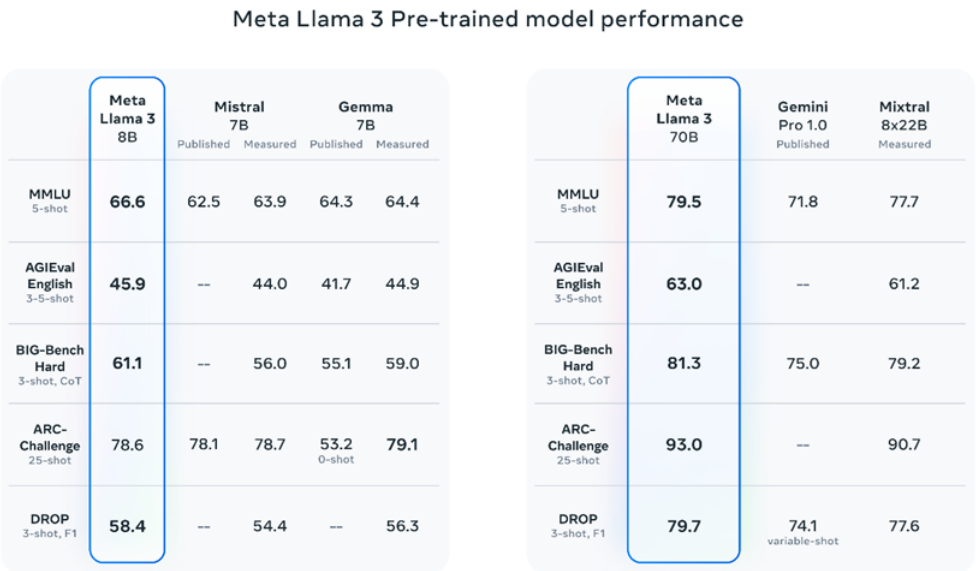

Figure 6.1: Llama3 performance

6.2 Setting Up the Environment

Nowadays almost every day a new open-source large language models (LLMs) or update for existing one is coming to the market and fine tuning of LLMs is one of the most essential parts to train models for Chat Bots and it usually requires a lot of effort to write codes for this process.
Using Llama Factory, a framework for fine tuning hundreds of LLMs without coding effort and skills[9], allows almost everyone to create their own database for specific domains and train them easily. Furthermore, it allows users to compare the efficiency of LLMs using the same and different training methods. Installation instructions for the framework are described in github[2] documentation.

6.3 Preparing the Dataset

Creating a high-quality and representative dataset for fine-tuning is crucial. The dataset for fine-tuning task is created in alpaca[7] format (collected as described in intent based model) which is supported on Llama-Factory framework. An example of "training data" looks as follow:

```
{"instruction": "How can I determine the appropriate placement of decoupling capacitors
  in a circuit?",
 "input": "",
 "output": "Decoupling capacitors should be placed as close as possible
  to the source for decoupling the signal."
 "text": "",
}
```

The data fields are as follows[6]:

- **instruction**: describes the task the model should perform. All instructions in the dataset is unique.
- **input**: optional context or input for the task. For example, when the instruction is "Summarize the following article", the input is the article. As it is optional, not defined in the dataset.
- **output**: the answer to the instruction.
- **text**: the instruction, input and output formatted with the prompt template used by the authors for fine-tuning their models, not defined in the dataset as it is optional.

6.4 Fine-Tuning of Llama3 Model

For our project we chose Llama 3 Model (has base and instruct models with 8B and 70B parameters) as we want to get best of from its text generation capabilities. Main differences between these models are:

Feature	Llama 3	Llama 3 Instruct
Type	Base Model	Fine-tuned Model
Usage	General text processing	Following instructions
Strength	Predicting next word, broad know. base	Understanding instruct., conversation
Weakness	Requires specific prompts	Limited to tasks it's fine-tuned for

From the table below we can conclude that with our setup (3 RTX A6000 GPUs with dedicated memory of 47.5 GB each and shared memory of 256 GB) we can train Llama3 8B using LoRA/QLoRA methods in any bit-depth and 70 B model in 4 bit bip-depth via QLORA on a single GPU and using multiple GPUs (distributed training) we can execute full parameter fine-tuning of Llama3 8B model as well.

Method	Bits	7B	13B	30B	70B	110B	8x7B	8x22B
Full	AMP	120GB	240GB	600GB	1200GB	2000GB	900GB	2400GB
Full	16	60GB	120GB	300GB	600GB	900GB	400GB	1200GB
Freeze	16	20GB	40GB	80GB	200GB	360GB	160GB	400GB
LoRA/GaLore	16	16GB	32GB	64GB	160GB	240GB	120GB	320GB
QLoRA	8	10GB	20GB	40GB	80GB	140GB	60GB	160GB
QLoRA	4	6GB	12GB	24GB	48GB	72GB	20GB	96GB
QLoRA	2	4GB	8GB	16GB	24GB	48GB	18GB	48GB

6.5 Execution

Following configurations are defined in llama3.json file for execution:

General Configuration

- **stage: "sft"** - specific parameter within Llama-Factory environment, "sft" indicates "supervised fine-tuning".
- **do_train: true** - specific parameter within Llama-Factory environment for training, set to "true" for training.

Model and Data

- **model_name_or_path: "meta-llama/Meta-Llama-3-8B"** - model used for fine-tuning, here Llama 3 with 8B parameters.

- **dataset: "decap_data"** - name of the dataset used for fine-tuning.

Fine-tuning Specifics

- **template: "llama3"** - configuration template for fine-tuning, here a base Llama 3 model.
- **finetuning_type: "lora"** - type of fine-tuning algorithm used in fine-tuning, here "lora" refers to Low-Rank Adaptation.
- **lora_target: "all"** - this parameter controls which parts of the model are adapted during LoRA fine-tuning, "all" indicates all layers will be adapted.

Training Configuration

- **output_dir: "ft_8bit"** - directory where the fine-tuned model will be saved.
- **per_device_train_batch_size: 3** - number of training examples processed per device in a single training step (depends on GPU memory).
- **gradient_accumulation_steps: 4** - number of training steps gradients are accumulated before performing an update.
- **lr_scheduler_type: "cosine"** - type of learning rate scheduler used during training, here "cosine" refers to a cosine annealing scheduler.
- **logging_steps: 10** - how often training metrics are logged during the process, every 10 steps in this case.
- **warmup_ratio: 0.1** - proportion of training steps for a gradual increase in learning rate before reaching the full value.
- **save_steps: 1000** - how often the model is saved during training, every 1000 steps in this case.
- **learning_rate: 5e-5** - initial learning rate used for training the model.
- **num_train_epochs: 3.0** - total number of epochs (iterations over the entire dataset) to train the model.
- **max_samples: 500** - limits the number of training samples used, optional limit of 500 samples here.
- **max_grad_norm: 1.0** - maximum norm for gradients, helps prevent exploding gradients.

Quantization and Optimization

- **quantization_bit: 8** - specifies the target bit-depth for model quantization (reducing model size by representing parameters with fewer bits), is set to 8 bits in this case.
- **loraplus_lr_ratio: 16.0** - specific parameter to LoRA fine-tuning which controls the learning rate ratio for certain parts of the model.
- **fp16: true** - whether to use mixed precision training with 16-bit floating-point numbers (can improve training speed).

The fine-tuning involves executing the scripts in the Llama-Factory either via WebUI or via cli framework, which allows fine-tuning the model in command window. Llama-Factory pre-built WebUI does not support multiple GPU setup, which is the case in our server, that is why cli framework with following command is used for training:

- llamafactory-cli train llama3.json

After training is completed, with following command we can access our trained model:

llamafactory-cli chat –model_name_or_path meta-llama/Meta-Llama-3-8B –adapter_name_or_path ft_8bit –template llama3 –finetuning_type lora

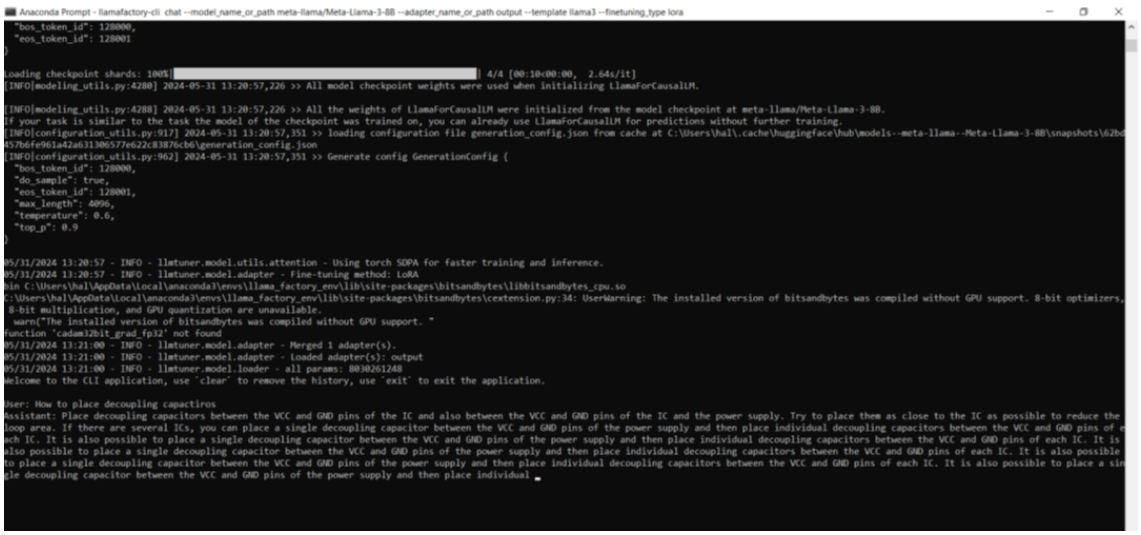

Figure 6.2: Accessing trained model via command window

6.6 Evaluating the Model

Evaluation of the LLMs performance is difference for each scenario and different general task benchmarks use various parameters as language understanding, coherence, context-sensitivity, reasoning etc. to evaluate model performance. In our case the model is defined for certain task in one domain which requires usage of specific downstream benchmarks as Question-answering benchmarks, which become fundamental component in the assessment of LLMs and their overall performance and key factors in evaluation is listed in tables below[10].

Key metrics of automatic evaluation

General Metrics	Metrics
Accuracy	Exact match, Quasi-exact match, F1 score, ROUGE score
Calibrations	Expected calibration error, Area under the curve
Fairness	Demographic parity difference, Equalized odds difference
Robustness	Attack success rate, Performance drop rate

Key factors in human evaluation

Evaluation Criteria	Key Factor
Number of evaluators	Adequate representation, Statistical significance
Evaluation rubrics	Accuracy, Relevance, Fluency, Transparency, Safety
Evaluator's expertise level	Relevant domain expertise, Task familiarity, Methodol. training

Following results were automatic generated after the training:


```
{
    "epoch": 3.0,
    "total_flos": 6460108876873728.0,
    "train_loss": 2.086656379699707,
    "train_runtime": 150.9668,
    "train_samples_per_second": 3.577,
    "train_steps_per_second": 0.099
}
```


As the table above from[10] shows, human evaluation is crucial for performance analysis of the trained models. But in our case, we could not accomplish this type of evaluation as the trained model responses include hallucinations and some links, which were not in the training dataset, which should be further investigated.

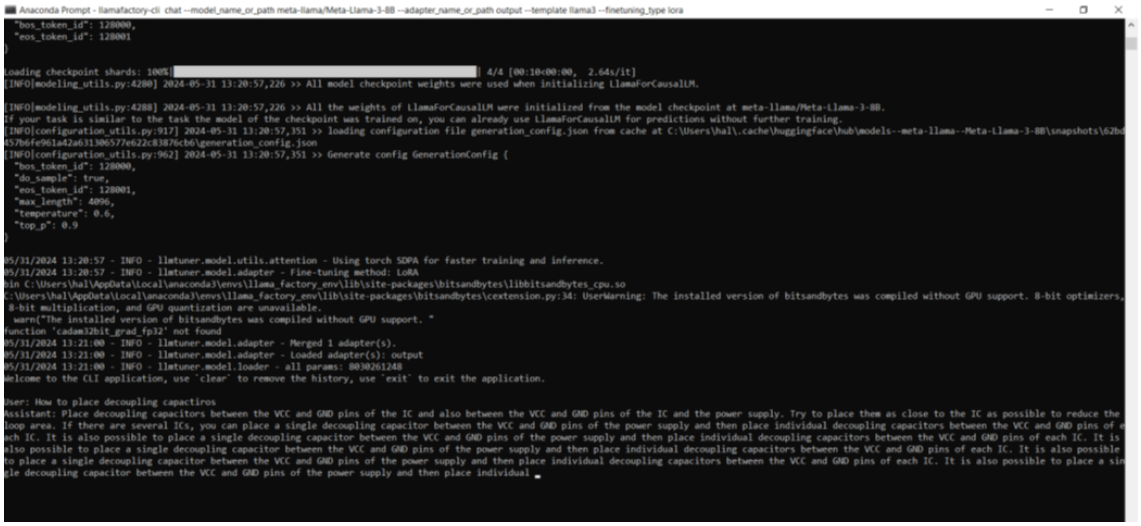

Figure 6.3: Hallucination

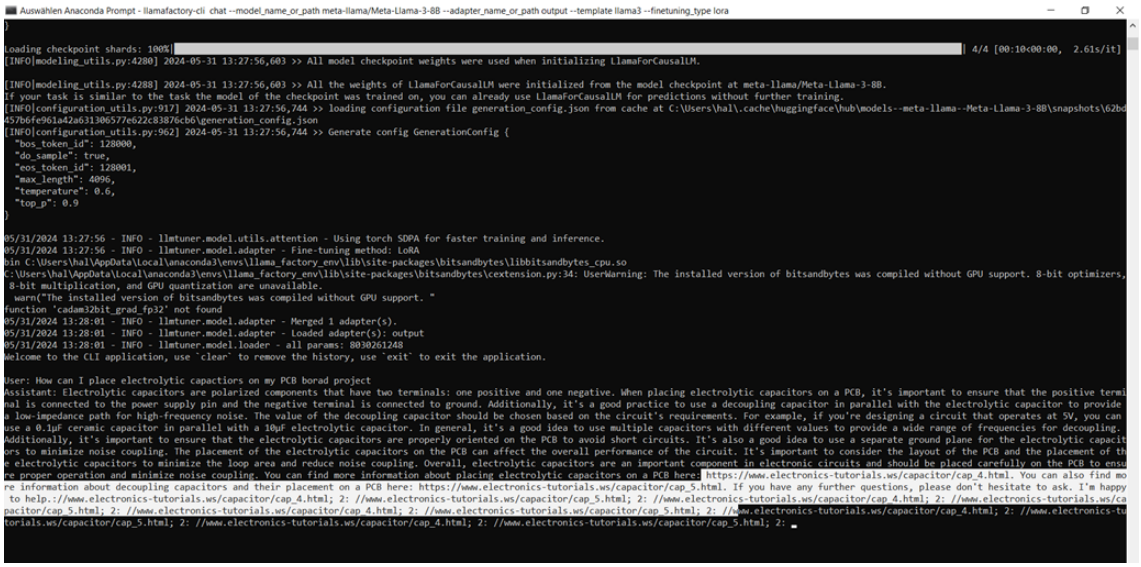

Figure 6.4: Unknown Link

Chapter 7

Conclusion

7.1 Conclusion

Within this project we have successfully implemented a Chat-bot using an intent based model of Rasa, which is complemented with powerful LLM such as Llama for text generation based on fine-tuned data and searching from different digital sources.
In addition to that we have built a WebUI which is integrated not only with our chat-bot but also with the other project gorups, which allows user to run scripts and make predictions in SI PI domains as well. While current system has a good performance on text based interactions between the user and Chat-bot, there is still some opportunities to enhance its functionality and user experience.
Following potential areas for future work that could improve the Chat-bot's performance and add more domain to extend its application.

7.2 Future work

1. **Exploring Different Types of LLMs**: The first immediate area for further work would be an integration of different types of LLMs for the same task. With that we could investigate and compare how effective smaller LLMs like Mistral is or test other competitors like Gemma not only with benchmarks but also with computational efficiency and resource requirements.

2. **Implementing Multimodal LLMs**: Another possible direction for future work would be the implementation of Multimodal LLMs. These models can understand and process data from different sources such as text, image, audio and video, which would extend the chatbot's capabilities and allow users to send circuit designs for analysis or simply communicate via voice to do a little brainstorming.

3. **Enhancing User Experience**: Along with these technical improvements, future work should also focus on enhancing the user experience. This includes developing

more intuitive user interfaces and providing more personalized interactions based on the user's history and preferences.

List of Figures

Bibliography

[1] Analog Devices, Inc. *LTspice XVII*. Software available from Analog Devices, Inc. 2023. URL: https://www.analog.com/en/design-center/design-tools-and-calculators/ltspice-simulator.html.

[2] hiyouga. *Llama Factory*. 2023. URL: https://github.com/hiyouga/LLaMA-Factory.

[3] Meta. *Llama 3*. 2024. URL: https://https://ai.meta.com/blog/meta-llama-3.

[4] OpenAI. *OpenAI: Fine-tuning*. 2023. URL: https://platform.openai.com/docs/guides/fine-tuning.

[5] Rasa Documentation. *Running the Action Server*. 2024. URL: https://rasa.com/docs/action-server/running-action-server.

[6] tatsu-lab. *Alpaca$_H$F*. 2023. URL: https://huggingface.co/datasets/tatsu-lab/alpaca.

[7] tatsu-lab. *Stanford$_A$lpaca*. 2023. URL: https://github.com/tatsu-lab/stanford_alpaca.

[8] Yurio Windiatmoko, Ridho Rahmadi, and Ahmad Fathan Hidayatullah. "Developing Facebook Chatbot Based on Deep Learning Using RASA Framework for University Enquiries". In: *IOP Conference Series: Materials Science and Engineering* 1077.1 (Feb. 2021), p. 012060. URL: https://dx.doi.org/10.1088/1757-899X/1077/1/012060.

[9] Junhao Zhanget al. Yaowei Zheng Richong Zhang. *LLAMAFACTORY: Unified Efficient Fine-Tuning of 100+ Language Models*. 2024. URL: https://arxiv.org/abs/2403.13372.

[10] Jindong Wang et al. Yupeng Chang Xu Wang. *A Survey on Evaluation of Large Language Models*. 2023. URL: https://arxiv.org/abs/2307.03109.

[11] Pat J. Zabinski, Ben R. Buhrow, Barry K. Gilbert, and Erik S. Daniel. "Applications of Optimization Routines in Signal Integrity Analysis". In: *DesignCon 2007*. Mayo Clinic. City, Country, 2007.

[12] Zuken Inc. *ECADSTAR*. Software available from Zuken Inc. 2023. URL: https://www.ecadstar.com/.