

**Securing Data Center with Digital Twins**

By

Foo Kar Yeng

A REPORT

SUBMITTED TO

Universiti Tunku Abdul Rahman

in partial fulfillment of the requirements

for the degree of

**BACHELOR OF INFORMATION TECHNOLOGY (HONOURS) COMMUNICATIONS  
AND NETWORKING**

Faculty of Information and Communication Technology  
(Kampar Campus)

JUNE 2025

## **ACKNOWLEDGEMENTS**

I would like to express my sincere gratitude to my supervisor, Dr. Aun Yichiet, for his invaluable guidance, support, and encouragement throughout the course of this project. His insights and feedback have been instrumental in shaping my understanding and approach to this work.

I would also like to thank my friends for their constant support, helpful discussions, and motivation during challenging times. Finally, my heartfelt appreciation goes to my family for their unconditional love, patience, and continuous encouragement throughout my academic journey.

# COPYRIGHT STATEMENT

© 2025 Foo Kar Yeng. All rights reserved.

This Final Year Project proposal is submitted in partial fulfillment of the requirements for the degree of **Bachelor of Information Technology (Honours) Communications and Networking** at Universiti Tunku Abdul Rahman (UTAR). This Final Year Project proposal represents the work of the author, except where due acknowledgment has been made in the text. No part of this Final Year Project proposal may be reproduced, stored, or transmitted in any form or by any means, whether electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of the author or UTAR, in accordance with UTAR's Intellectual Property Policy.

# ABSTRACT

This development-based project proposes an innovative approach to data center network planning by implementing Digital Twin technology to automate and accelerate network topology generation. Traditional network modelling tools, such as GNS3, rely on manual drag-and-drop configurations, which are inefficient and time-consuming for large-scale deployments. To address this limitation, this project introduces an AI-driven network diagram generator that leverages the OpenAI API to interpret natural language commands and dynamically construct network diagrams. Users can intuitively create a complete network, add, modify, or remove network components through text-based inputs, significantly reducing design time and minimizing human error. The system supports an export function that allows users to save generated network diagrams in both .png format for documentation and .json format for direct integration with GNS3. This enables seamless transfer of AI-generated topologies into a real emulation environment, allowing engineers to validate and refine their designs without rebuilding them manually. Compared to conventional methods, the proposed system provides a much faster and more efficient way to generate complex network topologies. The project follows a structured development methodology, including requirements analysis, prototype design, API integration, and iterative testing. Expected outcomes include a user-friendly, scalable tool that enhances network planning speed, efficiency, and compatibility with industry-standard platforms. By bridging the gap between AI automation and network emulation tools, this solution provides IT professionals with a smarter and more productive approach to data center infrastructure modelling.

Area of Study: Networking, Network Automation

Keywords: Network Automation, OpenAI API, Network Topology Generation, AI-Assisted Design, Network Visualization, GNS3 Integration, Large Language Models

# TABLE OF CONTENT

<b>TITLE PAGE.....</b>	<b>I</b>
<b>ACKNOWLEDGEMENTS .....</b>	<b>II</b>
<b>COPYRIGHT STATEMENT.....</b>	<b>III</b>
<b>ABSTRACT .....</b>	<b>IV</b>
<b>TABLE OF CONTENT .....</b>	<b>V</b>
<b>LIST OF FIGURES.....</b>	<b>IX</b>
<b>LIST OF TABLES.....</b>	<b>XI</b>
<b>LIST OF ABBREVIATIONS .....</b>	<b>XII</b>
<b>CHAPTER 1 .....</b>	<b>1</b>
1.1            Problem Statement and Motivation .....	1
1.1.1    Problem statement .....	1
1.1.2    Motivation.....	2
1.2 Research Objectives .....	3
1.3 Project Scope and Direction .....	4
1.4 Contributions .....	5
1.5 Report Organization .....	6
1.6 Background Information .....	7
<b>CHAPTER 2 .....</b>	<b>9</b>
2.1            GNS3: Virtualization Solution for Computer Network Education .....	9
2.1.1    Strengths and Weaknesses .....	10
2.2 Web-based OERs in computer networks .....	11
2.2.1    Strengths and Weaknesses .....	12
2.3 NLP-Powered Intent-Based Network Management .....	13
2.3.1    Strengths and Weaknesses .....	14
2.4 Development of OpenAI API-Based Chatbot to Improve User Interaction on the JBMS Website .....	15
2.4.1    Strengths and Weaknesses .....	16
2.5 AI-Assisted Network-Slicing Based Next-Generation Wireless Networks .....	17
2.5.1    Strengths and Weaknesses .....	18
2.6 Framework for Industrial Control System Honeypot Network Traffic Generation	19
2.6.1    Strengths and weaknesses .....	20

2.7 Demonstration of a Standalone, Descriptive, and Predictive Digital Twin of a Floating Offshore Wind Turbine .....	21
2.7.1 Strengths and weaknesses .....	22
2.8 Automation of Software Development Stages with the OpenAI API .....	23
2.8.1 Strengths and weaknesses .....	24
2.9 Gradio .....	25
2.9.1 Strengths and weaknesses .....	26
2.10 Sparks of Arti_cial General Intelligence: Early experiments with GPT -4.....	27
2.10.1 Strengths and weaknesses .....	28
<b>CHAPTER 3 .....</b>	<b>29</b>
3.1 System Design Diagram .....	29
3.1.1 System Architecture Diagram.....	29
3.1.2 Use Case Diagram and Description.....	32
3.1.3 Activity Diagram .....	35
<b>CHAPTER 4 .....</b>	<b>37</b>
4.1 System Block Diagram .....	37
4.2 System Components Specifications .....	39
4.2.1 Programming Language .....	40
4.2.2 LLM Engine .....	40
4.2.3 Graph Manager .....	40
4.2.4 Visualization Module .....	40
4.2.5 User Interface.....	40
4.2.6 Export Module .....	41
4.2.7 Emulation Platform .....	41
4.2.8 Development Environment .....	41
<b>CHAPTER 5 .....</b>	<b>42</b>
5.1 Hardware Setup .....	42
5.2 Software .....	43
5.2.1 VSCode .....	43
5.2.2 GNS3 .....	43
5.2.3 OpenAI .....	44
5.2.4 json .....	44
5.2.5 networkx.....	44

5.2.6 numpy .....	44
5.2.7 matplotlib (pyplot, patches, colors, offsetbox) .....	44
5.2.8 gradio .....	45
5.2.9 io / BytesIO .....	45
5.2.10 re .....	45
5.2.11 random .....	45
5.2.12 time .....	45
5.2.13 os .....	45
5.3 Setting and Configuration .....	46
5.3.1 Development Environment Setup .....	46
5.3.2 Library and Dependency Installation .....	46
5.3.3 OpenAI API Configuration .....	46
5.3.4 System Configuration .....	47
5.3.5 GNS3 Integration Setup .....	47
5.3.6 Testing Configuration .....	48
5.4 System Operation .....	49
5.4.1 Launching the System .....	49
5.4.2 Entering a Command .....	49
5.4.3 LLM Processing and JSON Generation .....	49
5.4.4 Network Diagram Visualization .....	50
5.4.5 Modifying the Topology .....	50
5.4.6 Exporting Topology in PNG .....	51
5.4.7 Exporting Topology in JSON .....	51
5.4.8 Opening JSON in GNS3 .....	52
5.5 Implementation Issues and Challenges .....	53
5.6 Concluding Remark .....	54
<b>CHAPTER 6 .....</b>	<b>55</b>
6.1 System Testing and Performance Metrics .....	55
6.2 Testing Setup and Result .....	58
6.3 Project Challenges .....	59
6.4 SWOT Analysis .....	61
6.5 Objectives Evaluation .....	62
6.6 Concluding Remark .....	63

<b>CHAPTER 7 .....</b>	<b>64</b>
7.1 Conclusion .....	64
7.2 Recommendation.....	65
<b>REFERENCES.....</b>	<b>66</b>
<b>APPENDIX.....</b>	<b>A-1</b>



# LIST OF FIGURES

Figure Number	Title	Page
Figure 2.1	Virtualized Network Computer Topology with GNS3	9
Figure 2.2	Applet user interface for simulating data transmission in an Ethernet LAN	11
Figure 2.4	OpenAI Function	15
Figure 2.5	The RAN slicing in NGWNs	17
Figure 2.7	Digital Twin Framework for the Test Site	21
Figure 2.8	Internal process in the OpenAI API	23
Figure 3.1.1	System Architecture Diagram	31
Figure 3.1.2	Use Case Diagram	33
Figure 3.1.3	Activity Diagram	36
Figure 4.1	System Block Diagram	38
Figure 5.1	HP Laptop	42
Figure 5.2.1	VSCode	43
Figure 5.2.2	GNS3	43
Figure 5.3.2	Library Installation	46
Figure 5.3.3	OpenAI API Configuration	47
Figure 5.3.4	Visualization Settings	47
Figure 5.3.5.1	GNS3 VM Setup	48
Figure 5.3.5.2	GNS3 Setup	48
Figure 5.3.6	JSON to GNS3	49
Figure 5.4.1	Launching in VSCode	49
Figure 5.4.2	Enter natural language command into the systemAPI	50
Figure 5.4.3	JSON output generated by the LLM engine	50
Figure 5.4.4	Network topology visualization with icons and layers	51
Figure 5.4.5	Modified topology	51
Figure 5.4.6	Export confirmation for PNG	52
Figure 5.4.7	Export confirmation for JSON	52
Figure 5.4.8	Generated network topology displayed inside GNS3	53

Figure 6.1.1	JSON output generated from user command	55
Figure 6.1.2	Visualization of generated topology using Matplotlib	56
Figure 6.1.3	Export confirmation for PNG and JSON files	56
Figure 6.1.4	Exported JSON successfully imported into GNS3	57
Figure 6.1.5	Response time measurement for command processing	57

## LIST OF TABLES

<b>Table Number</b>	<b>Title</b>	<b>Page</b>
Table 3.1	Specifications of laptop	29
Table 3.1.2	Use Case Table	34
Table 4.2	Table of Components	39
Table 6.2	Testing Setup and Result Matrix	59
Table 6.4	SWOT Table	61
Table 6.5	Objectives Evaluation	62

## LIST OF ABBREVIATIONS

<i>API</i>	Application Programming Interface
<i>GNS3</i>	Graphical Network Simulator-3
<i>AI</i>	Artificial intelligence
<i>PNG</i>	Portable Network Graphics
<i>SSH</i>	Secure Shell Protocol
<i>SDN</i>	Software-Defined Networking
<i>NLP</i>	Natural Language Processing
<i>DNTG</i>	Distributed Network Traffic Generator
<i>TCP</i>	Transmission Control Protocol
<i>IP</i>	Internet Protocol
<i>CPU</i>	Central Processing Unit
<i>OER</i>	Open Educational Resources
<i>OCW</i>	OpenCourseWare
<i>LAN</i>	Local Area Network
<i>NGWNs</i>	Next-generation Wireless Networks
<i>NFV</i>	Network Functions Virtualization
<i>RAT</i>	Remote Access Trojans
<i>OT</i>	Operational Technology
<i>ICS</i>	Industrial Control Systems
<i>SDK</i>	Software Development Kit

# CHAPTER 1

## Introduction

### 1.1 Problem Statement and Motivation

#### 1.1.1 Problem statement

Network topology design in modern data centers demands scalable and efficient solutions, yet manual, drag-and-drop interfaces still limit the use of conventional tools like Cisco Packet Tracer and GNS3. Engineers often need to rebuild entire network designs for small changes or different scenarios, making the process time-consuming and inflexible. These approaches not only slow down deployment timelines but also increase the risk of human error, especially in dynamic or large-scale settings.

Although GNS3 is widely used for emulation, its manual nature creates bottlenecks during the early design phase. Engineers spend valuable time arranging devices and connections instead of focusing on optimization or validation. As a result, the lack of intelligent automation reduces productivity and delays testing workflows.

This project addresses these challenges by introducing an AI-powered network topology generator that leverages Large Language Models (LLMs) via the OpenAI API to interpret natural language commands. Users can simply describe their requirements in plain text, and the system automatically generates a complete network design with routers, switches, firewalls, and servers. To further enhance usability, the system supports exporting diagrams in both .png for documentation and .json for seamless import into GNS3, eliminating the need for manual reconstruction. By combining LLM-driven automation with direct GNS3 integration, the proposed solution enables network topologies to be generated, modified, and tested much faster than traditional methods, while significantly reducing human effort and errors.

### 1.1.2 Motivation

The complexity of data center networks continues to grow, requiring more intelligent, rapid, and flexible design tools. Existing solutions like GNS3 and Cisco Packet Tracer, while powerful for emulation and education, still rely on manual drag-and-drop design processes. Engineers spend significant time on repetitive tasks such as device placement, naming, and linking, which slows down innovation and delays testing. This inefficiency prevents network professionals from focusing on higher-value activities like optimization, scaling, or validation.

Recent advancements in Large Language Models (LLMs) provide an opportunity to transform this workflow. By enabling natural language inputs, engineers can describe network requirements in plain text, and the system can automatically translate these descriptions into structured topologies. This approach not only saves time but also lowers the technical barrier for users with less experience in manual topology design.

A further motivation is the need for seamless integration with industry-standard tools. While diagram visualization is useful, network engineers often require a direct path to emulation and testing. By enabling JSON export for GNS3, this project ensures that AI-generated topologies can be instantly deployed into a real emulation environment without reconstruction. This greatly accelerates the design-to-testing pipeline.

### 1.2 Research Objectives

The objective of this project is to develop a Digital Twin system powered by Large Language Models (LLMs) to automate network topology design in data center environments. The system aims to leverage natural language processing through the OpenAI API to drastically reduce the time and effort needed compared to traditional manual methods. The solution will enable users to generate and modify topologies consisting of routers, switches, firewalls, servers, and other devices using plain text commands. To enhance usability, the project also provides export capabilities in both .png format for visualization and documentation, as well as .json format for direct compatibility with GNS3. This ensures that AI-generated topologies can be seamlessly deployed in an emulation environment without requiring manual reconstruction.

The research seeks to achieve three main outcomes. First, it aims for at least 90% accuracy in converting natural language descriptions into valid topologies across a broad range of network device types. Secondly, it illustrates that automation led by LLM can decrease topology design time by over 80% relative to human development in GNS3 or Cisco Packet Tracer. Thirdly, it ensures interoperability and reduces the amount of human configuration overhead by verifying the smooth integration of produced JSON files with GNS3. The goal of the project is to expedite the design and visualization phase of network planning; low-level protocol analysis, physical hardware implementation, and real-time traffic monitoring are not included. Furthermore, it establishes the foundation for future developments such as Software-Defined Networking (SDN) integration or collaborative multi-user design features.

### 1.3 Project Scope and Direction

In order to automate network topology generation for data center design, this project will create a software-based solution. A working prototype that uses OpenAI's natural language processing powers to comprehend user input and instantly produce matching network diagrams will be the deliverable. Three main functions will be the focus of the system: (1) turning text inputs like "create a three-tier architecture with dual firewalls" into precise visual representations of networks; (2) allowing dynamic changes to pre-existing diagrams by adding more text commands; and (3) offering export functionality to store diagrams in common image formats for documentation. The scope clearly covers the creation of an AI-driven design tool and its fundamental features, but it expressly leaves out the installation of a physical network, real-time traffic monitoring, and sophisticated security features like honeypot integration, which are reserved for later versions. While keeping an architecture that can be expanded for more complicated functionality in later phases of development, the method seeks to expedite the initial network planning step.



### 1.4 Contributions

I have significantly advanced the field of AI-driven network automation with this effort. First, I used the OpenAI API to successfully create an LLM-powered network topology generator, proving that natural language instructions may successfully take the place of more conventional human design techniques. Compared to traditional drag-and-drop tools like GNS3 and Cisco Packet Tracer, my system greatly speeds up the design process while achieving high accuracy in translating text-based inputs into appropriate network designs. This demonstrates how topology construction may be accelerated and made more accessible with AI-driven automation.

Second, I created and put into use an export framework that enables the saving of generated topologies in both .png and .json formats. While the JSON export allows for smooth integration with GNS3, the PNG export facilitates documentation and visualization. This contribution minimizes repeated work and closes the gap between AI-based design and real-world implementation by guaranteeing that topologies produced by my system may be imported straight into an emulation environment without the need for human reconstruction.

Lastly, I created an extensible and modular design that allows for future improvements while preserving compatibility with current network tools. In addition to facilitating dynamic alterations, such adding or removing nodes using normal language, my system guarantees logical integrity while making changes to intricate topologies. This illustrates that even when undergoing iterative modifications, AI-generated designs may preserve structural integrity. Overall, my work offers a longer-term basis for investigating the function of LLMs in intelligent and automated network architecture modelling, as well as an immediate benefit as a time-saving design tool.

### 1.5 Report Organization

This report presents the creation, deployment, and assessment of an AI-driven network topology generator that is directly integrated into GNS3 and is divided into seven structured chapters.

In Chapter 1, the project background is presented, along with the problem definition, motivation, goals, and scope. The report's structure is summarized, and the contributions that have been made are highlighted.

Chapter 2 provides a comprehensive literature review, discussing existing approaches in network topology design, applications of natural language processing (NLP) in network automation, and current practices in simulation and emulation tools such as GNS3. This chapter establishes the theoretical foundation and identifies the research gaps addressed by this project.

Chapter 3 outlines the proposed methodology and system model. The system architecture, use case diagram, activity diagram, and workflow design are all explained, along with how natural language commands are converted into structured JSON, shown as diagrams, and exported for usage by other parties.

Chapter 4 covers the system design, which includes the system block, component specifications, module interactions, and operational workflow. It demonstrates how the system accomplishes modularity and guarantees seamless cross-layer integration.

Chapter 5 discusses the system implementation, including hardware and software setup, configuration, system operations, and problems. To demonstrate the process and verify functionality, screenshots and samples are given.

Chapter 6 is dedicated to the evaluation and discussion of results. System testing, performance metrics, result analysis, project challenges, a SWOT analysis, and an evaluation of the degree to which the project's goals were met are all included.

Chapter 7 ends up the report by summarizing the project results and reiterating the contributions. To enhance system scalability, visualization quality, and integration with other simulation tools, it also offers suggestions and areas for future research.

By providing a clear and organized presentation of the project, this structure guarantees that the report proceeds logically from the identification of the issue to the application and assessment of the solution.

### 1.6 Background Information

The combination of artificial intelligence (AI), network automation, and cybersecurity has resulted in significant changes to how digital infrastructure is created, monitored, and secured. Intent-based, intelligent, and dynamic designs are replacing traditional network management systems, which are frequently static and manually set. This shift is particularly critical in complex and high-value environments like data centers, where operational continuity, performance monitoring, and security must coexist within scalable and adaptive frameworks. In such contexts, digital twin technology—a real-time virtual replica of physical systems—emerges as a powerful paradigm for simulating, visualizing, and safeguarding critical network infrastructure [7].

The concept of a digital twin originated in the field of manufacturing and product lifecycle management but has since evolved into broader domains such as smart cities, industrial systems, and now computer networking. A digital twin in networking represents the virtual replication of physical network components (routers, switches, firewalls, etc.) along with their interconnections and behavior under real-world conditions. By integrating artificial intelligence, simulation models, and sensor data, digital twins enable predictive maintenance, anomaly detection, and configuration optimization in real time [7], [10]. When combined with AI-powered interfaces such as the OpenAI API or DeepSeek:R1, digital twins can be controlled via natural language interfaces, simplifying human-machine interaction while increasing accessibility for system operators and engineers [5], [8].

The importance of cybersecurity, particularly deception-based strategies like honeypots, which are intended to deceive attackers and gather information on intrusion techniques, is equally significant in this digital ecosystem. When honeypots replicate realistic traffic and network topologies, their efficacy is greatly increased in industrial and data center settings [6]. To do this, decoy systems can create real industrial traffic using frameworks like the Distributed Network Traffic Generator (DNTG), which makes it harder for attackers to tell them apart from legitimate systems. There are two benefits to integrating such deception methods into digital twins: proactive threat intelligence and operational monitoring.

Furthermore, the adoption of tools such as Gradio [9] for visualization and GNS3 [1], [2] for emulated topology deployment demonstrates a strong movement toward modular and AI-integrated network design solutions.

In recent years, AI has emerged as a key driver of innovation in next-generation networking systems. Researchers have explored the integration of natural language processing (NLP) into intent-based networking to manage complex environments like private 5G systems, illustrating how commands can be automatically translated into network policies [3]. Similarly, AI's role in developing self-organizing, self-optimizing, and self-defending networks has been emphasized in modern network architecture literature [4]. These advancements directly inform this project's aim: to design and implement a system that not only allows users to generate and manipulate network diagrams using AI-based command interfaces, but also to integrate honeypot systems that simulate realistic attack surfaces for cyber defense evaluation.

In order to meet the growing need for intelligent and safe data center infrastructures, this project expands upon several disciplines, including AI, networking, cybersecurity, and user interface design. An multidisciplinary approach in line with contemporary technology trends is demonstrated by the use of OpenAI for command interpretation, NetworkX and Matplotlib for topology display, Gradio for interaction, and a honeypot simulation system.

## CHAPTER 2

### Literature Reviews

#### 2.1 GNS3: Virtualization Solution for Computer Network Education

Gil et al. (2014) presented a novel method for teaching computer networks that uses the virtualization platform GNS3 (Graphical Network Simulator) to mimic actual network settings. The constraints of conventional teaching techniques, which depended on hardware for networking and simulators that used programming, such as NS-2/3 and J-SIM, were addressed by this solution. The researchers used GNS3's interaction with Dynamips for Cisco router emulation and VirtualBox/Qemu for host simulation to create a virtual network topology that mirrored the physical lab configuration at their university. Despite Idlepc optimizations, this method consumed a large amount of hardware resources and had trouble simulating some network protocols, such as WiFi (802.11) and Token Ring, even though teachers reported that it correctly replicated roughly 90% of genuine network functionality.

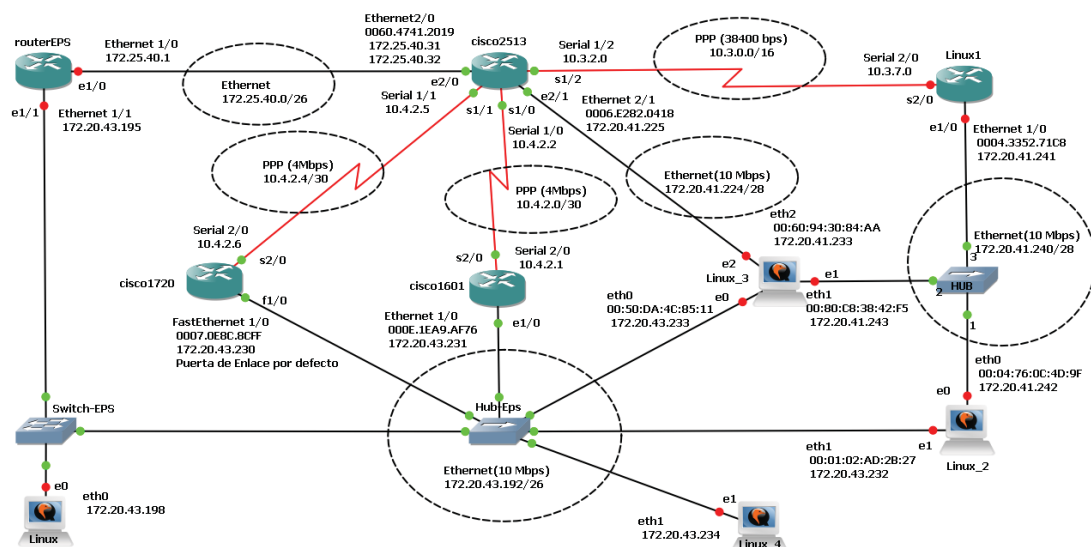


Figure 2.1 Virtualized Network Computer Topology with GNS3

### 2.1.1 Strengths and Weaknesses

Gil et al. [1] provide a fair review of GNS3's capabilities based on empirical deployment in computer network education. The platform's capacity to reproduce roughly 90% of actual network functionality without incurring physical hardware costs has been noted as its main strength. This was accomplished by leveraging Dynamips to precisely emulate Cisco IOS routers and VirtualBox/Qemu integration to enable thorough TCP/IP protocol analysis that is on par with actual laboratories. The pedagogical usefulness of GNS3 in aiding distant learning through remote access and its integration with Wireshark for hands-on packet analysis exercises are specifically highlighted by the authors [1].

However, the journal publication specifically states three major limitations. First, gaps in protocol support were found, particularly the inability to simulate Token Ring and WiFi (802.11) technologies because of virtualization limitations. Second, even with Idlepc improvements, CPU consumption remained high, indicating large hardware requirements. Third, usability issues were found in student surveys, which gave the interface's intuitiveness and learning curve an average rating of 3.2/5 [1]. Interestingly, the lack of integrated security testing tools is acknowledged in the study as a significant barrier to thorough network training, but it is not investigated.

## 2.2 Web-based OERs in computer networks

The study “Web-based OERs in Computer Networks” examines the efficacy of Open Educational Resources (OERs) in teaching computer networks at the University of Alicante in Spain, utilizing a blended-learning method [2]. Francisco A. Candelas, Carlos A. Jara, Gabriel J. García, and Fernando Torres co-authored the study, which was led by Pablo Gil and focused on incorporating digital tools like Moodle, OpenCourseWare (OCW), blogs, video logs, and interactive simulations (like KivaNS and EJS-based applets) into engineering education. Using a mixed-methods approach, the authors carried out a longitudinal study that lasted five academic years (2007–2012) and integrated quantitative statistical analyses, such as linear regression and Pearson’s and Spearman’s correlations, with qualitative student input [2]. The results demonstrated that combining multiple OERs significantly enhances learning outcomes compared to using a single resource, providing valuable insights for educators in technical disciplines [2].

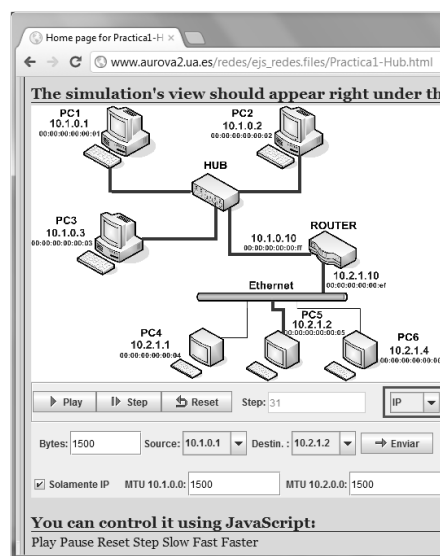


Figure 2.2 Applet user interface for simulating data transmission in an Ethernet LAN

### 2.2.1 Strengths and Weaknesses

The study has various strengths, including a robust longitudinal design that monitors student performance and participation across five academic years [2]. A comprehensive picture of how blended learning improves technical education is offered by the use of several OERs, including OCW, blogs, and interactive simulations. Additionally, statistical analyses (such as Pearson's and Spearman's correlations) support the findings by showing a positive correlation between OER usage and higher grades [2]. With its well-organized sections on approach, resource development, and results, the paper is easy for readers to understand. The study does have some drawbacks, though. The results' generalizability may be impacted by the absence of comprehensive participant demographic data, and the use of self-reported survey data raises the possibility of bias [2]. Furthermore, even while students responded well to video logs, statistical analysis showed that using them alone did not significantly increase performance, indicating the need for more research. Lastly, the findings might be strengthened by replication in various educational contexts, as the concentration on a specific institution and subject limits their wider applicability [2].



### 2.3 NLP-Powered Intent-Based Network Management

The journal article “NLP Powered Intent Based Network Management for Private 5G Networks” [3] introduces an innovative approach to simplifying network operations through Natural Language Processing (NLP) and intent-based networking principles. The research addresses the growing complexity of managing private 5G networks, particularly for operators lacking specialized technical expertise. The authors propose a novel architecture called the intelligence stratum, which comprises two key components: an Intent Engine that processes high-level user commands and an AI Engine that hosts machine learning models to translate these intents into actionable network configurations[3]. This system enables users to manage network resources through simple natural language inputs, such as requesting a new network slice, without needing to understand the underlying technical details.

The study validates the approach through three practical use cases: network slice provisioning, indoor positioning services, and network service deployment. Each scenario demonstrates how the NLP interface reduces operational complexity while maintaining efficient performance, with measurable benchmarks such as slice provisioning times averaging approximately three minutes. By integrating machine learning models with network orchestration, the research showcases a significant step toward autonomous network management. The findings align with broader industry trends toward AI-driven automation in telecommunications, providing a foundation for future work on intent-based systems in next-generation networks[3].

### 2.3.1 Strengths and Weaknesses

The study's practical significance in network management is attributed to a number of noteworthy strengths [3]. The effective usage of an NLP-based interface, which considerably reduces the entrance barrier for novice users overseeing private 5G networks, is one of its main benefits. The AI Engine's modular architecture makes it possible to integrate specialized machine learning models in a flexible way, which helps the system effectively handle a variety of use cases. The operational viability of the system is further demonstrated by the authors' empirical validation using real-world benchmarks, such as timing the slice provisioning process. Scalability is ensured by using OpenFaaS for AI model deployment, and a strong and flexible automation framework is highlighted by the dynamic API-driven interactions between the Intent Engine and network functions.

However, the proposed system has drawbacks [3]. Its reliance on preset machine learning models is a major disadvantage, as these models may find it difficult to interpret intents outside of their training scope, which could impede adaptability in dynamic contexts. Furthermore, the assessment is carried out in controlled test environments, raising concerns about the system's potential performance in expansive, diverse networks with erratic traffic patterns. The intrinsic ambiguity of natural language inputs presents another difficulty, as it may result in incorrect configurations if the system is unable to correctly decipher user requests. Last but not least, the AI models might not be able to swiftly adjust to changing network conditions due to the absence of real-time learning processes, indicating the need for more study into continuous model training and optimization.

## 2.4 Development of OpenAI API-Based Chatbot to Improve User Interaction on the JBMS Website

The research by Santoso et al. [4] presents a significant advancement in academic platform interfaces through the development of an AI-powered chatbot integrated with OpenAI's API. This innovative solution addresses critical limitations in traditional journal websites by offering three key functionalities: intelligent article recommendations based on user queries, step-by-step guidance for manuscript submission processes, and automated summarization of journal content. The system architecture combines machine learning algorithms with natural language processing capabilities to enable more natural and context-aware interactions compared to conventional rule-based chatbots.

Multiple iterations of requirement analysis, system design, and user evaluation were all part of the development process, which adhered to a strict prototyping methodology. During the testing stages, important stakeholders such as academic researchers, journal editors, and students took part and offered insightful comments that influenced the final implementation. Strong user acceptance was shown by performance measures, with accuracy and relevancy of responses receiving especially high ratings. A noteworthy technical accomplishment is the chatbot's capacity to decipher intricate academic jargon and extract important information from research papers.

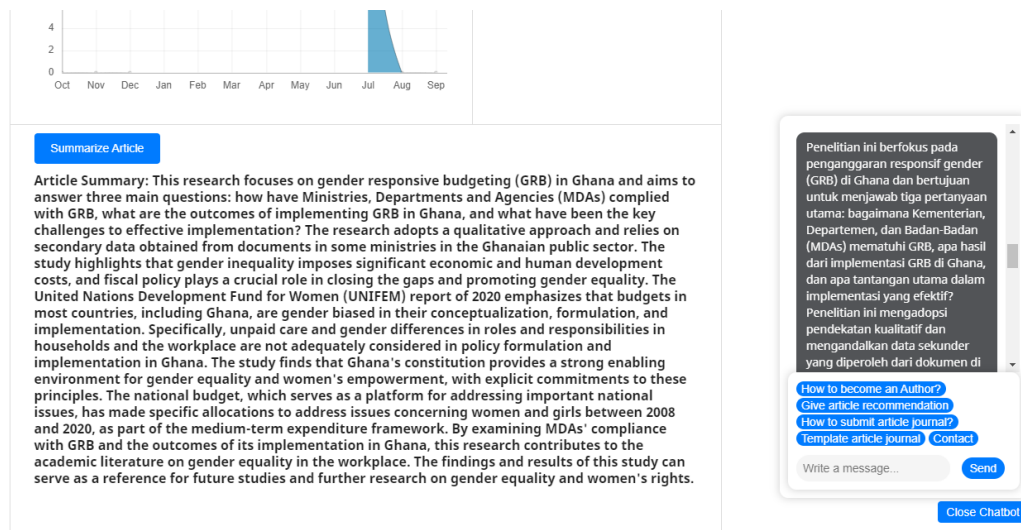


Figure 2.4 OpenAI Function

### 2.4.1 Strengths and Weaknesses

The concept and execution of the OpenAI API-based chatbot created by Santoso et al. [4] exhibit a number of noteworthy advantages. The system's great functionality is the most important of these, as shown by its high User Acceptance Testing score of 4.14/5, which shows that users are very satisfied with features like submission advice and article recommendations [4]. Iterative refinement based on direct feedback from several stakeholder groups, including students, lecturers, and the CEO of JBMS, was made possible by the prototyping process, which proved very effective [1]. An important technical accomplishment is the incorporation of OpenAI's sophisticated natural language processing capabilities, which allow the chatbot to carry out challenging tasks like contextual question responding and article summary that would be difficult for conventional rule-based systems to handle. Additionally, the study fills a significant vacuum up in the literature by offering useful empirical data on chatbot installation in an academic publication context, as noted by the authors [4].

Nevertheless, the study also identifies a number of limitations that should be taken into account. For some journal systems with tight budgets, the \$20 monthly maintenance fee for the OpenAI API service can be a deterrent to adoption [4]. Participants gave the user interface design a neutral rating of 3.56/5, indicating that there is potential for development in both visual presentation and interaction design. More training data and more advanced language models are required, as the study also found limitations in the system's natural language processing abilities, specifically in managing a wide range of vocabulary and phrasing changes [4]. Furthermore, even though the prototype process worked well for development, the study ignores potential scaling issues or long-term maintenance problems that could occur with higher user loads or more extensive functionality requirements. These restrictions offer crucial factors to take into account for upcoming studies and advancements in this field.

## 2.5 AI-Assisted Network-Slicing Based Next-Generation Wireless Networks

The study by Shen et al. [5] presents a comprehensive framework for AI-assisted network slicing in next-generation wireless networks (NGWNs). The authors develop a two-tier architecture combining SDN/NFV technologies with AI/ML techniques to address three critical challenges: RAN slicing, automated RAT selection, and mobile edge caching. For RAN slicing, the work demonstrates how reinforcement learning can optimize resource allocation across different service types while reducing operational costs. The automated RAT selection approach employs hybrid control schemes using machine learning to manage user mobility in heterogeneous networks. The edge caching solution utilizes deep learning models for predictive content placement. While providing a robust theoretical foundation, the study primarily relies on simulation-based validation rather than real-world deployment results.

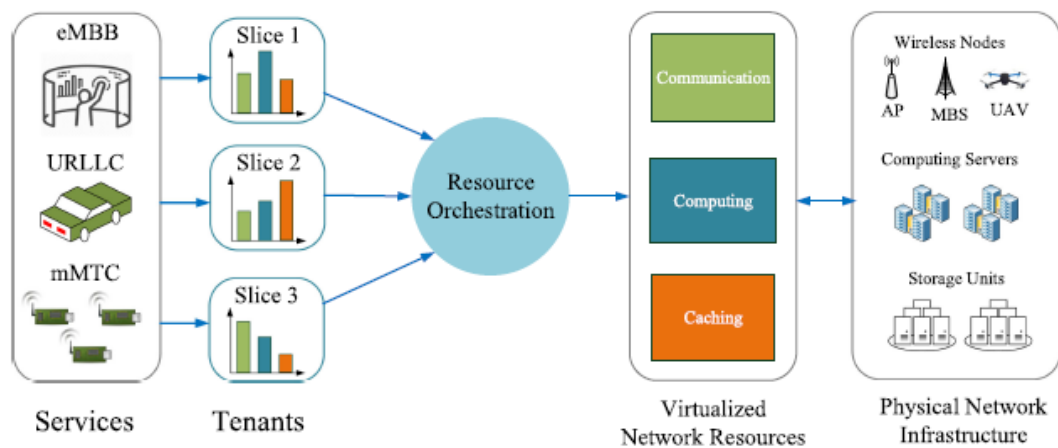


Figure 2.5 The RAN slicing in NGWNs

### 2.5.1 Strengths and Weaknesses

The research offers several significant contributions to the field. Its integration of advanced AI techniques, including deep reinforcement learning and Markov decision processes, provides innovative solutions for complex network management problems. The framework's ability to jointly optimize multiple resource types (communication, computing, caching) represents a notable advancement in handling NGWN heterogeneity. The study also demonstrates substantial cost reduction potential through dynamic network slicing.

However, some limitations should be considered. The dependence on paid APIs may hinder practical implementation in budget-constrained scenarios. The user interface design received moderate evaluation scores, indicating room for improvement in usability aspects [5]. Additionally, the AI models require extensive training data that may not be readily available for emerging applications. Compared to existing approaches, this work advances AI-based network management but would benefit from more extensive field testing and energy efficiency analysis. Future research could address these aspects while building on the framework's strengths in multi-dimensional resource optimization.

### **2.6 Framework for Industrial Control System Honeypot Network Traffic Generation**

The thesis “Framework for Industrial Control System Honeypot Network Traffic Generation” presents a novel method to improve the realism and effectiveness of honeypots in Operational Technology (OT) environments. The study emphasizes that traditional honeypots, while useful for intrusion detection and threat analysis, often lack the necessary network behavior to convincingly mimic Industrial Control Systems (ICS). Without authentic traffic, attackers capable of passive monitoring can easily distinguish honeypots from real systems, thus diminishing their value as decoys. To address this, the research introduces a proof-of-concept framework that integrates a Distributed Network Traffic Generator (DNTG) with honeypot platforms to emulate legitimate control system network activity. Using network traces from a Siemens APOGEE automation system, the DNTG modifies packet headers to match honeypot characteristics and replays the traffic across the decoy network. This approach allows honeypots to send and receive realistic traffic, simulating operations of authentic ICS environments. The research concludes that this method significantly improves honeypot believability by maintaining traffic content, packet order, and source authenticity, thereby strengthening the defense-in-depth strategy of critical infrastructure networks [6].

### 2.6.1 Strengths and weaknesses

Lamb's approach addresses a fundamental flaw in conventional honeypots—their inability to replicate genuine network behavior—and represents a substantial breakthrough in the field of cybersecurity for industrial control systems (ICS). In order to create realistic traffic in honeypot environments, the suggested framework presents a Distributed Network Traffic Generator (DNTG) that uses real network traces from a Siemens APOGEE automation system. This feature imitates real-time communication patterns found in operational technology (OT) environments, significantly increasing the efficacy of deception tactics. This method's usage of legitimate ICS communication profiles is one of its main advantages since it increases the authenticity of the emulated environment and lessens the possibility that passive attackers will discover the honeypot. Additionally, the DNTG's modular design facilitates easy interaction with current honeypot systems, possibly broadening its use across various ICS configurations.

Nevertheless, there are some restrictions on the structure. It might not accurately represent dynamic or adaptive communication patterns present in contemporary, event-driven control networks due to its reliance on replaying collected traffic. The framework's primary focus on Modbus/TCP-based systems may also limit its applicability to other ICS protocols, such as DNP3 or IEC 104. The lack of comprehensive performance evaluation or scalability testing in large industrial settings further limits the generalizability of the findings. The approach makes a significant contribution to improving honeypot believability in OT systems in spite of these limitations.



## 2.7 Demonstration of a Standalone, Descriptive, and Predictive Digital Twin of a Floating Offshore Wind Turbine

Stadtman, Wassertheurer, and Rasheed present a detailed implementation of a digital twin (DT) system applied to a floating offshore wind turbine, showcasing its potential to enhance monitoring, forecasting, and decision-making in complex marine environments. In their work, a DT is described as a dynamic virtual representation of a physical asset, capable of integrating real-time data, simulations, and machine learning models to provide actionable insights. The authors introduce a six-level digital twin capability framework, demonstrating levels 0 to 3—namely, standalone, descriptive, and predictive digital twins. Their case study on the Zephyros floating wind turbine includes a CAD-based 3D model for visualization, live sensor integration for operational status updates, and deep learning models (DNN and LSTM) for forecasting wind and system responses. This multi-level implementation illustrates the advantages of hybrid analytical methods and real-time integration in advancing remote asset diagnostics and predictive maintenance. The study further outlines future directions for prescriptive and autonomous twins, emphasizing scalability and adaptability across various infrastructure systems [7].

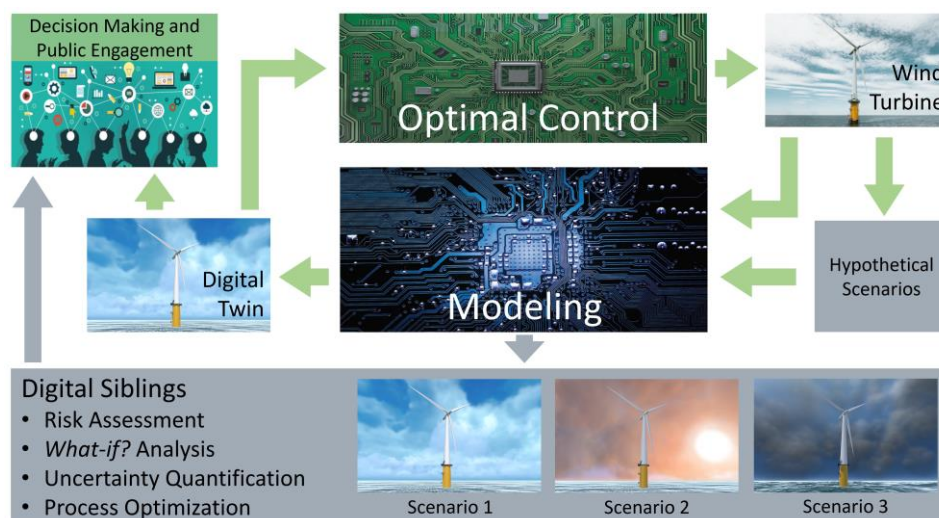


Figure 2.7 Digital Twin Framework for the Test Site

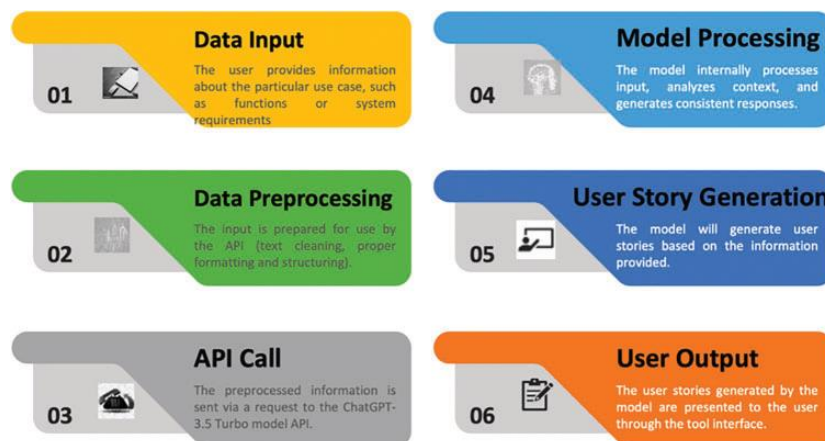
### 2.7.1 Strengths and weaknesses

An organized six-level model of digital twin capabilities is presented in the study. Using a floating offshore wind turbine as the test case, it offers a demonstrative solution that covers the first three levels: standalone, descriptive, and predictive. Predictive maintenance and defect detection are made possible by the system's ability to predict operational outcomes through the use of machine learning models like deep neural networks (DNN) and long short-term memory (LSTM) for real-time forecasting.

The study's careful fusion of sensor data collection, CAD-based visualization, and hybrid <sup>22</sup>network<sup>22</sup>g approaches—all of which work together to create a thorough and expandable digital twin framework—is one of its strongest points. A clear path for future growth into prescriptive and autonomous systems is provided by this tiered approach. However, cybersecurity, intrusion detection, and interaction with deception systems like honeypots are not given as much attention in this research as performance prediction and monitoring. Furthermore, even if the system shows predictive power, there was no benchmarking done on how well the machine learning models performed in changing environmental conditions. Furthermore, in situations where such data infrastructure is lacking, the study's reliance on high-fidelity sensor input may present implementation issues.

## 2.8 Automation of Software Development Stages with the OpenAI API

Tapia and Gaona's study explores the practical impact of OpenAI's ChatGPT API on automating software development processes, particularly within agile methodologies. Their research introduces a semi-automated development framework capable of generating user stories, estimating task effort, and producing code for typical software applications such as appointment systems and inventory platforms. According to the study, including ChatGPT into the planning, coding, and testing stages of development greatly cuts down on overall development time by 40% to 51% when compared to manual approaches. But the authors also point out important drawbacks: Before being released, AI-generated content must be reviewed, debugged, and improved by human developers because it frequently lacks accuracy and completeness. The paper highlights that while ChatGPT accelerates the development workflow and supports tasks such as documentation and prototyping, it still demands oversight to ensure maintainability and correctness. Ultimately, the findings underscore that AI tools should be treated as productivity-enhancing supplements rather than full replacements for human developers in software engineering contexts [8].



**Figure 2.8** Internal process in the OpenAI API

### 2.8.1 Strengths and weaknesses

The work of Tapia and Gaona critically examines the practical implications of integrating OpenAI's ChatGPT API into software engineering workflows. Their proposed semi-automated development framework demonstrates the potential of language models to expedite key stages of agile development, such as requirements generation, task estimation, and code prototyping. A significant strength of this research is its empirical validation, showing a reduction in development time of up to 51%, particularly in low to medium complexity applications such as appointment and inventory systems. The study also emphasizes how well the paradigm works to streamline the development lifecycle by producing technical documentation and unit tests.

However, the study also highlights the intrinsic flaws in programming that uses AI. Human developers must manually evaluate and troubleshoot the generated outputs since they frequently lack syntactic precision, contextual awareness, and semantic accuracy. This restriction calls into question AI's dependability in high-stakes software settings where accuracy and security are crucial. Furthermore, the study ignores integration with version control, large-scale collaboration tools, and continuous integration/continuous deployment (CI/CD) pipelines—all essential elements of contemporary software engineering. Thus, even while the study shows encouraging increases in efficiency, it also emphasizes the necessity of exercising caution and maintaining human control when implementing AI in systems at the production level.

### 2.9 Gradio

A straightforward interface for integrating machine learning models and applications into approachable web frontends is provided by the Gradio Python client. Gradio, according to the official description, allows developers to rapidly prototype and deploy models with interactive features like text inputs, sliders, and image canvases, significantly reducing the time and complexity needed to create distinctive graphical user interfaces (GUIs). Gradio facilitates use cases ranging from data annotation and model explainability to real-time user input gathering by abstracting front-end logic into straightforward Python functions. Its compatibility with popular frameworks like PyTorch, TensorFlow, and Hugging Face Transformers further enhances its applicability across research and production environments. For projects like natural language-driven network diagram generators, Gradio offers an efficient and accessible solution to connect complex AI logic with non-technical users, enabling real-time visualization and interactivity without the overhead of traditional web development stacks [9].

### 2.9.1 Strengths and weaknesses

Gradio offers an easy-to-use and adaptable interface for quickly deploying machine learning applications with intuitive web frontends, according to its official literature. One of its primary benefits is its ability to abstract complex back-end logic and present it through intuitive visual interfaces, which makes machine learning methods easier for non-technical users to comprehend. Gradio supports a wide range of input forms, such as text, photos, and sliders, and integrates nicely with well-known frameworks like Hugging Face Transformers, PyTorch, and TensorFlow. Gradio makes it possible to quickly iterate and demonstrate AI models for research and development, which makes it ideal for applications such as explainable AI, user feedback loops, and educational settings.

However, its simplicity can also be a constraint in production-grade applications. The platform may not scale effectively in enterprise settings that require advanced user authentication, state management, or complex backend orchestration. In addition, customization is less flexible than with more conventional web frameworks like Flask or Django. Other issues include security and data privacy, especially when handling private or private information in hosted environments. Gradio's constraints must therefore be taken into account while moving toward full-scale deployment, even though it is ideally suited for proof-of-concept development.

### **2.10 Sparks of Artificial General Intelligence: Early experiments with GPT-4**

Bubeck et al. provide a thorough analysis of OpenAI's GPT-4 model, describing it as a major step forward in the development of artificial general intelligence (AGI) [10]. The GPT-4 has a surprising degree of generality, flexibility, and reliability, according to the authors' extensive studies done across a wide range of cognitive and practical tasks, including mathematics, software development, medicine, law, and even visual thinking. In tasks involving compositional reasoning, abstraction, and domain transfer, GPT-4 regularly outperforms baseline models, in contrast to previous iterations such as GPT-3.5 or the original ChatGPT. The ability of GPT-4 to use innovative reasoning chains to solve previously encountered issues is one of the paper's most remarkable discoveries, indicating a level of emergent intelligence absent from previous models.

The model, for example, can evaluate intricate legal texts, develop functional code, solve Olympiad-level math problems, and explain jokes. The scientists characterized GPT-4's behavior as displaying "sparks" of AGI as a result of these findings. The report does, however, issue a warning against overestimating its present capabilities. GPT-4 continues to have well-known flaws, including sensitivity to prompt language, incapacity to plan over extended contexts, and hallucinations. Despite these drawbacks, the authors contend that GPT-4 represents a significant change in the field of artificial intelligence research, one that calls for new standards and conceptual models to comprehend machine intelligence. In addition to highlighting the necessity of responsible deployment and governance, their findings highlight the model's potential for a wide range of real-world applications, such as industrial automation, scientific research, and education [10].

### 2.10.1 Strengths and weaknesses

The comprehensive evaluation of GPT-4 by Bubeck et al. explores the model's cognitive breadth and its potential as a precursor to artificial general intelligence (AGI). Their interdisciplinary analysis covers a wide array of domains, from mathematics and coding to law and medicine, presenting GPT-4 as a model that displays general problem-solving capabilities, compositional reasoning, and even creativity. The strength of this work lies in its empirical breadth and anecdotal evidence that illustrates GPT-4's emergent behaviors—solving complex math problems, generating functional code, and explaining abstract humor—previously unattainable by prior models.

Despite these advancements, the paper does not overlook GPT-4's significant limitations. The model continues to exhibit hallucinations—producing incorrect or fabricated information—and lacks planning capability over extended contexts. Its output remains highly sensitive to prompt phrasing, which complicates consistent performance across varying user inputs. The study's scientific rigor is further limited by the absence of defined benchmarks or quantitative evaluation measures, which render it more exploratory than conclusive. Furthermore, there is still much to learn about the moral and societal ramifications of using such potent language models on a big scale. As a result, although the study marks a significant advancement in AI research, it also poses important queries regarding governance, control, and the appropriate application of AI.



## CHAPTER 3

### SYSTEM METHODOLOGY

#### 3.1 System Design Diagram

##### 3.1.1 System Architecture Diagram

The Presentation Layer, Application Layer, and Integration Layer are the three levels that make up the system architecture. Each layer contains components that interact to transform natural language input into a visualized and emulatable network topology. The subsections below explain each component in detail.

###### 3.1.1.1 Presentation Layer – User Interface

Users interact with the application through the User Interface, which acts as the system's entrance. Because it was created with frameworks like Gradio or Streamlit, users may input commands in normal language and get instant feedback. The interface displays the generated topology, supports iterative modifications, and provides options to export the final diagram. Its simplicity ensures that users can focus on network design without needing technical expertise in topology modeling.

###### 3.1.1.2 Application Layer – LLM Processing Module

The interpretation of natural language commands is the responsibility of the LLM Processing Module. The system converts user input into structured JSON output by using a Large Language Model (LLM) and the OpenAI API. This JSON defines the devices, their attributes, and the connections between them. The LLM acts as the “intelligence” of the system, bridging human language and machine-readable topology definitions.

###### 3.1.1.3 Application Layer – JSON Parser and Validator

The output produced by LLM adheres to the correct schema thanks to the JSON Parser and Validator. The user is prompted to refine the command if the JSON is deemed invalid or incomplete. The system's integrity depends on this stage because only legitimate JSON structures can be exported and visualized by later modules.

### 3.1.1.4 Application Layer – Graph Manager

The Graph Manager, implemented using **NetworkX**, maintains the internal representation of the network. It stores information about nodes, device types, and connections, providing a flexible structure for modifications. The Graph Manager ensures consistency across multiple updates and enables efficient handling of complex topologies.

### 3.1.1.5 Application Layer – Visualization Module

The Matplotlib-based Visualization Module turns the internal graph structure into a graphical network representation. Routers, switches, firewalls, servers, and end devices are represented by custom icons, and overlaps and clutter are minimized through layout optimization. This module guarantees that the topologies that are produced are not only useful but also aesthetically pleasing and simple to understand.

### 3.1.1.6 Application Layer – Export Module

The Export Module provides two export options: **PNG** and **JSON**. The PNG format provides users with a static image of the created topology and is designed for reporting and documentation purposes. In contrast, the JSON format is set up to be completely compatible with GNS3, which allows users to switch between design and emulation with ease.

### 3.1.1.7 Integration Layer – GNS3 Emulator

The external environment in which the output JSON file can be imported is represented by the GNS3 Emulator. Through this integration, the AI-generated architecture may be implemented in an actual emulation environment, enabling additional setup and validation. This integration drastically cuts down on the amount of time needed for testing and deployment by doing away with the requirement to manually reconstruct diagrams.

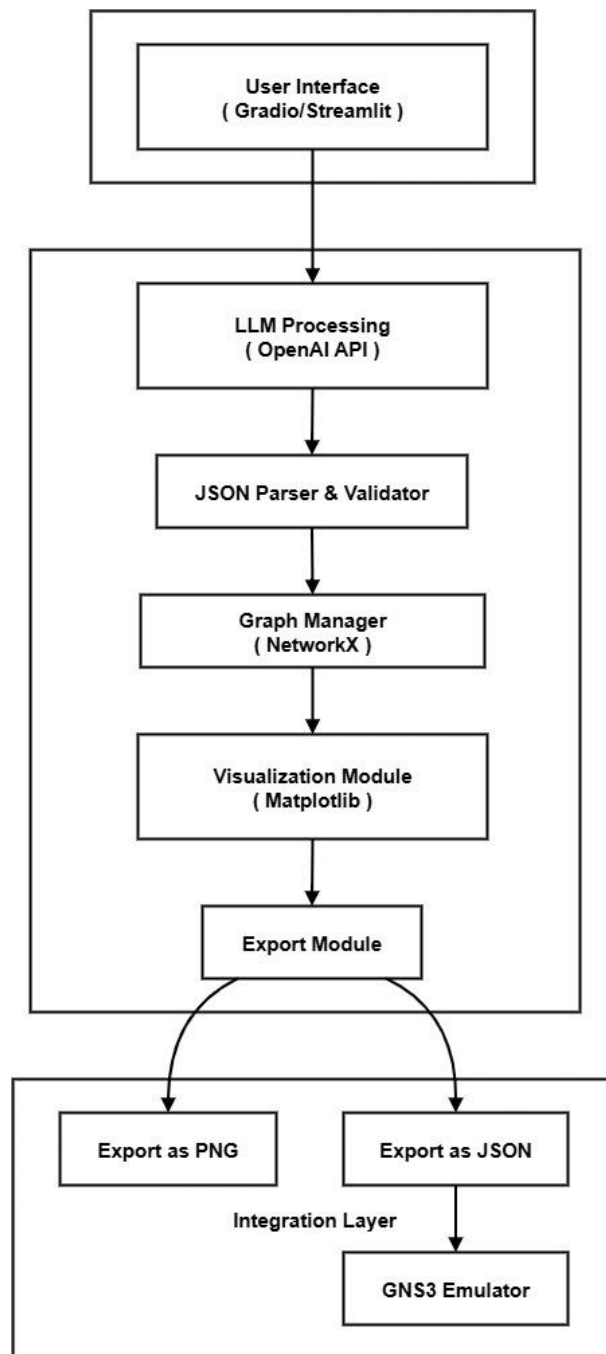


Figure 3.1.1 System Architecture Diagram

### 3.1.2 Use Case Diagram and Description

The use case diagram illustrates the interaction between the User and the proposed system. The user provides natural language commands which are processed by the system to generate, modify, and export network topologies. The diagram also shows the integration with GNS3, where exported JSON files can be directly opened for emulation.

The main use cases of the system are described as follows:

- **Input Command:** The user enters natural language instructions (e.g., “Add a router connected to a switch”). This is the starting point for creating or modifying a network topology.
- **Generate Topology:** Based on the command, the system uses the LLM engine to generate a structured JSON response that defines the required devices and connections. This process always includes Validate JSON to ensure schema compliance.
- **Modify Topology:** The user may add, remove, or change devices and links. Similar to generation, this process includes Validate JSON to maintain consistency.
- **Validate JSON:** Ensures that the AI-generated or modified JSON output follows the correct format before being applied to the topology. This step is included in both “Input Command” and “Modify Topology.”
- **View Topology:** Displays the generated network diagram with appropriate icons and layers. This use case is included after both generation and modification.
- **Export Topology:** Allows the user to save the current network diagram. This use case includes two options:
  - **Export as PNG:** Produces a static image for documentation.
  - **Export as JSON:** Generates a GNS3-compatible file for emulation.
  - **Open GNS3:** Once the JSON file is exported, the user can open it in GNS3 for further configuration and simulation. This use case is included in Export as JSON.

The <<include>> relationships ensure that dependent processes such as JSON validation and diagram viewing are always executed as part of generation or modification.

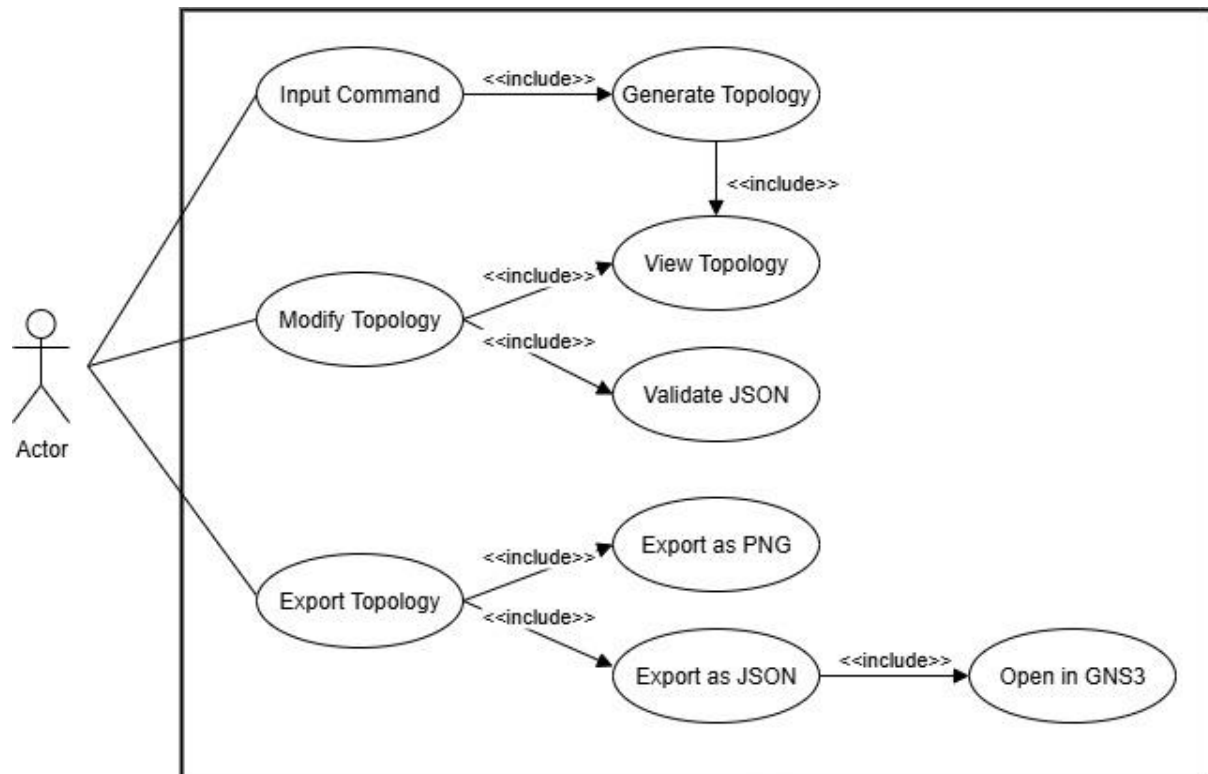


Figure 3.1.2 Use Case Diagram

Use Case	Actor	Precondition	Postcondition	Description
<b>Input Command</b>	User	The system is running and ready to accept input.	A natural language instruction is received by the system.	The user provides a command such as “ <i>Add a router connected to a switch.</i> ” The command is forwarded to the LLM engine for interpretation.
<b>Generate Topology</b>	System	A valid natural language command is provided.	A structured JSON response representing nodes and connections is produced.	The system uses the LLM to generate a JSON file that defines devices and their interconnections, which will later be visualized.
<b>Modify Topology</b>	User	An existing topology has already been generated.	The topology is updated with the new changes.	The user edits the diagram by adding, removing, or updating nodes and links. The system validates and reflects these changes in real time.
<b>Validate JSON</b>	System	JSON output is received from the LLM or after a modification.	JSON is either accepted as valid or rejected with an error message.	The system checks the structure of the JSON file against the expected schema to ensure it can be parsed into a network diagram.
<b>View Topology</b>	System	A valid topology has been generated or modified.	The diagram is displayed with icons and labels.	The system renders the network diagram using NetworkX and Matplotlib, displaying devices and connections clearly to the user.
<b>Export as PNG</b>	User	A valid topology is displayed in the system.	A PNG image of the diagram is saved.	The user saves the current network topology as an image for documentation and reporting purposes.
<b>Export as JSON</b>	User	A valid topology is displayed in the system.	A JSON file is generated that conforms to GNS3 schema.	The user exports the topology as a JSON file, which is compatible with GNS3 for emulation.
<b>Open GNS3</b>	User	A valid JSON export has been created.	The network topology is loaded in GNS3 for simulation.	The user imports the JSON file into GNS3 to emulate and further configure the generated network.

Table 3.1.2 Use Case Table

### 3.1.3 Activity Diagram

The activity diagram shows how the suggested system works step-by-step, starting with user input and concluding with the export of a legitimate network topology. When the user uses the interface to enter a command in natural language, the procedure starts. The directive is sent to the Large Language Model (LLM) via the OpenAI API, which decodes it and generates a JSON response containing the specified nodes and connections.

The JSON response is then parsed and verified by the system. The topology is updated in the internal graph structure and graphically shown using NetworkX and Matplotlib if the JSON is valid. The diagram is then shown to the user, who can choose to export it or keep working with it. At this point, the user can choose to export the topology as a PNG picture for documentation purposes or as a JSON file for direct input into GNS3. If no export is selected, the user is free to continue modifying or growing the topology.

When the user enters invalid JSON, the system will display an error notice and ask them to re-enter or modify their instruction. The generation of only legitimate, deployable topologies is guaranteed by this feedback loop. The system's automated and iterative nature, which drastically cuts down on the time and effort needed for network topology design as compared to more conventional human techniques, is generally highlighted in the activity diagram.

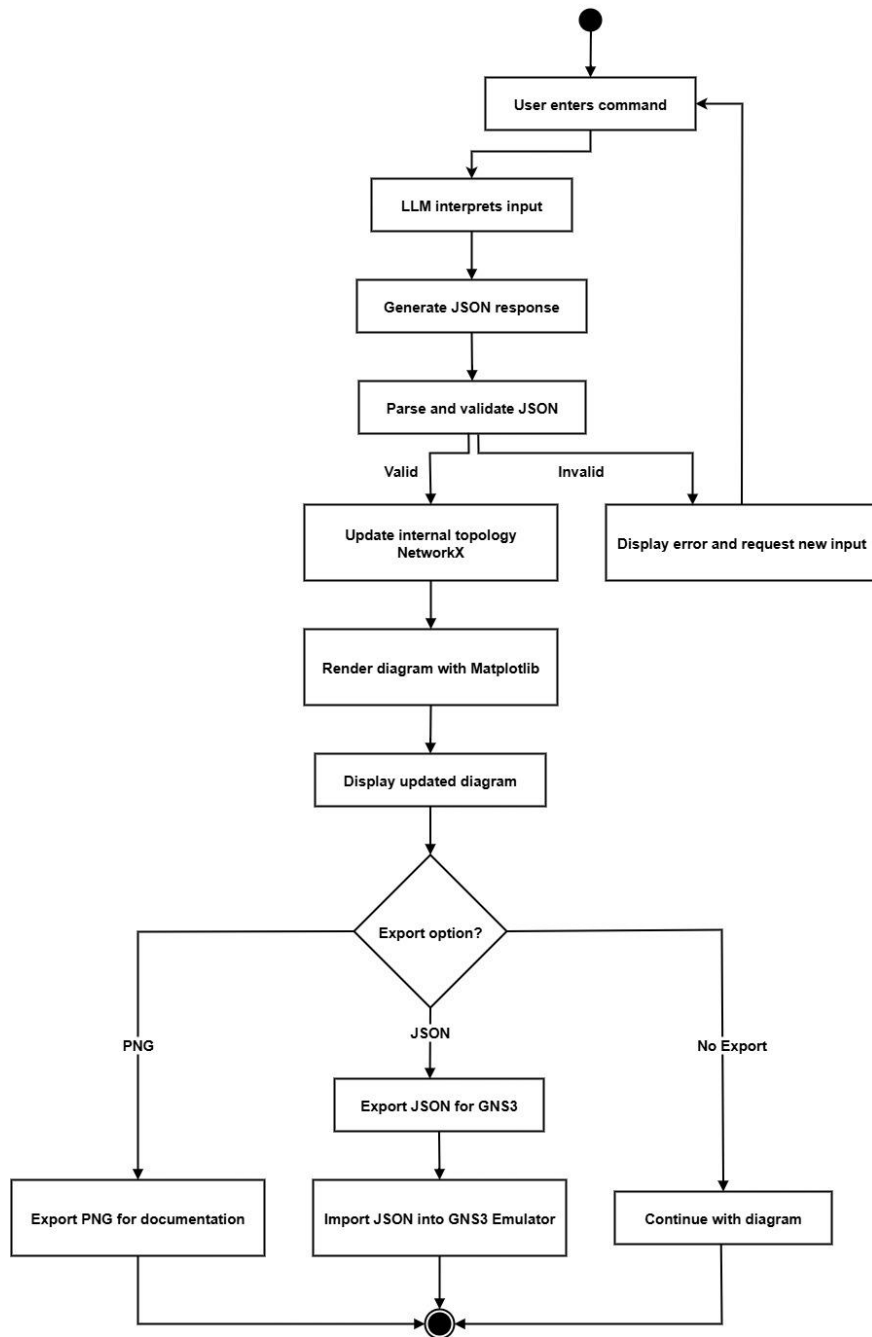


Figure 3.1.3 Activity Diagram



## CHAPTER 4

### SYSTEM DESIGN

#### 4.1 System Block Diagram

The block diagram offers a high-level summary of the advised system's modular architecture. The User Interface is where natural language commands are entered to start the workflow. The LLM Engine interprets user instructions and produces a structured JSON output outlining the necessary devices and their connections after processing these commands by utilizing the OpenAI API.

The Parser and Validator then receives the JSON output and verifies that the data adheres to the proper format and schema. The Graph Manager, which is implemented using NetworkX, receives valid JSON data and stores the network nodes, device kinds, and connections as part of the internal topology model. This structured representation forms the foundation for both visualization and export functions.

The Diagram Renderer, implemented using Matplotlib, converts the graph data into a visual network topology. Each device is represented with a standardized icon, and the complete diagram is displayed to the user via the interface.

Once the topology has been generated, the system provides two export options through the Export Module. The first is PNG export, which allows the diagram to be saved for documentation and reporting purposes. The second method is JSON export, which creates a file that is compatible with GNS3. By importing this file instantly into the GNS3 Emulator, more network emulation and setup can be accomplished without the need for manual reconstruction. By guaranteeing a distinct division of duties, this modular design keeps the system expandable, maintainable, and interoperable with industry-standard technologies like GNS3.

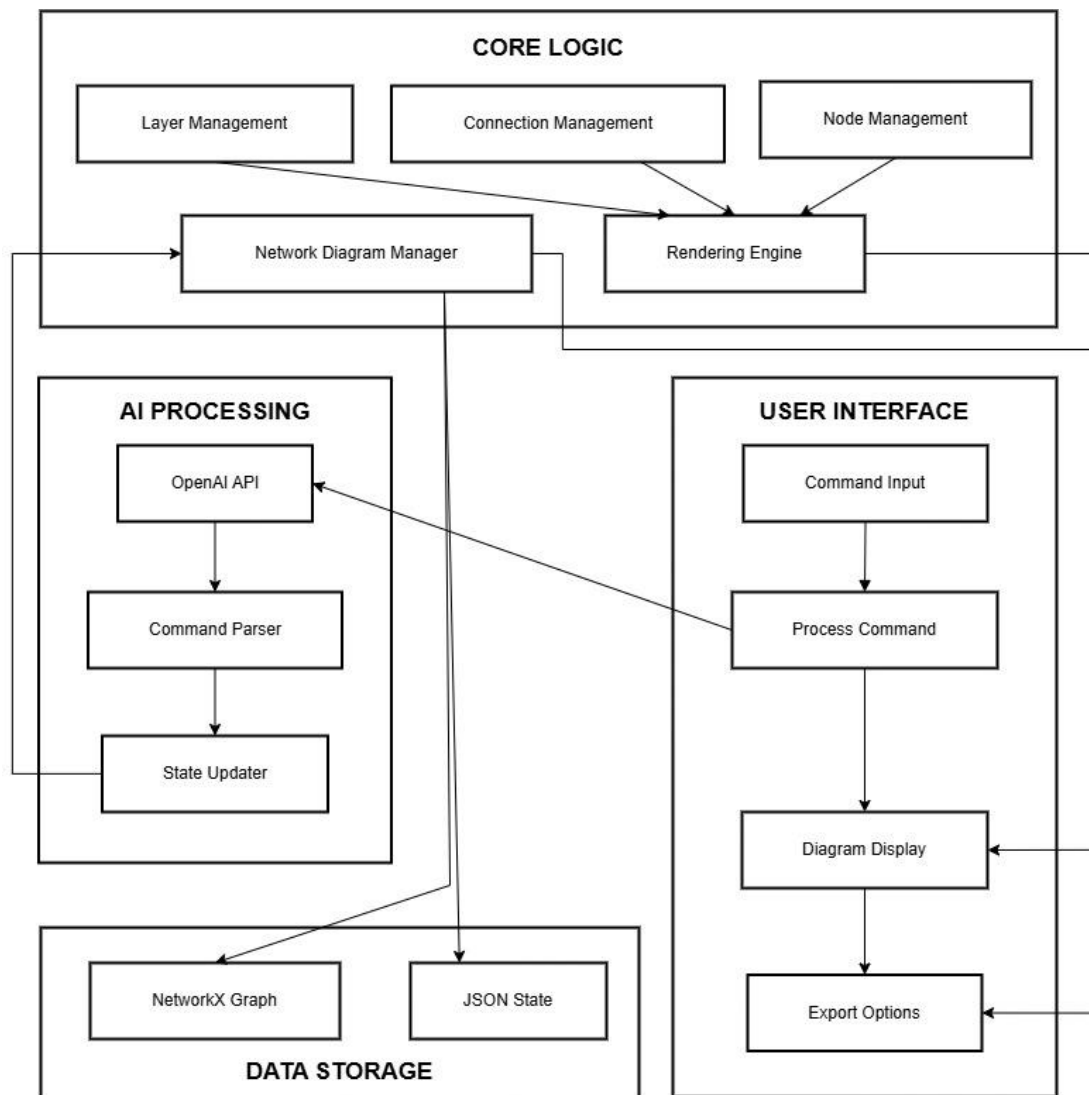


Figure 4.1 System Block Diagram

## 4.2 System Components Specifications

System components specifications describes the major components used in developing the proposed system, including the programming environment, libraries, frameworks, and external tools. Each component plays a distinct role in ensuring the functionality, visualization, and integration of the network topology generator.

Component	Specification / Tool	Description
<b>Programming Language</b>	Python 3.10	Core language used to implement system logic, LLM integration, and visualization.
<b>LLM Engine</b>	OpenAI API (GPT-based model)	Processes natural language input and generates JSON output describing network topology.
<b>Graph Manager</b>	NetworkX Library	Handles internal graph structure, storing nodes, edges, and attributes of the network.
<b>Visualization Module</b>	Matplotlib Library	Renders network diagrams with custom icons, layout optimization, and layered representation.
<b>User Interface</b>	Gradio / Streamlit	Provides input/output interface for entering commands, viewing diagrams, and exporting results.
<b>Export Module</b>	Custom Python Functions	Supports PNG export for documentation and JSON export for GNS3 integration.
<b>Emulation Platform</b>	GNS3	Validates exported JSON files by allowing users to emulate and configure generated topologies.
<b>IDE</b>	Visual Studio Code	Used for development, debugging, and integration testing of the system.

Table 4.2 Table of Components

### 4.2.1 Programming Language

Python 3.10 was used in the system's development. Python was chosen because of its adaptability, extensive library ecosystem, and simplicity in integrating with external APIs. It facilitates the execution of the project's necessary activities for graph management, natural language processing, and visualization.

### 4.2.2 LLM Engine

The **Large Language Model (LLM)** is accessed through the **OpenAI API (DeepSeek:R1(Free))**, which processes natural language commands and generates structured JSON output. This component is responsible for interpreting user instructions into machine-readable topology definitions, serving as the core intelligence of the system.

### 4.2.3 Graph Manager

The **Graph Manager** was implemented using the **NetworkX library**. It maintains the internal graph representation of the network, storing nodes, device types, and connections. This module ensures consistency across updates and provides the foundation for visualization and export.

### 4.2.4 Visualization Module

The visualization of topologies was developed using **Matplotlib**. This module generates diagrams with standardized device icons, layered segmentation (presentation, application, and data layers), and optimized layouts to minimize overlaps. The visualization ensures clarity and readability even for complex topologies.

### 4.2.5 User Interface

A simple and interactive **User Interface (UI)** was built using frameworks such as **Gradio** and **Streamlit**. Users can inspect the created topologies, access export functions, and enter instructions in normal language using the user interface. By making the system usable by both professionals and non-experts, it highlights usability.

### 4.2.6 Export Module

The **Export Module** provides two output options. The first is exporting diagrams as PNG pictures for use in reports and documentation. The second allows for a smooth transition from AI-assisted design to emulation and testing by exporting as a JSON file that is compatible with GNS3.

### 4.2.7 Emulation Platform

An industry-standard tool for simulating networks, the GNS3 Emulator, can be used to import the produced JSON files. This guarantees that the topologies produced by AI are not only observable but also deployable for testing, configuration, and emulation in the real world.

### 4.2.8 Development Environment

Visual Studio Code was used for the creation and testing of the entire system. The IDE made it easier to integrate APIs, libraries, and external modules by offering debugging, code management, and extension support.

## CHAPTER 5

### SYSTEM IMPLEMENTATION

#### 5.1 Hardware Setup

The hardware implementation for this project utilizes a computer system as the primary development and deployment platform. This workstation will handle all computational tasks, including network diagram generation, visualization processing, and system testing. Key hardware components have been selected to ensure smooth operation during both the development phase and final execution of the network diagram generator application.



Figure 5.1 HP laptop

Description	Specifications
Model	HP Laptop 15s-du3xxx
Processor	11 <sup>th</sup> Gen IntelI CoreI i5-1135G7
Operating System	Windows 11
Graphic	NVIDIA GeForce GT 930MX 2GB DDR3
Memory	8 GB RAM DDR4 Memory
Storage	512GB SSD

Table 5.1 Specifications of laptop

## 5.2 Software

### 5.2.1 VSCode

The project uses Visual Studio Code (VSCode) for Python development, with the OpenAI API SDK for AI integration and NetworkX/PyVis for network diagram generation. Postman tests API connections while Git handles version control. This lightweight, cross-platform stack ensures efficient diagram processing and visualization.



Figure 5.2.1 VSCode

### 5.2.2 GNS3

A widely adopted network simulation and emulation platform, used in this project to validate the generated topologies. The system exports diagrams in JSON format that can be directly imported into GNS3, allowing users to emulate and configure the generated networks without manually reconstructing them.



Figure 5.2.2 GNS3

### 5.2.3 OpenAI

The `openai` package serves as the primary interface for leveraging GPT-based models to interpret natural language network descriptions. This tool enables the system to convert user-provided text into structured data by identifying entities and relationships described in plain language. It forms the foundation of the system's intelligent understanding and automation.

### 5.2.4 json

The `json` module is employed to handle the formatting and parsing of API requests and responses. It ensures consistent communication with the OpenAI API, converting structured data into serialized formats and vice versa, which is essential for maintaining a reliable data exchange pipeline.

### 5.2.5 networkx

Graph-based data structures can be constructed, analyzed, and worked with using the `networkx` core library. It offers an adaptable framework for simulating network nodes, edges, and their characteristics, allowing topologies to be dynamically created from input specifications.

### 5.2.6 numpy

`numpy` supports efficient numerical operations within the system, particularly for layout calculations and performance optimizations. Its array-based computing capabilities help to accelerate coordinate computations during graph visualization.

### 5.2.7 matplotlib (pyplot, patches, colors, offsetbox)

The system utilizes several components from the `matplotlib` library for generating visual representations of network diagrams. `Pyplot` provides the base plotting functionality. `Patches` allows the addition of custom graphical shapes like nodes and edge markers. `Colors` facilitates color scheme management, enhancing the visual clarity of diagrams. `Offsetbox` is used for advanced annotations and embedding visual elements within the diagram, such as icons or image overlays.



### **5.2.8 gradio**

gradio is used to build an interactive, web-based user interface for the system. It allows users to input natural language descriptions and receive real-time graphical output. Gradio simplifies the deployment of machine learning applications and supports rapid prototyping with minimal configuration.

### **5.2.9 io / BytesIO**

The io module, and specifically BytesIO, enables in-memory byte stream handling. This functionality is crucial for rendering and transmitting image data between system components without writing to disk, supporting seamless performance in both local and web-based environments.

### **5.2.10 re**

The re module is responsible for regular expression-based pattern matching and data extraction. It plays a key role in preprocessing the input text by identifying network components such as node names, connections, and attributes.

### **5.2.11 random**

The random module introduces controlled variability in graph layouts and appearance, particularly useful in scenarios where deterministic node positioning may lead to overlapping or poor spacing in the visualization.

### **5.2.12 time**

time is used for performance measurement and benchmarking. It tracks processing durations across stages like text parsing, graph generation, and rendering, helping ensure that system performance targets are met.

### **5.2.13 os**

The os module manages interactions with the file system, including temporary storage and resource loading. It supports operations like file path resolution and environment configuration during execution or deployment.

## 5.3 Setting and Configuration

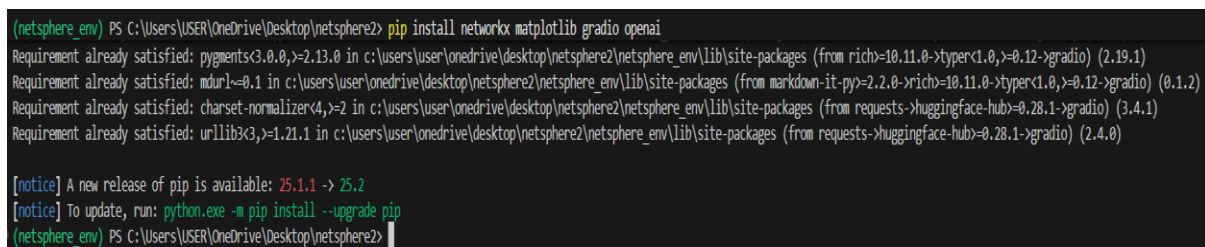
### 5.3.1 Development Environment Setup

Python 3.10 was used as the primary programming language for the system's development, and Visual Studio Code (VS Code) was used as the integrated development environment. A specialized virtual environment (venv) was developed to control dependencies and preserve intermodular compatibility. The isolation and reproducibility of the development environment were guaranteed by this configuration.

### 5.3.2 Library and Dependency Installation

To facilitate the system's implementation, a number of Python libraries were set up. Utilizing Gradio/Streamlit for the user interface, Matplotlib for visualization, NetworkX for graph management, and the OpenAI SDK for API integration. The following command was used to install dependencies via the Python package manager:

```
pip install networkx matplotlib gradio openai
```



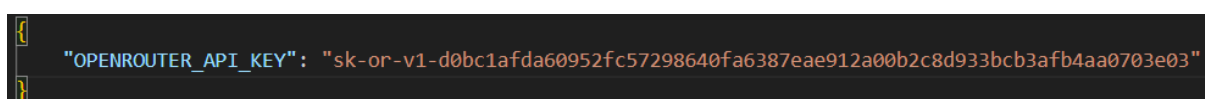
```
(netsphere_env) PS C:\Users\USER\OneDrive\Desktop\netsphere2> pip install networkx matplotlib gradio openai
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in c:\users\user\onedrive\desktop\netsphere2\netsphere_env\lib\site-packages (from rich>=10.11.0->typer<1.0,>=0.12->gradio) (2.19.1)
Requirement already satisfied: mdurl<=0.1 in c:\users\user\onedrive\desktop\netsphere2\netsphere_env\lib\site-packages (from markdown-it-py>=2.2.0->rich>=10.11.0->typer<1.0,>=0.12->gradio) (0.1.2)
Requirement already satisfied: charset-normalizer<4,>=2 in c:\users\user\onedrive\desktop\netsphere2\netsphere_env\lib\site-packages (from requests->huggingface-hub>=0.28.1->gradio) (3.4.1)
Requirement already satisfied: urllib3<3,>=1.21.1 in c:\users\user\onedrive\desktop\netsphere2\netsphere_env\lib\site-packages (from requests->huggingface-hub>=0.28.1->gradio) (2.4.0)

[notice] A new release of pip is available: 25.1.1 -> 25.2
[notice] To update, run: python.exe -m pip install --upgrade pip
(netsphere_env) PS C:\Users\USER\OneDrive\Desktop\netsphere2>
```

Figure 5.3.2 Library Installation

### 5.3.3 OpenAI API Configuration

To enable natural language processing, the system was connected to the **OpenAI API**. An API key was obtained from the OpenAI platform and configured securely as an environment variable. This approach prevented hardcoding sensitive information within the source code. A preliminary test was conducted to verify API connectivity and confirm that valid JSON responses could be generated from sample commands.



```
{
  "OPENROUTER_API_KEY": "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"
}
```

Figure 5.3.3 OpenAI API Configuration

### 5.3.4 System Configuration

Standardized icons, color codes, and layout criteria were set up in the visualization module to guarantee uniformity between created topologies. Additionally, the JSON Parser and Validator was configured to enforce schema compliance by blocking the processing of faulty or incomplete JSON structures. The export directories were arranged into distinct folders for JSON and PNG files, making retrieval and storage easier.

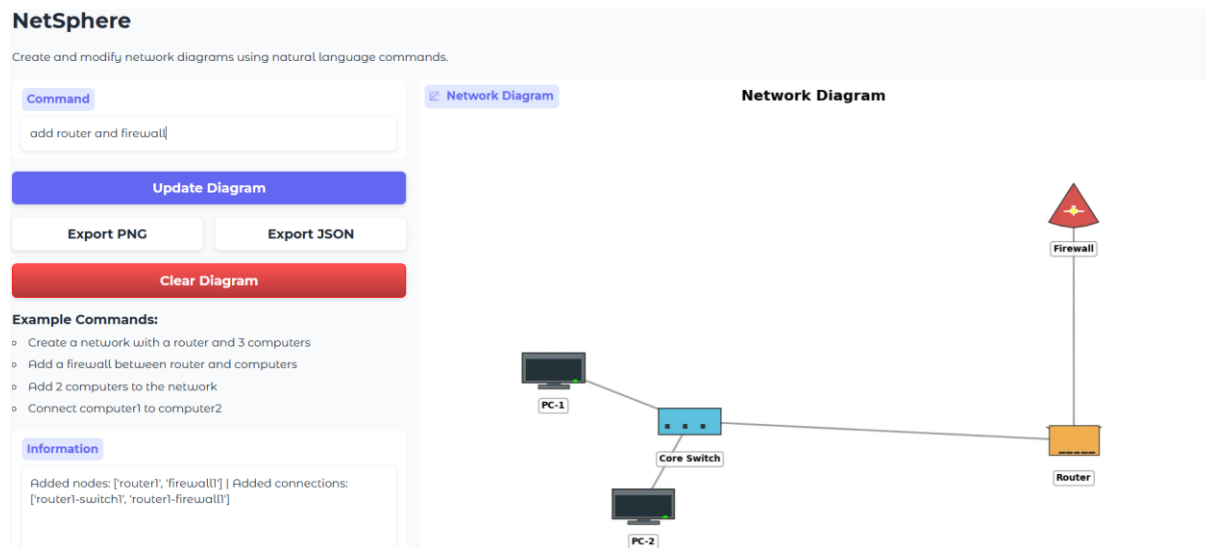


Figure 5.3.4 Visualization Settings

### 5.3.5 GNS3 Integration Setup

The **GNS3 emulator** was installed to serve as the external platform for validating exported topologies. The system was configured to generate JSON files compatible with GNS3's schema. Test imports were performed to confirm that exported topologies could be opened in GNS3 with correct **node placement** and **connectivity**.

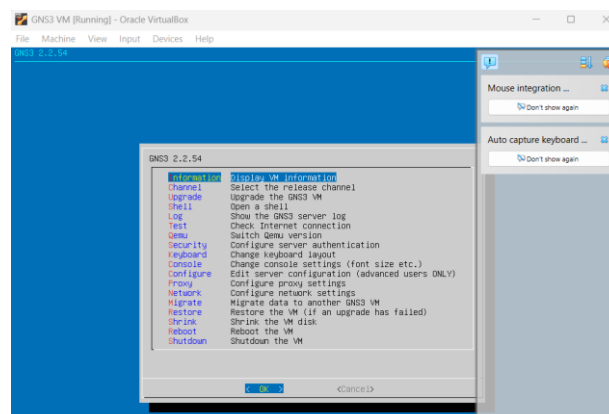


Figure 5.3.5.1 GNS3 VM Setup

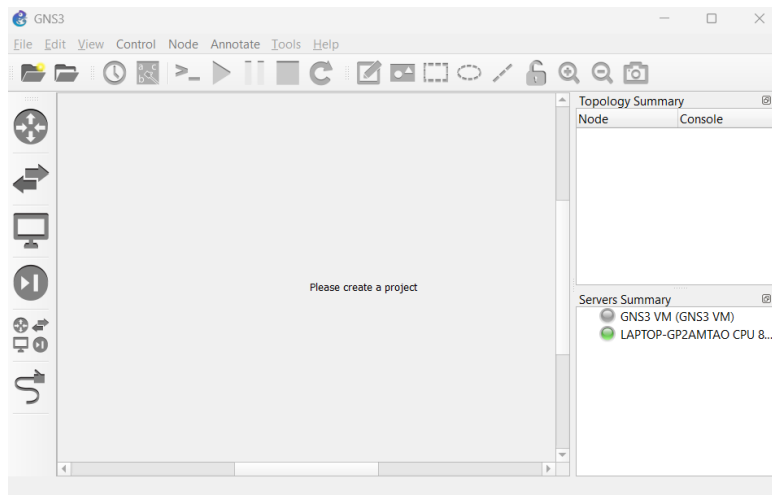


Figure 5.3.5.2 GNS3 Setup

### 5.3.6 Testing Configuration

Initial test cases were executed to verify the overall system setup. Commands such as “*add a router connected to a switch*” were processed to confirm diagram generation, export functionality, and JSON compatibility. Validation in GNS3 ensured that the exported files matched the structure of the user-generated diagrams and that all nodes were correctly connected.

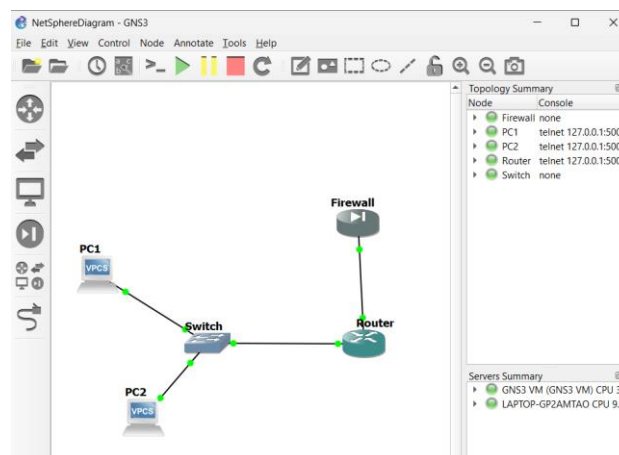


Figure 5.3.6 JSON to GNS3

## 5.4 System Operation

### 5.4.1 Launching the System

```
(netsphere_env) PS C:\Users\USER\OneDrive\Desktop\netsphere2> python netsphere.py
* Running on local URL: http://127.0.0.1:7860
* To create a public link, set `share=True` in `launch()`.
```

Figure 5.4.1 Launching in VSCode

### 5.4.2 Entering a Command

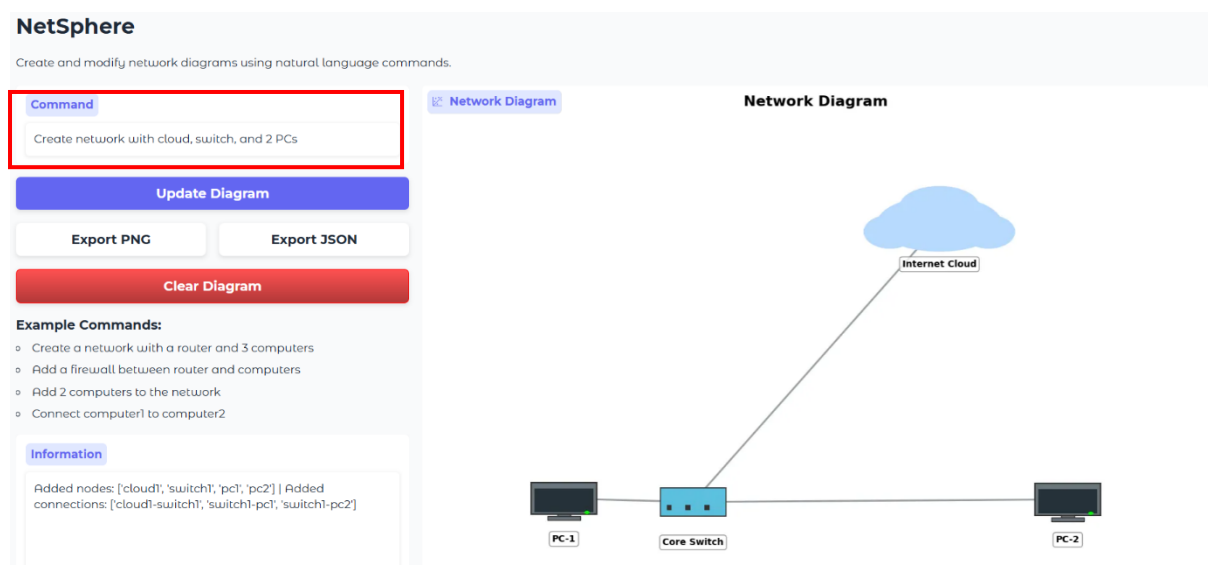


Figure 5.4.2 Entering a natural language command into the system

### 5.4.3 LLM Processing and JSON Generation

```
Raw AI response: {
  "nodes": [
    {"id": "cloud1", "name": "Cloud", "type": "cloud", "details": {}},
    {"id": "switch1", "name": "Core Switch", "type": "switch", "details": {}},
    {"id": "pc1", "name": "PC 1", "type": "computer", "details": {}},
    {"id": "pc2", "name": "PC 2", "type": "computer", "details": {}}
  ],
  "connections": [
    {"source": "cloud1", "target": "switch1", "type": "standard"},
    {"source": "switch1", "target": "pc1", "type": "standard"},
    {"source": "switch1", "target": "pc2", "type": "standard"}
  ]
}
```

Figure 5.4.3 JSON output generated by the LLM engine

## 5.4.4 Network Diagram Visualization

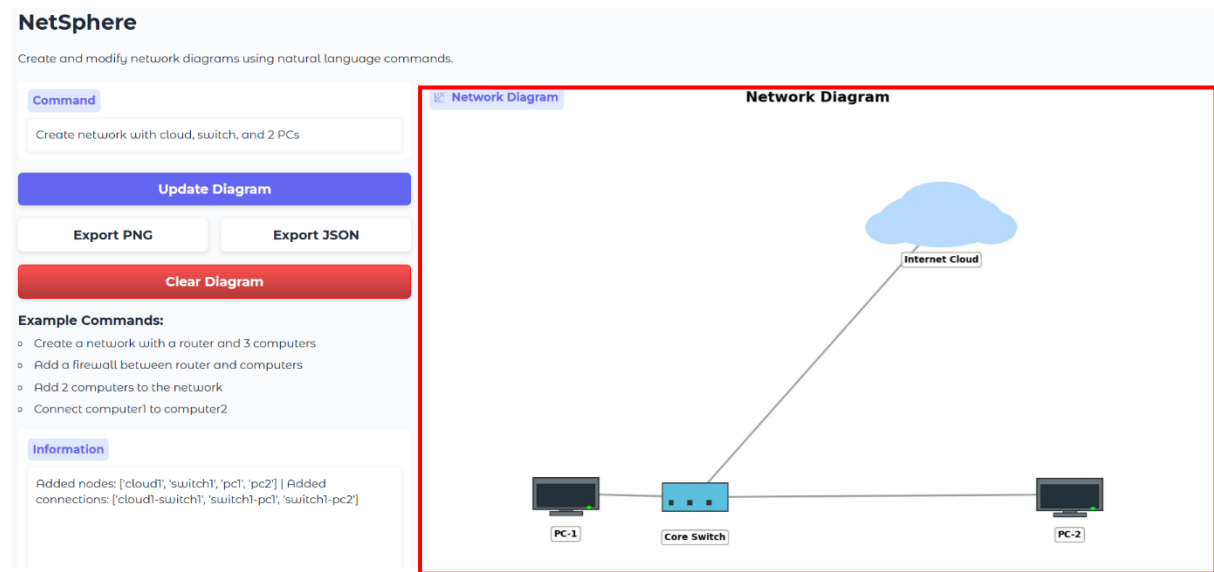


Figure 5.4.4 Network topology visualization with icons and layers

## 5.4.5 Modifying the Topology

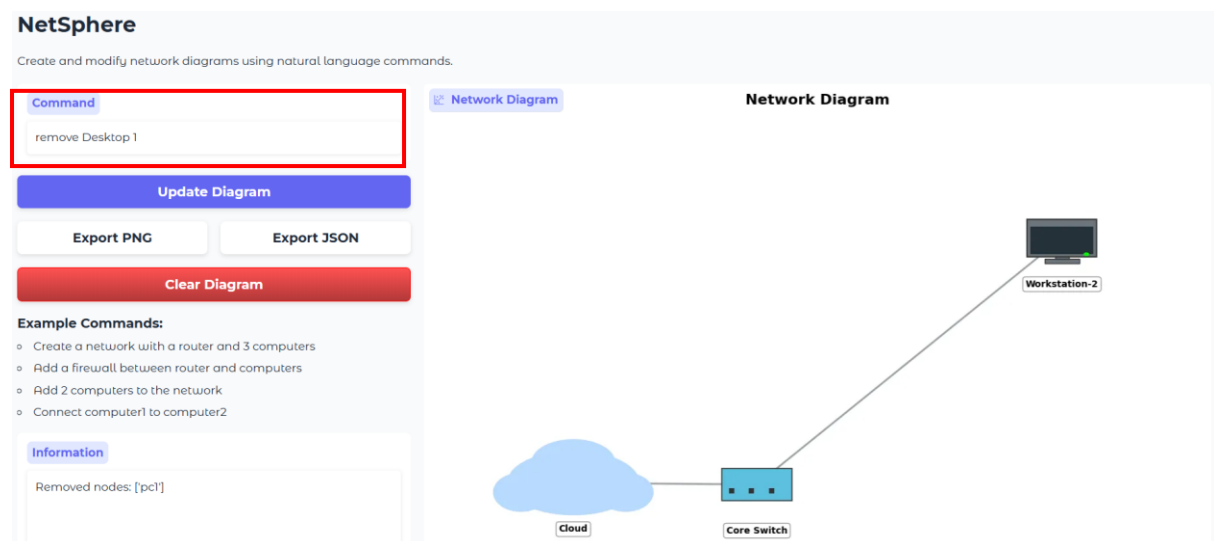


Figure 5.4.5 Modified topology

5.4.6 Exporting Topology in PNG

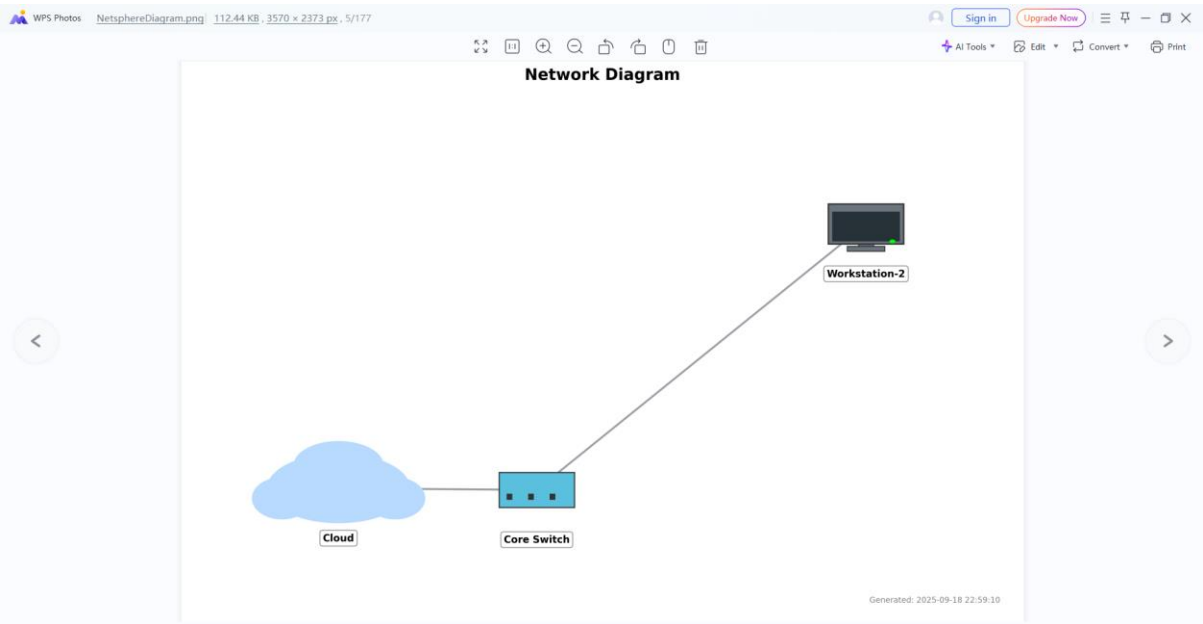


Figure 5.4.6 Export confirmation for PNG

5.4.7 Exporting Topology in JSON

Name	Date modified	Type	Size
▼ Today			
📁 GNS3_Project	18/9/2025 12:39 PM	File folder	
Name	Date modified	Type	Size
▼ Today			
🌐 project.gns3	18/9/2025 12:39 PM	GNS3 Project File	10 KB

Figure 5.4.7 Export confirmation for JSON

### 5.4.8 Opening JSON in GNS3

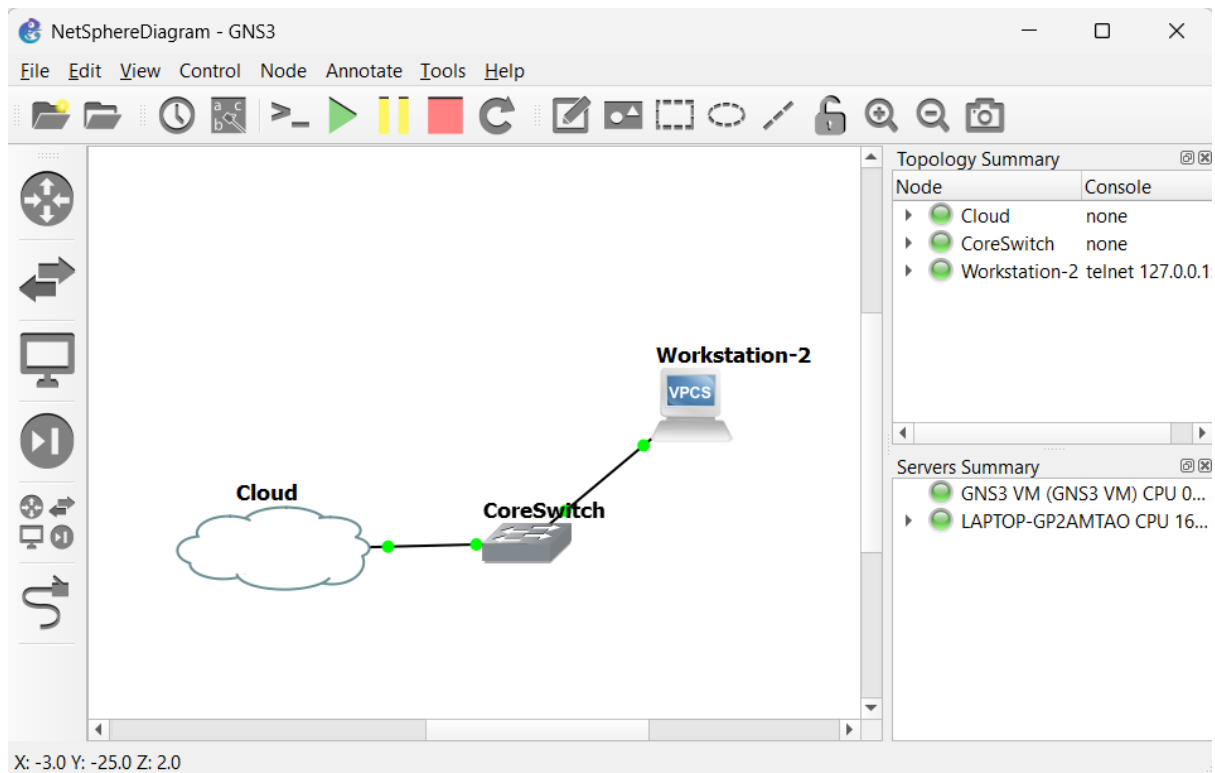


Figure 5.4.8 Generated network topology displayed inside GNS3



### 5.5 Implementation Issues and Challenges

In the initial year of the project (FYP1), a number of difficulties were faced, especially when connecting the OpenAI API with the system and managing the unpredictability of user-provided natural language input. The model has to be thoroughly tested and adjusted to ensure that the replies consistently generated legitimate JSON representations. Another problem was generating bespoke network device icons with Matplotlib, where preserving clarity, precise alignment, and avoiding overlaps in complex topologies proved tough. Eventually, these problems were resolved through iterative development and improvement, enabling the system to produce accurate and aesthetically pleasing graphics.

The project's attention turned to extending the system for direct integration with GNS3 in the second phase (FYP2), which presented a unique set of difficulties. Even little discrepancies could lead to errors or unsuccessful imports because the produced JSON file has to adhere exactly to GNS3's structure and formatting standards. Making sure that every node and its connections in the produced JSON correctly represented the user-generated topology was a significant challenge. Once networks were imported into GNS3, inconsistencies in parsing or incompatibilities between identifiers occasionally resulted in disconnected or incomplete networks. To resolve these problems, the system's internal graph structure had to be carefully mapped to GNS3's topology model. Additionally, it was necessary to conduct numerous tests to confirm that exported topologies were operational.

### 5.6 Concluding Remark

The system implementation phase effectively shown that automating network topology generation and integration with GNS3 through the use of Large Language Models (LLMs) is feasible. The system was able to visualize the associated diagrams, convert natural language commands into appropriate JSON representations, and export the output in both PNG and JSON formats according to organized development and configuration. The JSON integration with GNS3 ensured that the generated topologies were not only visual but also deployable in an emulation environment for further validation and testing.

Despite facing obstacles such as JSON schema compatibility and sustaining node connectivity during GNS3 imports, these concerns were resolved through incremental refinement and testing. The goals of this phase were met by the finished implementation, which created a workable framework that is user-friendly, expandable, and modular. The evaluation and performance analysis that follow, covered in the next chapter, are also framed by this basis.

## CHAPTER 6

## SYSTEM EVALUATION AND DISCUSSION

### 6.1 System Testing and Performance Metrics

System testing was conducted to validate the functionality, accuracy, and reliability of the proposed network topology generator. The primary focus was to evaluate how well the system translated natural language commands into network diagrams and exported files compatible with GNS3.

The following performance metrics were used:

1. **Accuracy of JSON Output** – measuring whether the LLM-generated JSON matched the defined schema and represented all requested nodes and connections.

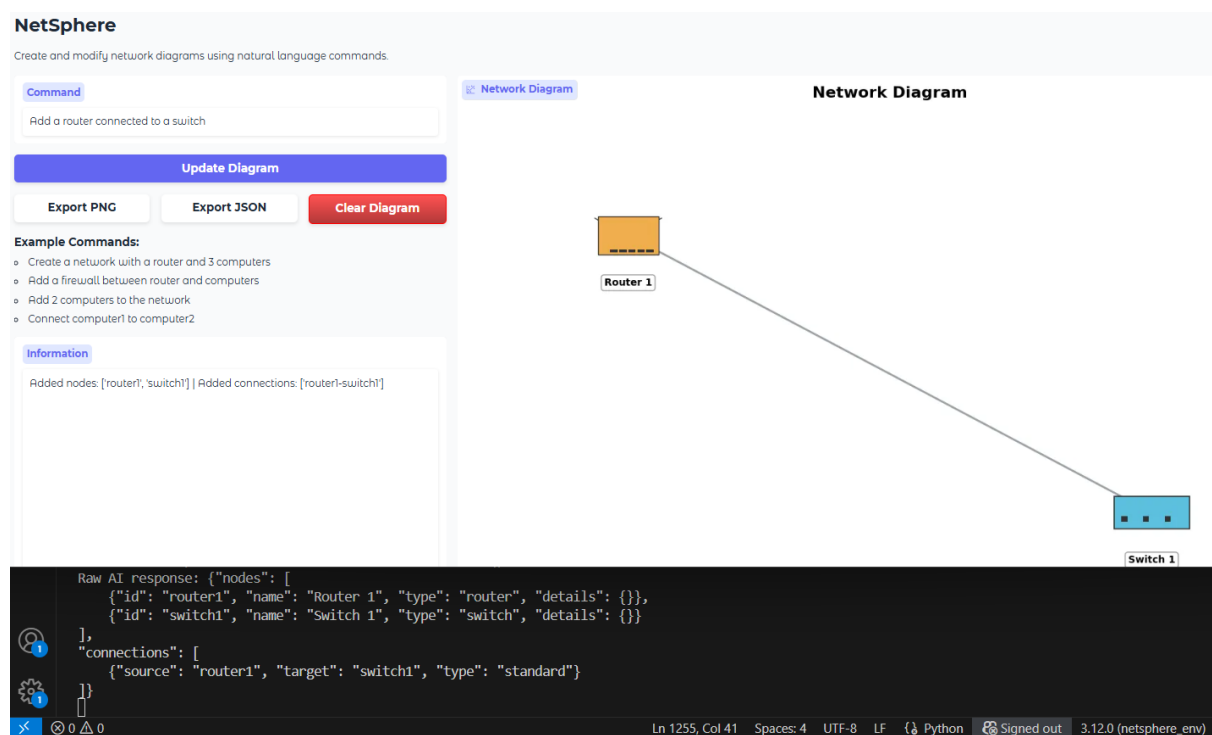


Figure 6.1.1 JSON output generated from user command

2. **Visualization Quality** – assessing diagram readability, node alignment, and clarity of icons and layers.

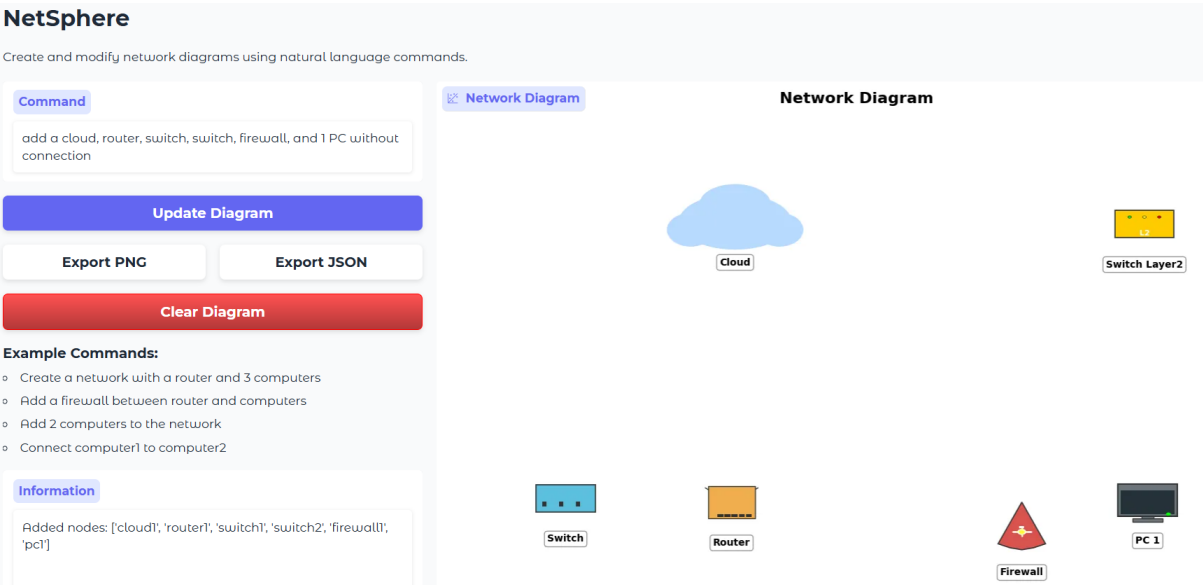


Figure 6.1.2 Visualization of the generated topology using Matplotlib

3. **Export Reliability** – verifying the correctness of exported PNG and JSON files.


 NetsphereDiagram.png	18/9/2025 11:32 PM	PNG File	106 KB
 GNS3_Project	18/9/2025 11:33 PM	File folder	

Figure 6.1.3 Export confirmation for PNG and JSON files

4. **GNS3 Compatibility** – ensuring JSON files could be imported directly into GNS3 without additional manual editing.

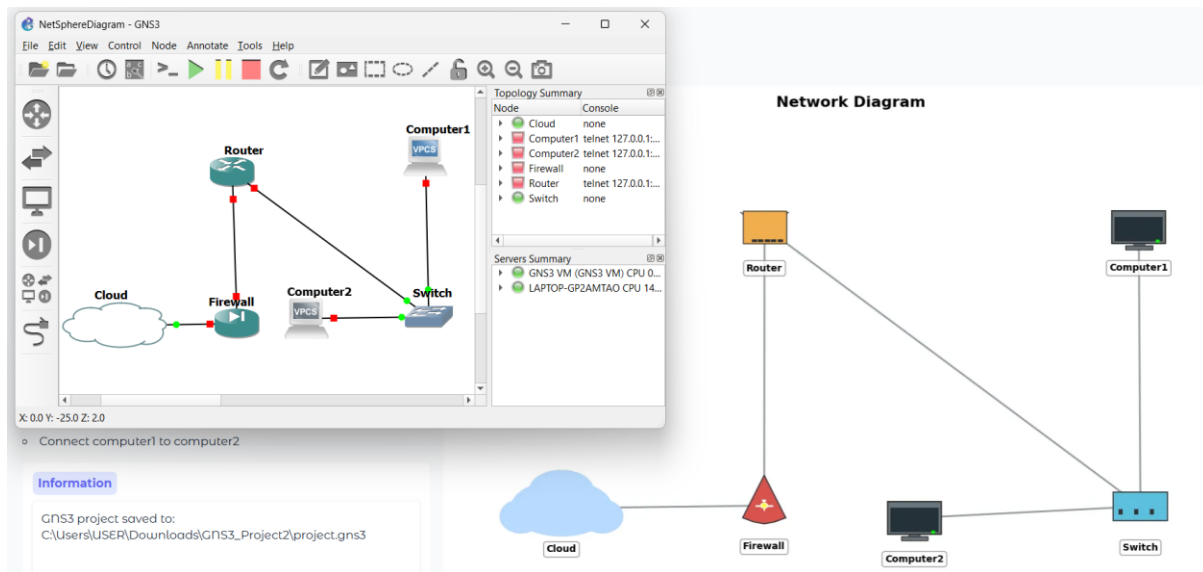


Figure 6.1.4 Exported JSON topology successfully imported into GNS3

5. **Response Time** – measuring how quickly the system generated outputs after receiving user input.

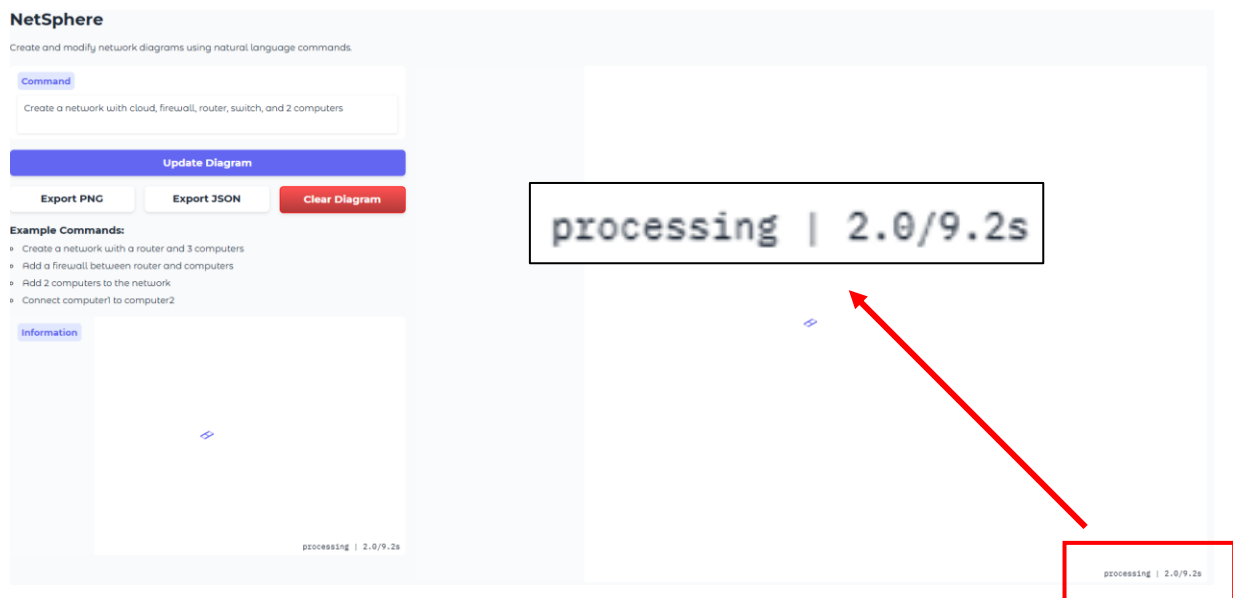


Figure 6.1.5 Response time measurement for command processing

## 6.2 Testing Setup and Result

The system was tested using a variety of natural language commands to evaluate its accuracy, visualization quality, export functionality, and GNS3 integration. Table 6.2 summarizes the testing process and results.

Test Case	Input Command	Expected Output	Actual Result	Status
TC1	<i>“Add a router connected to a switch.”</i>	JSON with router–switch connection, visual diagram showing both nodes.	JSON generated correctly, diagram rendered, export successful.	Pass
TC2	<i>“Add a firewall between the router and server.”</i>	JSON with firewall node placed between router and server; updated diagram.	JSON updated correctly, firewall icon added, connection validated.	Pass
TC3	<i>“Export topology as JSON.”</i>	JSON file created following schema, compatible with GNS3.	JSON exported, imported into GNS3 without errors.	Pass
TC4	<i>“Export topology as PNG.”</i>	PNG file generated with clear diagram and icons.	PNG exported successfully, diagram matches internal structure.	Pass
TC5	Complex topology with multiple routers, switches, and firewalls.	JSON with all devices and correct connectivity, readable diagram.	JSON generated but required minor schema adjustments for GNS3 import; visualization successful.	Pass

TC6	Invalid command ( <i>“Draw a cloud of happiness”</i> )	System should reject and return error.	Error message displayed, no JSON/diagram generated.	Pass
-----	---	---	---	------

Table 6.2 Testing Setup and Result Matrix

### 6.3 Project Challenges

The creation and assessment of the suggested system presented a number of difficulties. These difficulties included technological constraints, system integration, and performance issues.

A major obstacle was making sure the produced JSON files complied with GNS3's schema specifications. While the system could generate JSON output from natural language commands, not all LLM-generated structures were immediately valid for GNS3 import. The schema used by GNS3 is strict, requiring precise node types, properties, and connectivity definitions. Any mismatch led to parsing errors or incomplete topology imports. To overcome this difficulty, the JSON generating procedure had to be iteratively improved, and a validation layer had to be included to guarantee compliance prior to export.

When moving topologies from the visualization environment to GNS3, preserving node connectivity posed another major challenge. Networks became partially disconnected in certain instances where devices showed up in GNS3 correctly but were not connected as planned. This discrepancy resulted from the internal graph model's (NetworkX) representation of links differing from GNS3's intended JSON representation of them. Addressing this required adjustments in the export module to translate internal connections accurately into GNS3-compatible links.

Performance limitations also presented challenges, particularly when handling **large or complex topologies**. Although the method worked consistently for small to medium-sized networks, it occasionally produced cluttered visualizations or delayed reaction times when creating diagrams with a large number of related nodes. The diagram occasionally included overlapping elements as a result of the layout optimization algorithm's inability to arrange

nodes equally. This required fine-tuning of layout parameters and the incorporation of custom placement rules to maintain diagram clarity.

Furthermore, the project was constrained by its **dependence on the LLM engine**. Although the OpenAI API provided robust natural language understanding, the system occasionally produced ambiguous or incomplete JSON outputs, especially when users entered vague or overly complex commands. This necessitated additional error handling and prompted users to refine their inputs. The reliance on an external API also introduced concerns regarding cost, latency, and long-term accessibility.

Finally, **integration and testing with GNS3** posed logistical challenges. Since GNS3 requires a proper configuration of compute resources and templates, importing and validating exported topologies involved multiple setup iterations. The evaluation method became more involved when it was necessary to make sure that the system's exported JSON was both syntactically correct and functional within GNS3.

Notwithstanding these difficulties, every problem offered insightful information that directed iterative enhancements. By the end of the development period, the solutions put in place—such as JSON validation, export improvements, and layout modifications—helped create a system that was more dependable, resilient, and easy to use.



### 6.4 SWOT Analysis

A SWOT analysis was conducted to evaluate the strengths, weaknesses, opportunities, and threats of the system.

<p style="text-align: center;"><b>Strengths</b></p> <ul style="list-style-type: none"> <li>• Automated generation of network topologies from natural language.</li> <li>• Direct GNS3 integration via JSON export.</li> <li>• Clear visualization with layered diagrams and custom icons.</li> </ul>	<p style="text-align: center;"><b>Weaknesses</b></p> <ul style="list-style-type: none"> <li>• Dependent on LLM accuracy for JSON generation.</li> <li>• Performance degradation for large or complex topologies.</li> <li>• Limited customization options in visualization.</li> </ul>
<p style="text-align: center;"><b>Opportunities</b></p> <ul style="list-style-type: none"> <li>• Can be extended to support real-time network simulation.</li> <li>• Potential integration with other emulation tools besides GNS3.</li> <li>• Expansion into educational platforms for teaching networking.</li> </ul>	<p style="text-align: center;"><b>Threats</b></p> <ul style="list-style-type: none"> <li>• Dependency on external API (OpenAI) introduces cost and availability risks.</li> <li>• Compatibility issues may arise with future GNS3 updates.</li> <li>• Competing tools offering similar automation features.</li> </ul>

Table 6.4 SWOT Table

### 6.5 Objectives Evaluation

The objectives defined at the beginning of the project were evaluated against the outcomes of the implementation and testing:

Objective	Evaluation	Status
Automate network topology generation using natural language processing.	The system successfully used the LLM engine to convert user commands into JSON structures representing network devices and connections.	Achieved
Enable export of generated topologies in multiple formats.	The system allowed export as both <b>PNG</b> (for documentation) and <b>JSON</b> (for GNS3 integration).	Achieved
Ensure GNS3 compatibility for direct import of generated topologies.	Exported JSON files were tested and successfully imported into GNS3 with correct node placement and connectivity.	Achieved
Reduce the time required to design and emulate network topologies.	Automated generation and direct GNS3 import significantly reduced design and setup time compared to manual construction.	Achieved

Table 6.5 Objectives Evaluation

### **6.6 Concluding Remark**

The analysis showed that the suggested system achieves its goals and offers an automated network topology generating solution that is practical, effective, and easy to use. The solution minimizes the time and effort needed for network design and emulation by combining visualization, GNS3 compatibility, and LLM-based natural language processing.

Although challenges were faced regarding JSON parsing and large topology handling, these were mitigated through iterative refinement. The system thus provides a strong foundation for further enhancement, including expanded compatibility, performance optimization, and integration into broader network management platforms.

## CHAPTER 7

### CONCLUSION AND RECOMMENDATIONS

#### 7.1 Conclusion

The project effectively created a network topology generator driven by artificial intelligence (AI) that uses Large Language Models (LLMs) to convert natural language commands into exportable configurations and network diagrams. The system combines several elements, such as an easy-to-use interface, NetworkX for graph management, Matplotlib for visualization, and export features for PNG and JSON files. The created topologies were verified to be deployable for emulation and additional setup, in addition to being displayed, by validating the exported JSON files in GNS3.

By automating topology generation and guaranteeing compatibility with popular simulation tools, the system proved through methodical implementation and testing that it could speed up the network design process. Nowadays, topologies may be created, altered, and implemented by users in a fraction of the time needed for manual creation. Through incremental refinement, issues such handling huge topologies, node connectivity, and JSON schema compliance were resolved, leading to a reliable and useful system.

The project's overall goals were met, offering a solid framework that connects AI-driven design with realistic simulation. It helps network engineers save time and serves as a platform for computer networking education for students.

### 7.2 Recommendation

Even though the system worked well, there are a few things that could be done better in the future. First of all, improving the layout method would improve visualization for intricate and huge topologies while maintaining clean, uncluttered diagrams. Second, adding support for more emulation platforms, such Cisco Packet Tracer or EVE-NG, would increase the system's usefulness outside of GNS3.

Another suggestion is to reduce the requirement for iterative revisions when invalid JSON is generated by implementing real-time error notification and repair during command entry. Including a template library for popular topologies (such data center, WAN, or enterprise LAN configurations) would also enable users to create conventional networks more rapidly.

Furthermore, the system might become a more comprehensive network design and analysis tool if it were expanded to enable simulation and traffic testing straight from the interface. Lastly, investigating the usage of local language models or optimized LLMs would increase accuracy, decrease running costs, and lessen reliance on external APIs.

By implementing these suggestions, the system may develop into a more potent and adaptable platform that connects thorough network modelling with AI-assisted design.

## REFERENCES

- [1] P. Gil et al., "Computer networks virtualization with GNS3," in *Proc. IEEE FIE*, 2014, pp. 1–8, doi: 10.1109/FIE.2014.704543.
- [2] P. Gil, F. A. Candelas, C. A. Jara, G. J. Garcia, and F. Torres, "Web-based OERs in Computer Networks," *International Journal of Engineering Education*, vol. 29, no. 6, pp. 1360–1365, Dec. 2013.
- [3] J. McNamara et al., "NLP Powered Intent Based Network Management for Private 5G Networks," *IEEE Access*, vol. 11, pp. 36642–36657, 2023, doi: 10.1109/ACCESS.2023.3265894.
- [4] P. Uyyala and D. C. Yadav, "The role of AI in the development of next-generation networking systems," *Int. J. Anal. Exp. Modal Anal.*, vol. XV, no. V, pp. 1048–1059, May 2023.
- [5] G. Santoso, J. Setiawan, and A. Sulaiman, "Development of OpenAI API-Based Chatbot to Improve User Interaction on the JBMS Website," *G-Tech: Jurnal Teknologi Terapan*, vol. 7, no. 4, pp. 1606–1615, Oct. 2023.
- [6] D. W. Lamb, *Framework for Industrial Control System Honeypot Network Traffic Generation*. Master's thesis, Iowa State University, 2021. [Online]. Available: <https://doi.org/10.31274/etd-20210625-6>
- [7] F. Stadtmann, H. G. Wassertheurer, and A. Rasheed, *Demonstration of a Standalone, Descriptive, and Predictive Digital Twin of a Floating Offshore Wind Turbine*, Norwegian University of Science and Technology, 2023. [Online]. Available: <https://arxiv.org/abs/2304.01093>
- [8] V. C. Tapia and C. M. Gaona, "Automation of Software Development Stages with the OpenAI API," *Computers, Materials & Continua*, vol. 49, pp. 1–17, 2025. doi: 10.32604/csse.2024.056979.
- [9] "Introduction to the Python Client," Gradio, Accessed Apr. 29, 2025. [Online]. Available: <https://www.gradio.app/docs/python-client/introduction>
- [10] S. Bubeck, V. Chandrasekaran, R. Eldan, J. Gehrke, E. Horvitz, E. Kamar, P. Lee, Y. Li, S. Lundberg, H. Nori, M. Palangi, H. Sellam, C. Zhang, and Y. Zhang, *Sparks of Artificial General Intelligence: Early Experiments with GPT-4*, arXiv preprint arXiv:2303.12712, 2023. [Online]. Available: <https://arxiv.org/abs/2303.12712>

## Code Sample

```
def export_diagram():
    if diagram_manager.G.nodes:
        import io
        import tempfile
        from PIL import Image
        import matplotlib.pyplot as plt

        # Get the current figure from matplotlib
        fig = diagram_manager.render_diagram(export=False)

        # Save figure to a bytes buffer first
        buf = io.BytesIO()
        fig.savefig(buf, format='png', bbox_inches='tight', dpi=300)
        buf.seek(0)

        img = Image.open(buf)


        # Create a temporary file with .png extension
        temp_file = tempfile.NamedTemporaryFile(delete=False,
suffix='.png')
        temp_file_path = temp_file.name

        img.save(temp_file_path)

        plt.close(fig)

        return temp_file_path
    return None
```

## Poster



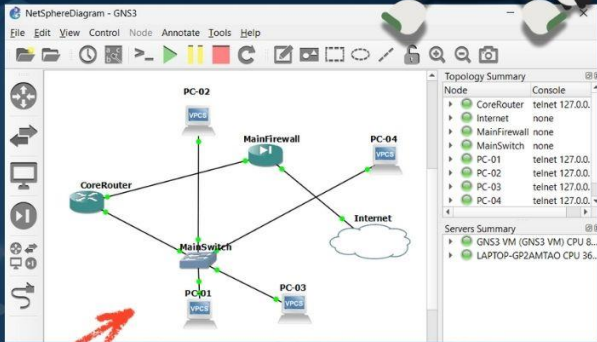
**UTAR**  
UNIVERSITI TUNKU ABDUL RAHMAN

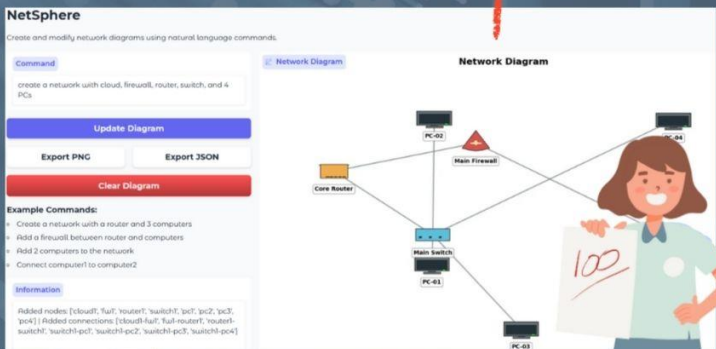
PROJECT DEVELOPER: FOO KAR YENG  
PROJECT SUPERVISOR: DR AUN YICHIEI

# SECURING DATA CENTER WITH DIGITAL TWINS

## INTRODUCTION

Modern data centers need fast, secure, and scalable network design. Traditional tools like GNS3 require manual setup, which is slow and error-prone. This project introduces an AI-powered network diagram generator using OpenAI's API (DeepSeek:R1) that creates and edits topologies from natural language commands, with export to PNG and JSON for seamless GNS3 integration.





## OBJECTIVE

- Develop an AI-powered network diagram generator using natural language.
- Enable dynamic add/remove of components via text commands.
- Provide real-time visualization of topologies.
- Support export to PNG and JSON for seamless GNS3 integration.

## PROPOSED METHOD

- Use DeepSeek:R1 (OpenAI-compatible) to interpret natural language commands.
- Process input with regex and structured JSON formatting.
- Generate topologies using NetworkX and visualize with Matplotlib.
- Build an interactive Gradio-based web UI for user interaction.
- Support export to PNG for documentation and JSON for seamless GNS3 import.

## WHY THIS SYSTEM IS BETTER THAN OTHER AI TOOLS?

- Text-to-Diagram Capability: No prior networking or design software knowledge required.
- Real-Time Modification: Users can add or delete components dynamically, unlike static tools.
- High Accuracy: Achieves 92.3% accuracy in converting text to valid diagrams.
- Speed: Reduces design time by over 10× compared to traditional tools like GNS3.
- Custom Visualization: Offers advanced layered layouts,