COMP2043.GRP Interim Group Report

Logic Diagram Generation Software

Team202419

Supervisor
Dr. Huan Jin

Team members
Youyao Gao (20516639/ scyyg6)
Jason Lymand WIDJAYA (20512013 / scyjw13)
Shu Qi Lee (20611680 / hcysl7)
Tongchang Liu (20513017 / scytl5)
Yuhang Wu (20513290 / scyyw20)
Ziqi Li (20514283 / scyzl13)

Submitted on 5 December 2024

School of Computer Science University of Nottingham Ningbo China

Contents

1 Introduction

This project aims to develop a software application that enables users to input extensive textual logic descriptions and generate corresponding logic diagrams by recognizing the logical relationships within the text. In the contemporary software engineering landscape, the ability to transform unstructured textual information into structured visual representations is invaluable. This project addresses the need for a user-friendly application that simplifies the process of logic diagram generation from verbose text, thereby enhancing comprehension and facilitating better communication of complex ideas.

1.1 Current System

Current tools for creating logic diagrams, such as Microsoft Visio, Visual Paradigm, and Lucidchart, require users to input and organize elements manually. Thus making the process time-consuming and prone to error. Some systems, like PlantUML, can generate logic diagrams automatically, but users might need technical skills, making it unsuitable for non-technical audiences. In addition, no software currently available on the market can generate logic diagrams directly from natural language descriptions, leaving a gap for easy and automated logic diagram generation solutions. Current technologies, including LLMs like ChatGPT-4 and visualization software like Graphiz, cannot fully automate logic diagram generation from natural language description. As a result, users will need lots of time and effort to create natural language descriptions into logic diagrams. These limitations in the current system highlight the need for a tool that integrates NLP and visualization to improve the accessibility, efficiency, and accuracy of users in various industries.

1.2 Proposed Solution

To address the drawbacks of existing software capabilities to generate logic diagrams from the textual description, we propose to develop the Logic Diagram Generation Software called **ChartGPT**. This software will utilize NLP methodologies with advanced graphic visualization tools to automate the process of logic diagram generation. It is suitable for both technical and non-technical users to eliminate the pain point of manual logic diagram generation. The key features of the proposed solution are as follows:

- **Develop an NLP Algorithm:** Design and implement an NLP algorithm that has the capability of accurately interpreting and analyzing the semantic structure of input textual description, recognizing logical relationships between words, and identifying key logic diagram components such as entity, relationship, attribute, and cardinality.
- **Automate logic diagram Generation:** Integrate NLP algorithm to automate the process of logic diagram generation that follows standard logic diagram conventions and is easily understood by end-users.

- **User-Friendly Interface:** Create a responsive and user-friendly interface that enables users to input textual logic descriptions effortlessly and view the generated logic diagrams in real time.
- **User Testing:** Implement a user testing phase to gather feedback on usability and functionality. Allowing developers to improve the software based on user feedback over time.

2 Background and Relevant Research

2.1 Background

Visual graphic tools, such as logic diagrams and diagrams, are estimated to play an essential role in over 60% of business and technical workflows globally (Verified Market Reports, 2023). Despite this demand, most existing tools available for diagram generation deliberately rely on manual effort, which could consume excessive time while being susceptible to human error. For example, software like Microsoft Visio or Lucidchart requires users to manually define each step, connecting each element and ensuring each logical accuracy (Team, 2023). This often leads to inefficiencies in very complex projects. While some automated tools exist for creating charts from code, they are mostly tailored for users with technical skills and cannot process natural language description inputs (Bostock, 2023). These drawbacks present significant barriers to users from non-technical backgrounds who need accessible and intuitive solutions for logic diagram creation.

Logic diagrams are essential tools for visualizing complex processes across industries, using standardized symbols to clearly depict logical relationships. They help users understand workflows, make decisions, and identify system flaws by breaking down complexities into manageable components. Widely used in business modeling, system design, and decision-making, they streamline communication and enhance clarity. As shown in Figure 1, Figure 2, and Figure 3, there are 3 common types of logic diagrams, which are temporal logic diagram, parallel logic diagram and Judging logic diagram. These types of logic diagrams may serve as the final output format of our software.

With the increasing dependency on NLP across businesses, having tools that integrate NLP to automate time-consuming tasks has become increasingly common. LLMs like OpenAI's ChatGPT have been broadly used for comprehending and summarizing data (OpenAI, 2023). However, current NLP models and text-to-diagram tools focus on general tasks, such as converting structured data into visualizations, and do not have the capabilities to specifically address the generation of logical logic diagrams (Unknown, 2021). This barrier represents evidence of pain points in sectors such as business and education, where users need tools that can transform natural language descriptions into logic diagrams.

The absence of dedicated software for creating logic diagrams directly from natural language descriptions on the market highlights the need for an innovative

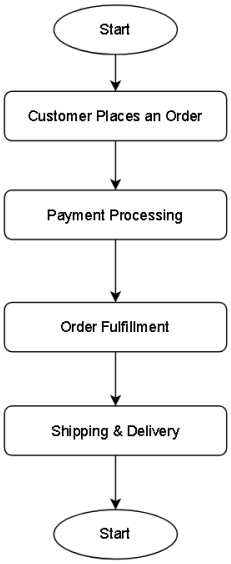

Figure 1: Temporal Logic Diagram

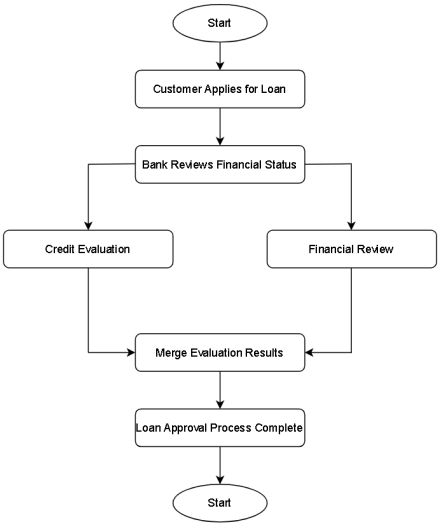

Figure 2: Parallel Logic Diagram

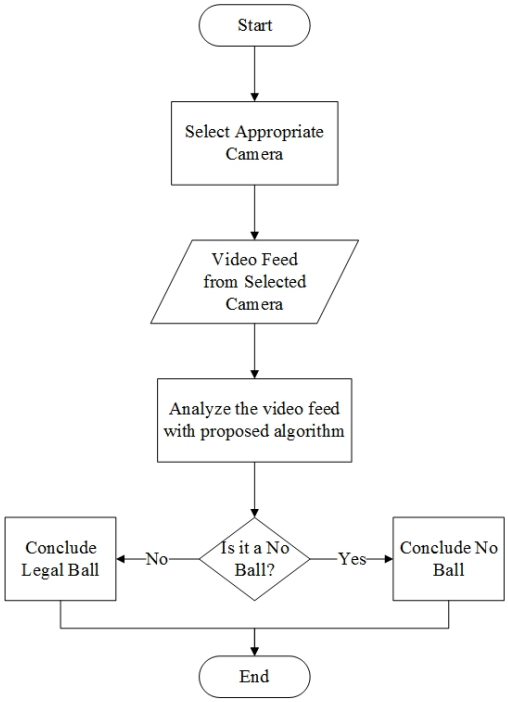

Figure 3: Judging Logic Diagram
(Chowdhury et al., 2016)

solution (Unknown, 2021). Using NLP algorithms, our Logic Diagram Generation Software aims to solve this pain point, enabling technical and non-technical users to input natural language descriptions and receive automatically generated structured logic diagrams (Gamage, 2024). This solution will improve efficiency and ensure accuracy, making the software valuable across various industries and applications (Verified Market Reports, 2023).

2.2 Motivation

The growing reliance on logic diagram creation in some industries, such as business, software development, and education, creates the need for software that could simplify and automate the process of logic diagram generation (Verified Market Reports, 2023). Several existing software like Microsoft Visio, Visual Paradigm, and Lucidchart require users to input and organize manually, resulting in substantial time consumption and increased likelihood of inaccuracies (Team, 2023). LucidChart, an online charting tool, allows users to create logic diagrams and provides some automation based on the logical relationships the user requires. However, it still relies on manual input and adjustment by the user to complete the logic diagram. Furthermore, the application's functionality and design are unfriendly to non-technical users for more complex logic diagrams. Despite some efforts that have been made to automate the logic diagram creation, it still requires technical skills to create it (Unknown, 2021), which still cannot accommodate non-technical users. This lack of accessible solutions for non-technical users creates a need for fully automated logic diagram generation software, which can generate correct logic diagrams according to the natural language description given by the users (Tian et al., 2024).

The global logic diagram software market reflects this demand, with an estimated CAGR of 8.3% from 2024 to 2030 (Verified Market Reports, 2023). Despite this exponential and sustainable growth, there has yet to be an attempt to specifically automate the generation of logic diagrams from natural language description by utilizing NLP. Available software will need structured input such as code or require the user to create the logic diagrams manually. This software, **ChartGPT**, aims to serve the gap to make automated logic diagram generations for technical and non-technical users by using NLP to extract logical relationships and translate them into an accurate and structured logic diagram (Tian et al., 2024).

ChartGPT software enables users to input a paragraph containing logical relationships. It automatically analyzes the text using NLP techniques, extracting key terms to describe entities and identify the logical connections. By combining the power of NLP with D3.js for visualization, ChartGPT first performs a comprehensive paragraph analysis to ensure accurate identification of logical relationships. Based on this analysis, the system generates a logical diagram following the identified relationships. This integration of NLP and D3.js ensures that the generated logic diagrams are accurate, highly interactive, and visually intuitive. Additionally, users can modify and download the generated logic diagram for immediate use, making it a user-friendly tool that simplifies complex logic visualization.

ChartGPT will offer users a significant benefit by automating the logic diagram generation process, reducing the time and effort needed compared to creating a logic diagram manually. This also minimizes errors and enhances productivity, especially for professionals with complex logic documentation. Non-technical users like teachers and business analysts can also generate logic diagrams without requiring technical expertise or being familiar with complex tools (Verified Market

Reports, 2023).

ChartGPT is highly flexible software and can be used across various sectors, such as:

- **Software Development:** For system design, debugging, and documentation.
- **Business and Finance:** For workflow modeling, operational planning, and decision-making processes.
- **Research and Analysis:** For visualizing and presenting discovery clearly.
- **Education:** For visualizing concepts in teaching and learning materials (Verified Market Reports, 2023).

2.3 Related Work

This section provides an overview of fundamental research and existing software related to the study. This section reviews recent advancements in logic diagram generation and NLP technologies for automated logic diagram generation and compares existing software solutions for logic diagram creation.

2.3.1 Literature Review

In recent years, data visualization and NLP technologies have garnered extensive attention in research and applications across various fields. By reviewing a large amount of literature, we have gained insights into cutting-edge technologies to lay the groundwork for our software development.

Research on NLP

Because of the difficulties of natural language processing, Zeng explored techniques for converting natural language to ROM (Zeng, 2008). ROM stands for the Recursive Object Model, which is a novel graphical language developed through mathematical derivation designed to represent linguistic elements in natural language, particularly the structure of technical English. There are five symbols to describe objects and their relationships: Object is the most basic unit and can represent any entity or concept. Compound Objects represent complex concepts like phrases or paragraphs. A Constraint Relationship is a limiting or descriptive link between objects. Connection Relationship represents the connects two objects without a constraint. A Predicate Relationship describes an action or state, typically involving verbs. The existence of ROM makes the processing of natural language easy.

Research on Text-to-visualization

Regarding the research on text-to-visualization, Chen explored some techniques for generating visualizations from natural language queries in two of their studies (Chen et al., 2022a, 2022b) . Through a type-directed visualization synthesis

framework, these studies demonstrated how to generate complex charts automatically from users' natural language input, which reduces the technical barrier for users while enhancing the operability of data visualization. Tian further advanced this field by proposing a method based on LLMs (such as ChatGPT) to generate high-quality visualizations from natural language (Tian et al., 2024). This approach leverages the semantic understanding and contextual modeling capabilities of language models, significantly improving the accuracy and efficiency of chart generation.

Regarding visualization research, Bostock introduced D3.js in his study, a widely used visualization framework based on data-driven documents (Bostock, 2023). Supporting interactive chart design, D3.js offers developers substantial flexibility while lowering the complexity of implementing sophisticated data visualizations. Additionally, Ellson et al. (2001) proposed Graphviz, a software tool that visualizes graph structures, providing a standardized method for handling graphs involving nodes and edges. Meanwhile, Community (2023) explored the applications of PlantUML, showcasing its advantages in quickly generating UML diagrams. Furthermore, Team (2023) compared the pros and cons of manual and automated visualization tools across different application scenarios. They suggested that automated tools, such as Lucidchart, significantly simplify process design and exhibit highly efficient workflow management, especially in collaborative environments.

Research on Analogous approach

Due to the complexity of creating Systems Modeling Language (SysML) diagrams, which often require significant manual input, the process is time-consuming and prone to errors. Zhong put forward an approach for SysML diagrams. The method employs NLP techniques to extract information from unstructured natural language texts, such as specifications, manuals, technical reports, and maintenance reports and then generates system modelling diagrams (Zhong et al., 2023). This approach has two key contributions. Firstly, it provides a flexible method for handling diverse sources without imposing strict format constraints on the input documents. Secondly, by integrating NLP techniques with lexical databases, the method enhances the generation of system models.

2.3.2 Existing Software

Existing software, such as Microsoft Visio, Lucidchart, and Visual Paradigm, is often used for creating logic diagrams and diagrams (Team, 2023). However, this software still depends on manual input, making the process tedious and vulnerable to potential errors, especially for creating complex charts. Automated solutions like D3.js and Graphviz provide more flexibility for developers, allowing them to generate logic diagrams automatically (Bostock, 2023; Ellson et al., 2001) but still require structured input. PlantUML, another existing software that is capable of generating logic diagrams, also relies heavily on specific syntax, limiting accessibility for non-technical users (Community, 2023). Despite these advancements, there is

no existing software that fully automates the generation of charts directly from natural language descriptions.

3 Software Requirements

3.1 Requirements Elicitation

The project needs to understand the requirements of stakeholders who know their expectations well. Insights about this project were gathered through interviews to ensure the project's function matched the user expectations and intended impact.

3.2 Requirements Specification

3.2.1 Functional Requirements

The first functional requirement is that users can provide detailed textual descriptions, which the system prints out a logic diagram. There are three types of logic diagrams: Temporal logic diagram, parallel logic diagram, and judging logic diagram. The shapes included are ovals, rectangles, parallelograms, and diamonds. Among these, the oval represents the start or end node. The rectangle indicates a specific step or operation, that is, the processing procedure; the parallelogram is used to denote data input or output operations, and the diamond is used to represent decision statements. The system must ensure that the generated logic diagram is accurate and aligns with the user's input text. This means we need to improve the algorithm's accuracy to over 90%.

Secondly, the system enables users to customize the generated diagrams, allowing them to review and modify elements as needed. Users can click on each node to view detailed information, such as the node's name and its associated input text. If any node's content does not align with the original logical description, users can directly edit the node's text or redefine its logical expression.

Thirdly, to enhance usability, the platform allows interface customization, including options like dark mode, catering to diverse user preferences. Once finalized, the diagrams can be exported in multiple formats, such as PNG, JPEG, and PDF, ensuring seamless integration into various professional documents, presentations, and reports, another functional requirement. This intuitive, user-centric design seamlessly integrates automation with customization, providing a powerful and efficient solution for generating and managing logic diagrams.

The fourth functional requirement is login and retaining user history. While this is a standard feature in most systems, we have decided not to implement it initially. Instead, we will add this functionality later when the user base grows or accessing historical records becomes a essential need.

3.2.2 Non-Functional Requirements

The system is meticulously designed with a focus on usability, performance, and reliability, aiming to provide users with a smooth and efficient experience. Its intuitive interface ensures that users can easily navigate and utilize the tool, even without professional expertise or prior training.

Firstly, the system must support high concurrency, remaining stable and functional when at least 10 users are using it simultaneously, without crashes or significant performance degradation. Additionally, the system's performance must ensure efficient response times, processing textual inputs of up to 1000 characters within a maximum of 2 seconds.

3.3 Requirements Validation

Steps are taken to ensure the supervisor fully understands the development plan and vision for the project. A comprehensive document detailing all software requirements is drafted and provided to the supervisor for a thorough review. After the assessment, a follow-up meeting is held to discuss the details, during which the supervisor confirms that the requirements are correct and aligned with the project objectives. With this confirmation, the confidence that the software meets shareholder expectations and the ability to deliver the intended value increases.

4 Software Design

4.1 UML Diagram

The Use Case Diagram, shown as Figure 4, illustrates the key functionalities of the software and the interactions between the system and users. The User can input text, view, download, and edit the generated logic diagram. The system preprocesses the text, identifies logical relationships, and generates a logic diagram, supporting extensions for editing and downloading logic diagrams.

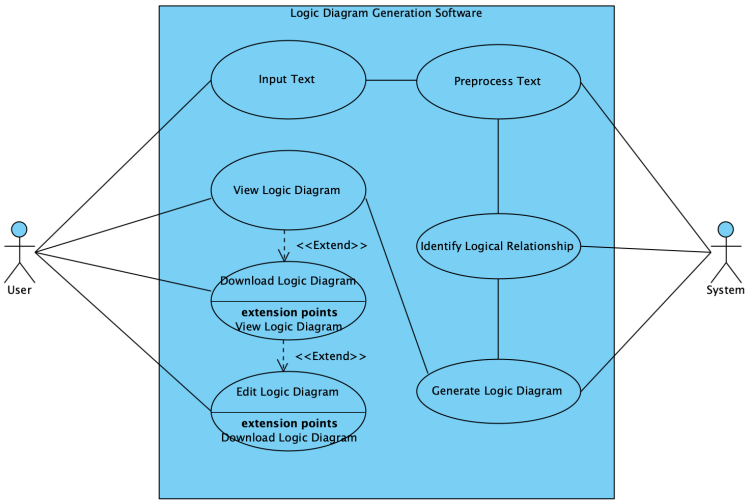

Figure 4: Use Case Diagram

The Sequence Diagram, which is shown in Figure 5, demonstrates the interaction sequence between the components of the system for generating a logic diagram. The User inputs text via the UI, which is preprocessed to clean the data. The NLP Module identifies logical constructs, and the logic diagram Generation Module creates a logic diagram. The Visualization Library renders it, and the user reviews or modifies it if needed.

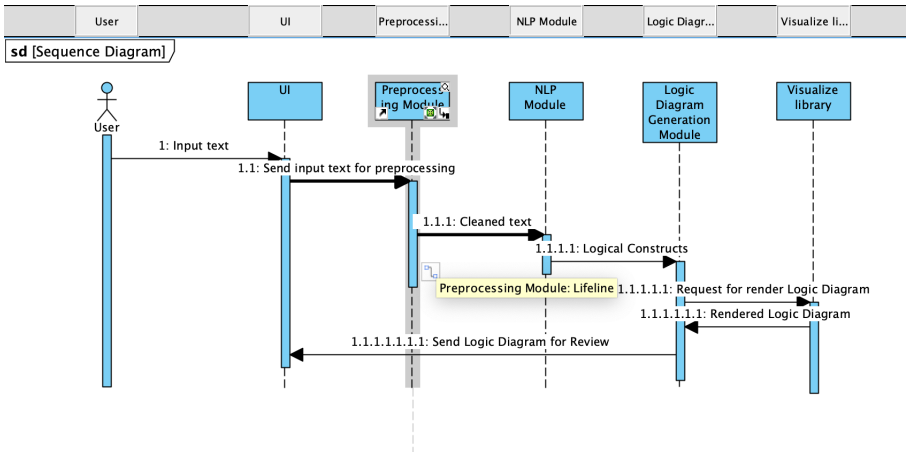

Figure 5: Sequence Diagram

The Activity Diagram, Figure 6, describes the step-by-step workflow of the software. The User provides text, which is cleaned and analyzed for logical relationships. A logic diagram is generated and displayed. The User decides whether to download and edit it or finalize the process, ensuring flexibility and refinement.

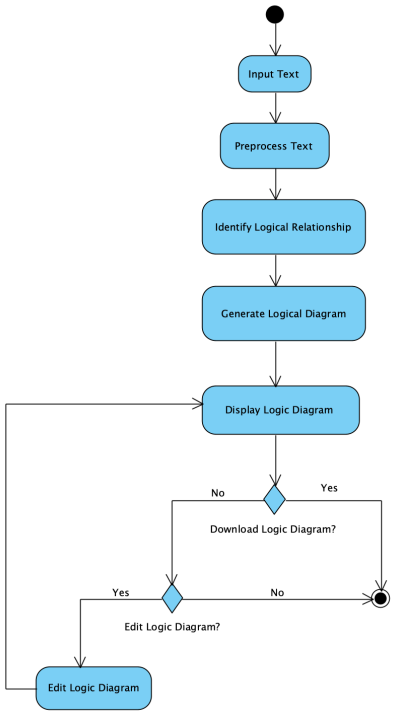

Figure 6: Activity Diagram

4.2 Prototypes

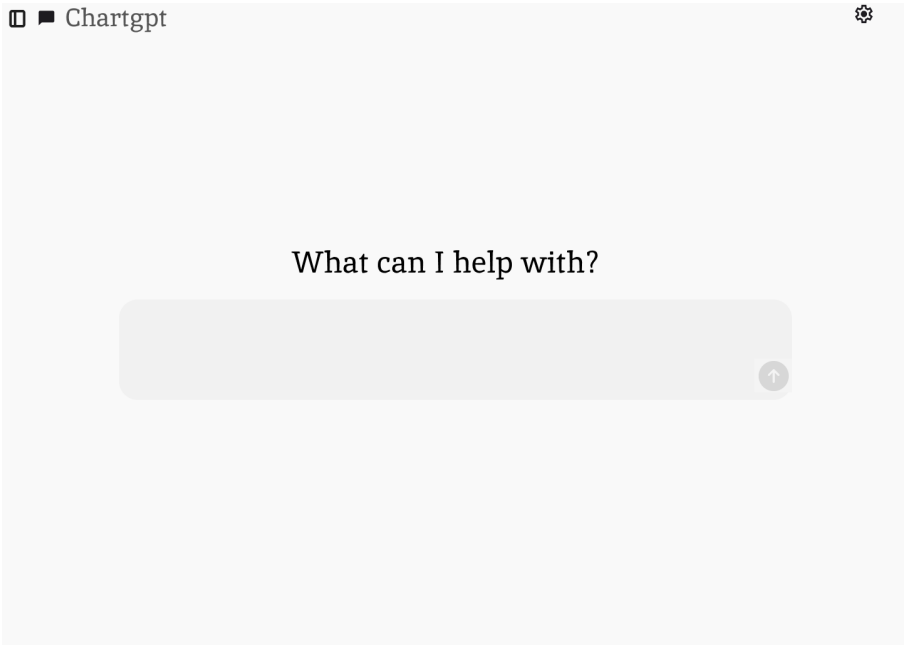

Figure 7: Prototype - 1

Figure 7 shows the main menu of our app. It contains three functions: write text, submit the text, and set. Users can put their text into the dialog box and then click the submit button on the left; the system will analyze the text.

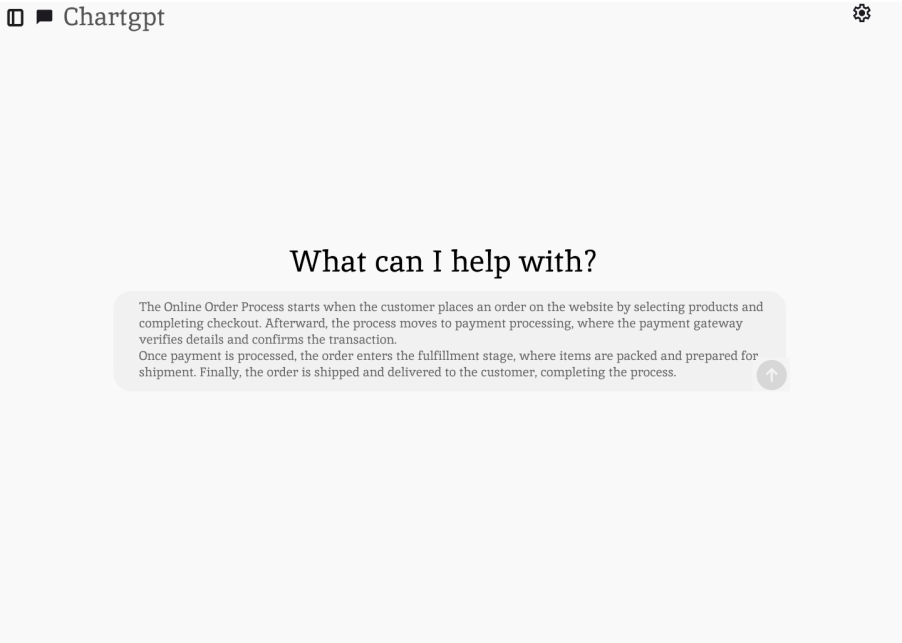

Figure 8: Prototype - 2

Figure 8 shows the user input an example text. When the user clicks the submit button, it will go to Figure 9. It shows the app has analyzed the text and then generates a logic diagram.

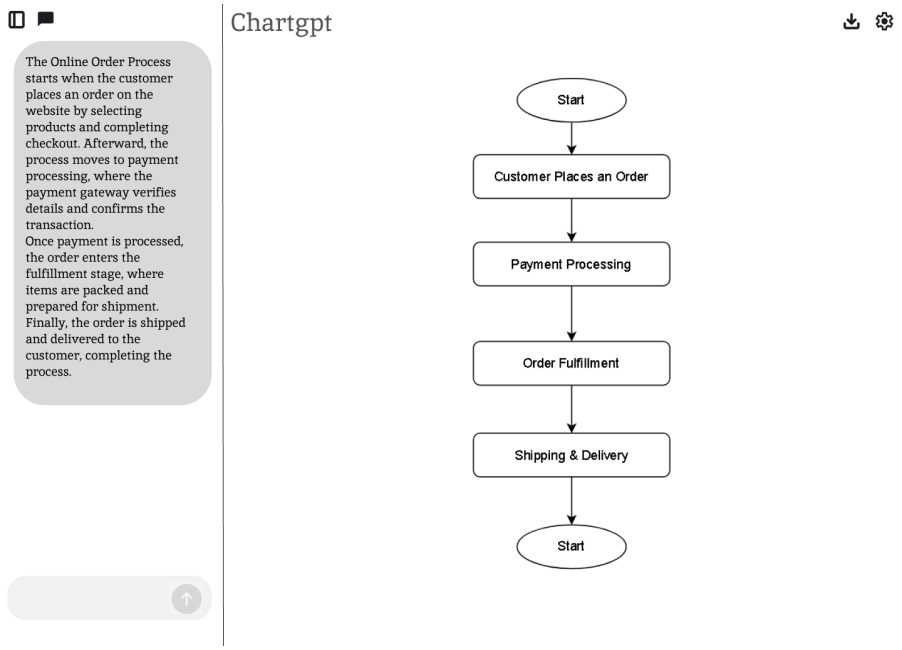

Figure 9: Prototype - 3

When users click the downloading button, the software will shows Figure 10, which shows the downloading pages, users can check the address of the downloading diagram and delete it.

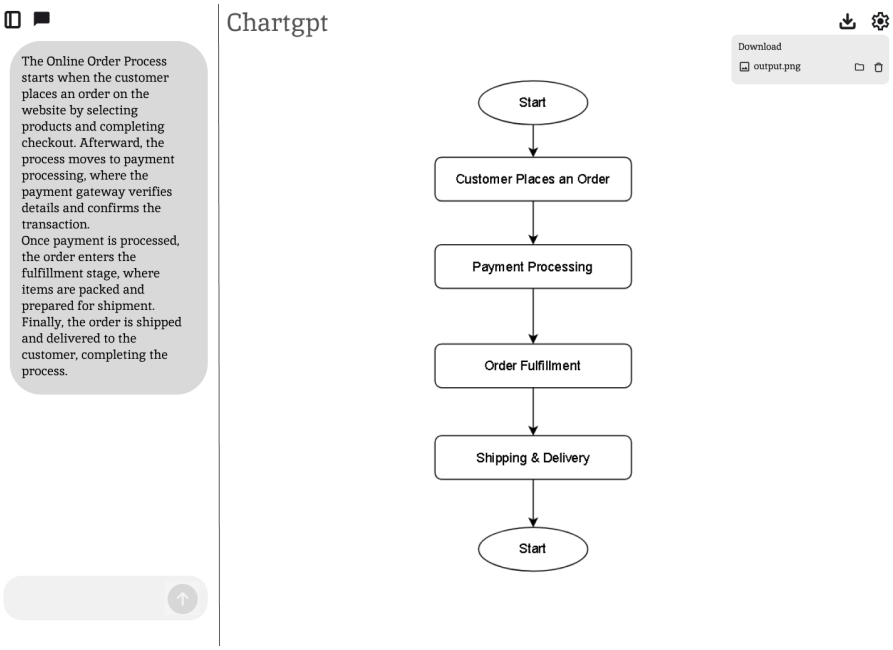

Figure 10: Prototype - 4

By clicking the second button on the head, which is the changing button, the software will demonstrate Figure 11 or Figure 12; users can click on the diagram's relationships and nodes to change by their selves.

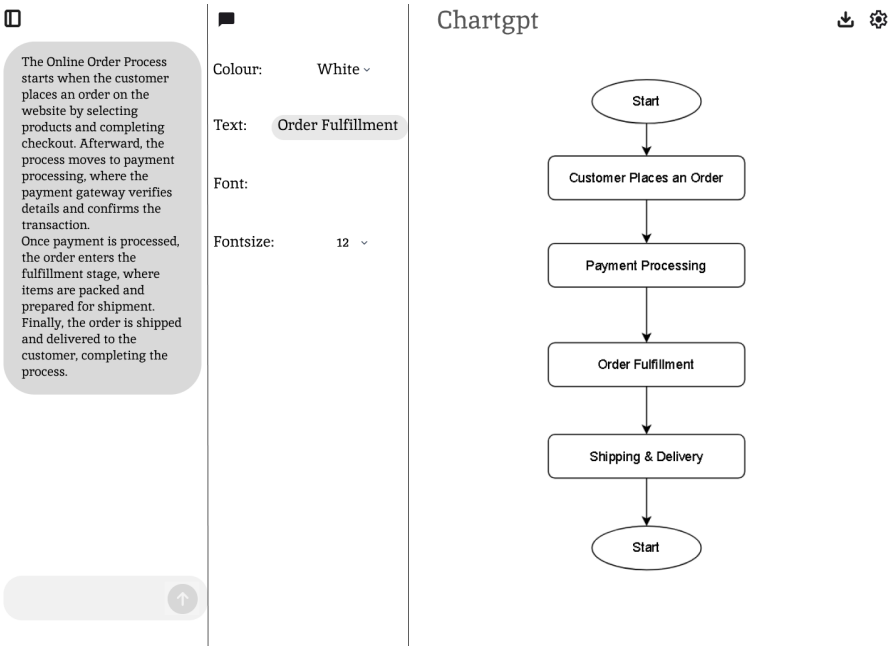

Figure 11: Prototype - 5

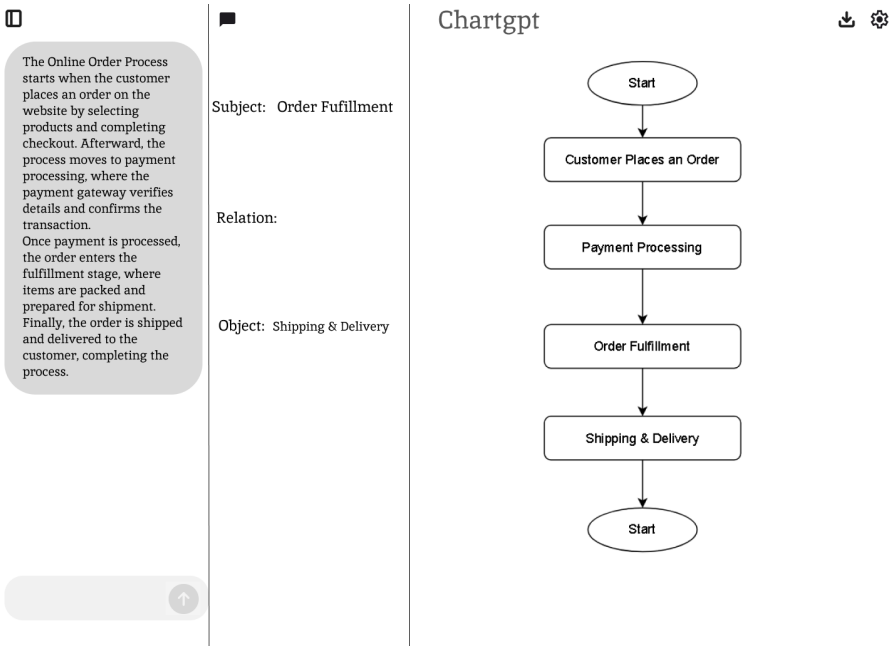

Figure 12: Prototype - 6

When users click the setting button, it will shows Figure 13. The software allows users to change the theme of it.

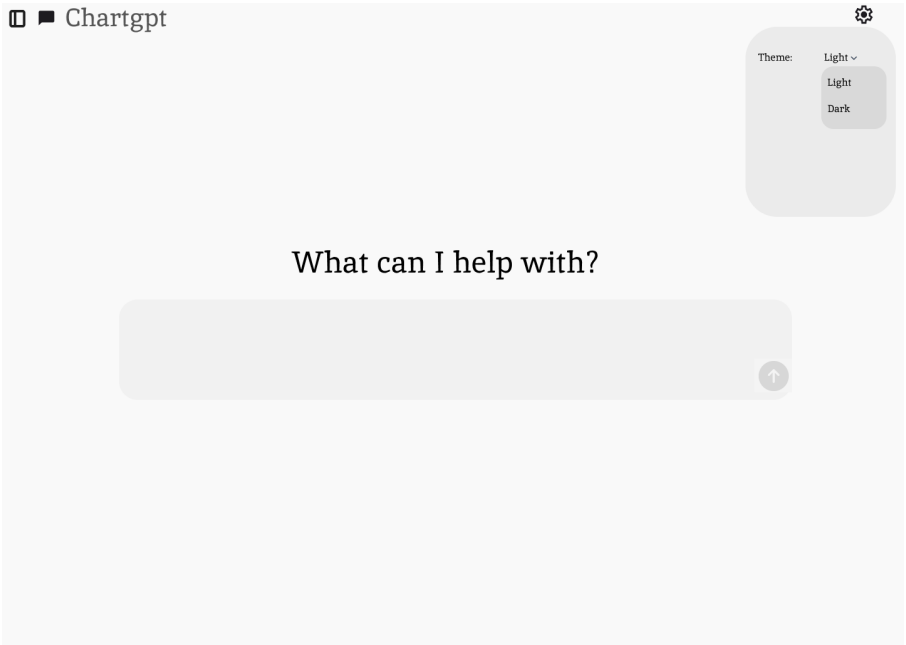

Figure 13: Prototype - 7

4.3 Design Decisions

These are the design decisions made so far for the project. The aim is to make the website more robust and scalable while ensuring it is user-friendly and intuitive.

4.3.1 Server

Currently, no server is used because the software is a basic standalone website for generating logic diagrams based on user input text.

However, if the website is published in the future, a backend server will certainly be needed to store the vast amount of user and logic diagram data. Since the project scale is still small, Flask, a lightweight API framework, will likely be used to process NLP inputs and generate JSON files for the D3.js logic diagram generation.

For data storage, MongoDB is an idea for the project as it can store logic diagram data in JSON format. In addition, MongoDB is scalable and flexible, allowing it to handle increasing amounts of user and logic diagram data as the project grows.

4.3.2 Front-end

The website uses HTML, CSS, and JavaScript to make it more interactive. D3.js is also essential in this project, as it enables the generation of logic diagrams after receiving the JSON input from the user.

A robust front-end framework like React.js can enhance the user experience and streamline development if the project grows. The advantage of using React.js is that it makes the website much more scalable, modular, and easier to maintain in the future. React.js will also allow for the efficient state management and integration of new features as needed

In addition to CSS, Bootstrap can be used to customize the website's appearance as it provides us with a more responsive design.

4.3.3 Back-end

Python is the primary programming language used to run the project's NLP. Its versatility, ease of use, and extensive libraries of spaCy and NLTK will make it an excellent choice for processing user input and generating structured data for logic diagrams.

The first step in the data processing pipeline is tokenization. This step will break down raw input text into smaller units called tokens. These tokens can be individual words, punctuation marks, or other meaningful text components. For example, the sentence "The quick brown fox jumps over the lazy dog" will be tokenized as: "The", "quick", "brown", "fox", "jump", "pass", "the", "lazy", and "dog". Tokenization is an important step because it will isolate the main components of the sentence, making it easier for the software to interpret every word separately and understand its importance in the sentence.

The second step will be removing the stop words. Stop words are common words such as "the", "is", "on", and "and" that appear frequently in the text but do not have a noteworthy meaning to the overall sentence. These words are often filtered out during text processing to focus on more important words. Removing stop words also allows the system to reduce the amount of unimportant data and focus on keywords that convey the core meaning of the sentence. For example, after the stop words are removed. The sentence "The quick brown fox jumped over the lazy dog" becomes: "quick", "brown", "fox", "jump", "lazy", and "dog".

The algorithm will use part-of-speech (POS) labeling in the third step. POS labeling will identify the grammar and assign it to each token for every word in the sentence. For example, "fox" will be categorized as a noun, and "jump" will be categorized as a verb. This categorizing will help the system identify important elements, such as the sentence's subject (noun) and action (verb). POS tagging is essential because it provides insight into the grammatical structure of a sentence, which is necessary to understand how different parts of a sentence relate to each other and to extract logical relationships.

To determine the logical relationships in the input text, the software will use a thesaurus of connectives, which will categorize words and phrases based on the relationships that they represent, shown in Listing 1, such as temporal, conditional, casual, parallel, contrastive, and alternative connectives. For example, temporal connectives like "during", "at the same time" and "then" help establish the sequence of the events, while conditional connectives like "as long as" and "if" represent the relations between actions or conditions. The full list of connectives used in this project is shown in Listing 1.

After processing the dataset, the next step is to split the data into training and testing sets. This step is important for evaluating the model's performance and ability.

The training set will be used to train the model to achieve higher accuracy. The training set includes most of the processed data fed into the model, allowing it to recognize patterns, relationships, and grammatical structures in the information. In text processing, the training set contains labeled examples whose input sentences are tokenized, stop words removed, and POS tagged according to the correct logical relationships or structures that the model wants to learn. The model modifies its parameters based on its errors during training, gradually improving its predictions.

After the model is trained, it will be evaluated using the testing dataset. A test dataset consists of a subset of data the model has not encountered during the training phase. This dataset will evaluate the model's ability to handle new and unseen input. In text processing, a test dataset consists of sentences treated like the training data (with tokenization, removal of stop words, and part-of-speech tagging applied). However, the logical relationships or structures are hidden from the model, which is tasked with predicting these logical relationships based on the patterns learned during training. The model's efficacy will be assessed by contrasting

its predictions with the actual outcomes in the test dataset.

By splitting the data into training and testing sets, it can evaluate its capability to handle data different from the training set. Making sure that the model can accurately predict outcomes for new and unfamiliar inputs. This step is crucial for assessing the model's performance in practical situations where the input data may differ from the input data during training.

Traversing the words in the sentence to extract the connectives that represent the logical relation and determine the logical relation of the context connected by the word is shown in Listing 1. To structure the data, the project decides to use JSON FORMAT to extract the content and logical relationships of sentences, as shown in Listing 2. JSON's lightweight and human-readable nature is very suitable for representing hierarchical and relational data, which is expected when creating logic diagrams. A concept called tight before and tight after is introduced here. If there is a temporal relation between two objects, it shows the basic relationship in JSON. For more complex logical relationships like parallel actions or judgment, the project considers setting up special forms in JSON to describe their relationships. Combining tokenization, stop word removal, and POS tagging techniques, our software ensures accurate text processing to generate structured JSON data from unstructured natural language descriptions.

The project will use Data-Driven JavaScript (D3.js) to visualize the results of the data structure. Data-driven JavaScript offers robust tools for rendering hierarchical data and tree-like structured data and can generate various shapes like circles, rectangles, and polygons. These features make it suitable for creating logic diagrams. After the user enters descriptive text into our website's input field, JavaScript could fetch the input and send it to our back-end using an HTTP POST request via FLASK, a lightweight Python web framework. Then, the logic diagram in JSON FORMAT is generated using back-end NLP technology, as mentioned above. Our NLP technology ensures data is formatted into a standardized JSON format representing the logic diagram. After that, Flask will respond to this JSON data to the front end, allowing JavaScript to use this JSON data to render the logic diagram and display it on the website through Data-Driven JavaScript. D3.js processes the data using a built-in tree and hierarchy function. The tree function is used to compute the layout of the nodes and links that can organize the logic diagram into a tree-like structured format. The hierarchy function is also a key part because it can structure the JSON data into a parent-child relationship, which is suitable for generating the logic diagram that includes logical relations. Combining both tree and hierarchy functions and mapping each node and line ensures a clear logic diagram generation.

```
{
temporal_connectives = [
    'first', 'second', 'third', 'next', 'then', 'after that', 'subsequently',
    'finally', 'eventually', 'at last', 'in the end', 'when', 'before', 'after',
    'since', 'until', 'as soon as', 'once', 'meanwhile', 'while', 'simultaneously',
    'concurrently', 'then', 'immediately', 'later', 'prior to', 'earlier', 'henceforth',
    'thereafter', 'in the meantime', 'ongoing', 'up until', 'following', 'shortly after',
    'at the same time', 'during'
]
conditional_connectives = [
    'if', 'unless', 'provided that', 'assuming that', 'as long as', 'in case', 'on condition that
    'supposing that', 'only if', 'in the event that', 'contingent upon'
]
causal_connectives = [
    'because', 'since', 'therefore', 'thus', 'as a result', 'consequently', 'hence', 'due to',
    'owing to', 'resulting from', 'on account of', 'for this reason', 'in view of', 'thanks to',
]
parallel_connectives = [
    'and', 'also', 'as well as', 'in addition', 'moreover', 'furthermore', 'besides',
    'along with', 'together with', 'coupled with', 'what's more', 'not only... but also', 'plus'
]
contrastive_connectives = [
    'but', 'however', 'nevertheless', 'although', 'though', 'even though', 'despite', 'yet',
    'whereas', 'nonetheless', 'on the other hand', 'on the contrary', 'even so', 'albeit',
    'notwithstanding', 'conversely', 'regardless'
]
alternative_connectives = [
    'else', 'otherwise', 'or', 'alternatively', 'or else', 'as an alternative', 'in place of',
    'instead of', 'rather than'
]
```

Listing 1: Example Thesaurus Construction

```json
{
  "steps": [
    {
      "step": "start process user enters credentials logs system",
      "relation": "and",
      "relation_type": "parallel"
    },
    {
      "condition_block": {
        "condition": "user administrator privileges access admin dashboard",
        "if_true": [],
        "if_false": []
      }
    },
    {
      "step": "user choose view reports update settings",
      "relation": "or",
      "relation_type": "alternative"
    },
    {
      "condition_block": {
        "condition": "choose view reports system generates report summary",
        "if_true": [
          {
            "step": "opt update settings prompted save changes editing"
          }
        ],
        "if_false": []
      }
    },
    {
      "step": "user logs end session",
      "relation": "finally",
      "relation_type": "temporal"
    }
  ]
}
```

Listing 2: JSON Format

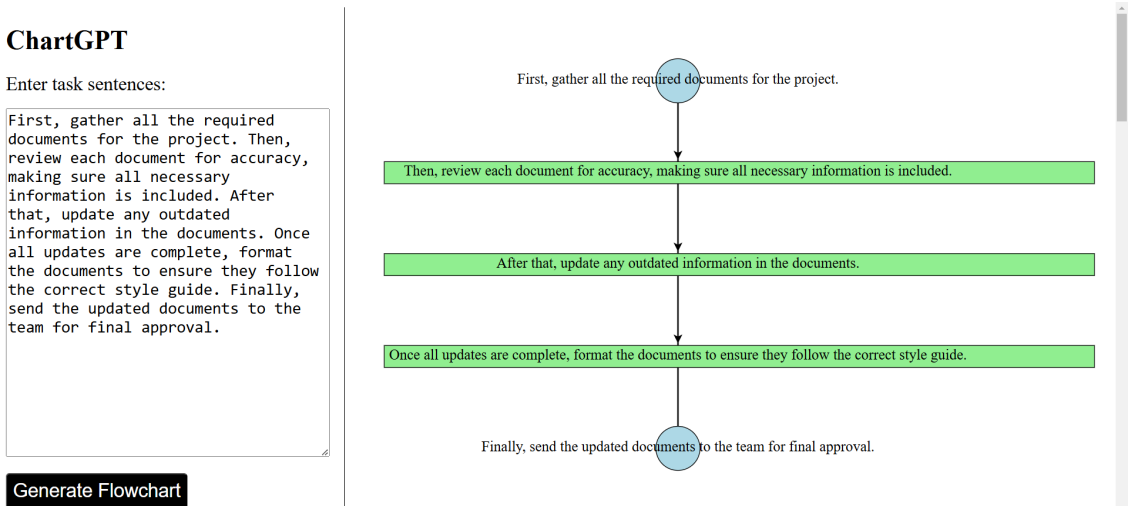

Figure 14: Sample Logic Diagram

Additionally, implementing unit tests for both Python and JavaScript components would ensure the reliability and maintainability of the codebase.

4.4 Supporting Development Tools

For our project, we've chosen a selection of tools to enhance productivity and collaboration. GitHub, with its efficient branching and merging, serves as our code hosting platform, ensuring seamless code integration and backup capabilities. Overleaf, a LaTeX editing tool, facilitates real-time collaboration on professional documentation, supported by version control for structured management. Microsoft Teams centralizes communication and project coordination, integrating with Microsoft Office for task management and featuring quick-resolution tools like chat and video conferencing. PyCharm, our IDE, is equipped with Python development features like code completion and debugging, integrated with version control for efficient coding. Visual Paradigm aids in UML diagram creation for clear system architecture visualization, while Figma enables interactive UI/UX design prototyping for consistent design and collaborative feedback. This suite of tools creates a cohesive ecosystem that streamlines our project's development process.

4.5 Technical Stack

This project generates logical logic diagrams from extensive text descriptions using NLP techniques. The technical stack chosen is designed to ensure smooth integration, high performance, and ease of visualization. Each stack component is selected to handle specific tasks that support the project's goals.

4.5.1 Front-end

The front-end interactions were asked to be flexible. As a result, our application is built using a combination of HTML, CSS, Node.js, and React. Integrating these four technologies ensures compatibility and flexibility in the front-end environment,

making it easy to handle real-time updates, manage user interactions, and ensure the application remains fast and responsive.

4.5.2 Back-end

The back-end of our application is developed using Python, which serves as the main processing engine for handling user input, managing data, and executing the NLP algorithms. The back-end is also responsible for managing the logic diagram generation process and serving the generated images to the front end.

- **Python and Flask/Django**: Python is our main back-end language, chosen for its readability and wide range of libraries, especially for NLP. Flask or Django creates a RESTful API, allowing seamless communication between the front-end and back-end components.

4.5.3 NLP Algorithm

The core of this application's logic logic diagram generation is based on NLP algorithms. For this, we use Python's `nltk` and `spaCy` libraries, which provide robust NLP capabilities for analyzing and extracting information from text inputs.

- **NLTK (Natural Language Toolkit)**: NLTK is used for basic text processing tasks, such as tokenization, stemming, and stopword removal. It helps us preprocess the text input before passing it to the more advanced models.
- **spaCy**: spaCy is employed for more advanced NLP tasks like named entity recognition, dependency parsing, and part-of-speech tagging. These features enable us to extract meaningful relationships and structures from the input text, which can then be translated into logical nodes and connections in the logic diagram.

4.5.4 Visualization and Logic Diagram Generation

To convert the NLP-processed data into visual logic diagrams, we use `d3.js` and `graphviz`.

- **d3.js**: D3.js is a powerful JavaScript library for data-driven document manipulation. In this project, D3.js allows us to dynamically render and manipulate the structure of the generated logic diagrams on the front end, giving users an interactive view of the output.
- **Graphviz**: Generates initial logic diagram structures from text data, creating SVG files for further customization with D3.js, streamlining graph layout through automatic node and edge arrangement.

5 Project Management

5.1 Roles and Responsibility

Table 1: Roles and Responsibility

Role	**Member Name**	**Responsibility**
Team Leader	Youyao Gao	• Guide and support team members • Set the deadlines and meeting schedules • Communicate the team's goals and priorities • Building and maintaining relationships with supervisor
Technical Lead	Yuhang Wu	• Provide technical guidance • Lead the development of the software • Mentor team members • Manage technical risks and issues
Front-end	Tongchang Liu, Youyao Gao	• Collaborate with the team to develop the front-end • Write and test the front-end code
Back-end	Shuqi Lee, Jason Widjaya, Tongchang Liu, Ziqi Li	• Design the back-end architecture • Write and test the back-end code • Collaborate with the front-end team
NLP Algorithm Engineer	Yuhang Wu, Shuqi Lee	• Develop NLP models to interpret input text • Collaborate with other engineers to integrate NLP models • Optimize NLP performance
Writer	Youyao Gao, Jason Widjaya, Ziqi Li	• Create, edit, and review documentation • Gather and organize information.

5.2 Time Plan

The project Gantt charts are drawn using Excel. This project Gantt charts show the time allocation from October 2024 to early April 2025.

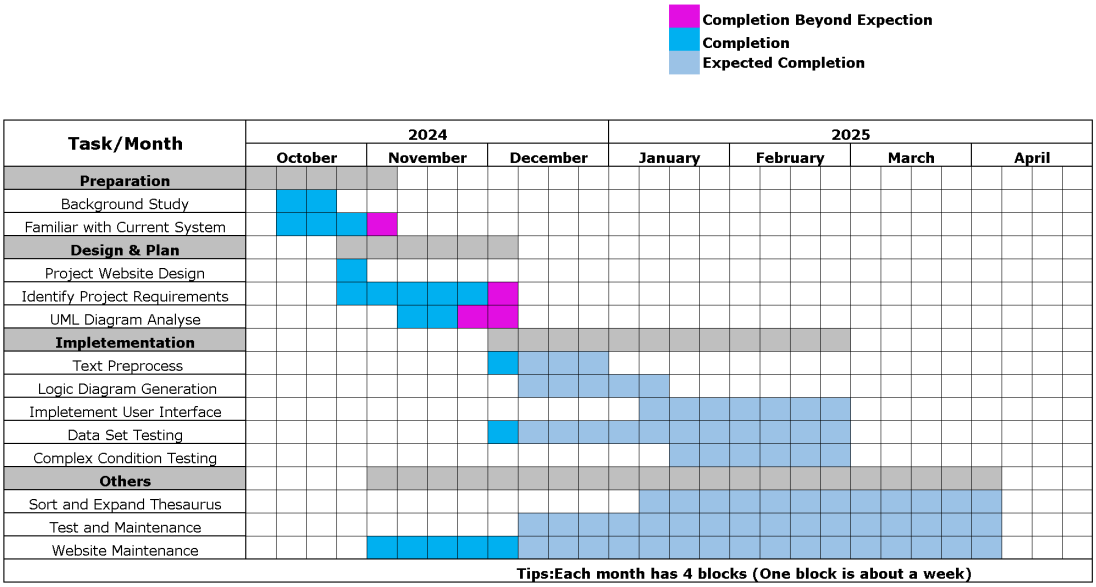

Figure 15: Gantt Chart

Figure 15 shows the project plan, which is basically based on chronological order. In October, the team mainly spent time on background research and learning about the current system. The team created a preliminary plan for the project in November and began some early design. Due to insufficient direct communication and inconsistent opinions, the team extended the time for determining requirements, and accordingly, the team also modified the analysis of UML diagrams. In December, the team will officially start systematic project implementation according to the plan. In the future, the team will continue to expand the thesaurus and code maintenance.

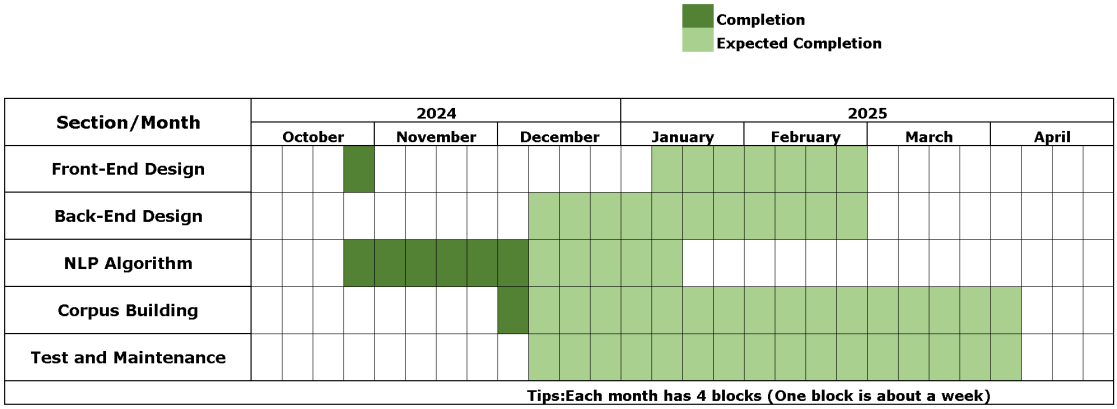

Figure 16: Labor Division

As shown in Figure 16, the team divides the whole project into five sections, and the diagram shows the timeline for each section.

6 Problems Encountered

6.1 Team Management Issues

The team has encountered some management issues. Firstly, team members occasionally miss group meetings, which significantly affects the efficiency of our collaboration and, consequently, everyone's time. Therefore, the team decides to strengthen the rules of teamwork. The team leader would announce everyone's task assignments and the results to be achieved at the end of each meeting and check and verify them before the next meeting. Additionally, there were disagreements among team members about how the work should be completed. However, in the end, everyone coordinates and agrees to divide into two groups: one to complete the report writing and the other to learn and familiarize themselves with technical tools. This arrangement maximizes the utilization of the team's skills by allowing members who excel in writing and those who excel in programming to focus on what they do best.

Problems in team collaboration are inevitable, but through the efforts and compromises of group members, everyone can work together more efficiently to complete project tasks and achieve better results.

6.2 Technical Issues

6.2.1 NLP Keyword search

After using NLP processing techniques to convert large sentences into words by tagging, the team tried to use verbs to determine the objects and order them directly. However, it was discovered that language's logical structure is not always linear and fixed; it may include complex relationships such as parallelism or conditionality. To solve this problem, the team developed a method to identify connectives within sentences that indicate the order, generating a better understanding of how objects relate to each other.

6.2.2 Keyword Application

To clarify the relationship between objects through connectives, the team created a thesaurus of connectives to determine the relationship between contexts by word matching. However, it is easy to overlook the relationship when encountering connectives not included in the thesaurus, potentially leading to gaps in understanding.

6.2.3 Logical relation judgment

Because the team relies on connectives to identify logical relationships, a special case can arise where a single word represents two distinct logical relationships. For instance, the word "since" illustrates this issue. In the sentence, "Since the project started, the team has grown significantly", the word "since" connects the timing of the project's start with the team's growth, indicating a temporal relationship. However, "since" expresses a causal relation in the sentence "Since the deadline is tomorrow, we need to work overnight", as it shows the reason for working overnight

is that the deadline is tomorrow. To address this challenge, the team must analyze the context and meaning of the sentence to interpret the intended relationship accurately.

6.2.4 Determining logical relationships

The team decided to use JSON FORMAT to implement logical relationships in a structured way. Initially, the team focused on objects and successfully generated a logic diagram based on chronological order. However, this method often led to disorder and unclear logic diagrams for complex logical relationships. After group research and discussion, the team introduced the concept of "tight front" and "tight behind", meaning that for a specific object, its logical upper layer and logical lower layer are determined. Moreover, it was found during testing that the method can also show a clear, logical relationship when an object has multiple "tight front" and "tight behind".

6.2.5 Modify the Logic Diagram on the Front-End

In the team's vision, users would be able to enter text and make changes to the flowchart as they saw fit. While achieving simple text edits in the process is relatively straightforward, modifying the progress structure is a significant challenge, especially when ensuring smooth communication between the front-end and back-end systems for dynamic interactions.

7 Conclusion

Based on the achievements this semester, it can be confidently concluded that the project has laid a strong foundation for success. The team approaches each stage methodically, beginning with an in-depth literature review to build a solid understanding of the project landscape. With the supervisor, Dr.Huan Jin, thorough requirement engineering is prioritized to ensure that every need is identified and addressed before moving into the software design. This commitment enables the team to make informed and strategic decisions regarding programming languages, methodologies, and collaboration tools, all of which are chosen to best support the objectives

As with any ambitious project, challenges are encountered along the way. However, each obstacle becomes an opportunity for growth. Through open communication and mutual support, these issues are tackled collectively, strengthening teamwork and enhancing individual skills. The efforts to resolve these difficulties advance technical understanding and refine collaborative processes, which will continue to benefit the team in future phases of development.

To conclude, the autumn semester has been a period of growth, learning, and achievement for the team. The critical early phases of the project are successfully navigated, setting the stage for continued progress. Looking ahead, there is confidence in preparedness and excitement for the next steps in transforming plans into

a fully realized solution.

References

Bostock, M. (2023). *D3.js: Data-driven documents for visualizations* [Accessed: 2024-11-12]. D3.js Community. https://d3js.org/

Chen, Q., Pailoor, S., Barnaby, C., Criswell, A., Wang, C., Durrett, G., & Dillig, I. (2022a). Type-directed synthesis of visualizations from natural language queries. *Proceedings of the 43rd ACM SIGPLAN International Conference on Programming Language Design and Implementation*, 590–603. https://doi.org/10.1145/3519939.3523728

Chen, Q., Pailoor, S., Barnaby, C., Criswell, A., Wang, C., Durrett, G., & Dillig, I. (2022b). Type-directed synthesis of visualizations from natural language queries. https://arxiv.org/abs/2209.01081

Chowdhury, A., Rahim, M. S., & Rahman, M. (2016, October). *Flow chart*.

Community, P. (2023). Plantuml documentation [Accessed: 2024-11-12]. https://plantuml.com/

Ellson, J., Gansner, E. R., Koutsofios, E., North, S. C., & Woodhull, G. (2001). Graphviz - graph visualization software [Accessed: 2024-11-12]. https://graphviz.org/

Gamage, Y. L. (2024). Automated software architecture diagram generator using natural language processing [Accessed: 2024-11-28]. https://www.researchgate.net/profile/Yasitha_Gamage/publication/371011658_Automated_Software_Architecture_Diagram_Generator_using_Natural_Language_Processing/links/6470c29b6fb1d1682b0d0e3d/Automated-Software-Architecture-Diagram-Generator-using-Natural-Language-Processing.pdf

OpenAI. (2023). Chatgpt: An ai model for text understanding and summarization [Accessed: 2024-11-12]. https://openai.com/research/

Team, L. (2023). *Diagramming and visualization tools: Manual vs automated approaches*. Lucid Software Inc. https://www.lucidchart.com/pages/

Tian, Y., Cui, W., Deng, D., Yi, X., Yang, Y., Zhang, H., & Wu, Y. (2024). Chartgpt: Leveraging llms to generate charts from abstract natural language. *IEEE Transactions on Visualization and Computer Graphics*, 1–15. https://doi.org/10.1109/tvcg.2024.3368621

Unknown, A. (2021). Flowchart generation from natural language descriptions. *arXiv preprint arXiv:2104.04584*. https://arxiv.org/abs/2104.04584

Verified Market Reports. (2023). Online flowchart software market size and forecast 2023 to 2030 [Accessed: 2024-11-12]. https://www.verifiedmarketreports.com/product/online-flowchart-software-market/

Zeng, Y. (2008). Recursive object model (rom)—modelling of linguistic information in engineering design. *Computers in Industry*, *59*(6), 612–625. https://doi.org/https://doi.org/10.1016/j.compind.2008.03.002

Zhong, S., Scarinci, A., & Cicirello, A. (2023). Natural language processing for systems engineering: Automatic generation of systems modelling language diagrams. *Knowledge-Based Systems*, *259*, 110071. https://doi.org/https://doi.org/10.1016/j.knosys.2022.110071

A Appendices - Meeting Minutes

A.1 Formal Meeting 1

Date and Time: 18 October 2024 at 11:20-12:00PM
Location: PMB 409
Supervisor: Huan Jin
Content:

- Introduction of team members and their specialty
- First phase of requirement engineering
- Discussion regarding the ethical issue

A.2 Formal Meeting 2

Date and Time: 24 October 2024 at 11:20-12:00PM
Location: PMB 409
Supervisor: Huan Jin
Content:

- Current progress report
- Setting a standard of the software version that will be used
- Further requirement engineering
- Discussion about literature review

A.3 Formal Meeting 3

Date and Time: 31 October 2024 at 11:20-12:00PM
Location: PMB 449
Supervisor: Huan Jin
Content:

- Current progress report
- Set the workflow processes
- Discussion on a minimum prototype version of the software
- Further requirement engineering
- Discussion about literature review

A.4 Formal Meeting 4

Date and Time: 31 October 2024 at 11:20-12:00PM
Location: PMB 449
Supervisor: Huan Jin
Content:

- Current progress report
- Set the workflow processes
- Discussion on a minimum prototype version of the software
- Further requirement engineering
- Discussion about literature review

A.5 Formal Meeting 5

Date and Time: 7 November 2024 at 11:20-12:00PM
Location: PMB 449
Supervisor: Huan Jin
Content:

- Current progress report
- Set the workflow processes
- Discussion on a minimum prototype version of the software
- Further requirement engineering
- Discussion about literature review

A.6 Formal Meeting 6

Date and Time: 14 November 2024 at 10:00-:1100PM
Location: Dr.Huan Jin office
Supervisor: Huan Jin
Content:

- Current progress report
- Further discussion on workflow and back-end
- Revise interim report structure

A.7 Formal Meeting 7

Date and Time: 21 November 2024 at 10:00-:1100PM
Location: PMB 449
Supervisor: Huan Jin
Content:

- Current progress report
- Interim report evaluation and feedback

A.8 Formal Meeting 8

Date and Time: 28 November 2024 at 10:00-:1100PM
Location: PMB 449
Supervisor: Huan Jin
Content:

- Current progress report
- Interim report evaluation and feedback
- Back-end feedback

B Appendices - LIST OF ABBREVIATIONS

Acronym	Description
NLP	Natural Language Processing
UI	User Interface
UX	User Experience
UML	Unified Modelling Language
LLMs	Large Language Models

Table 2: List of Abbreviations