

# **State-of-the-Art CCUS integration to flowchart development using Generative AI and Large Language Models**

*Submitted by:*

**Harshit Bhalla**

**2020CH10088**

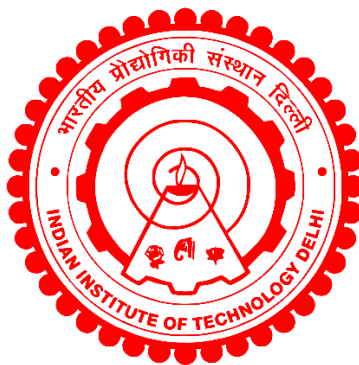
*Under the guidance of:*

**Prof. Hariprasad Kodamna**

*in partial fulfilment of the requirements*

*for the degree of*

**Bachelor of Technology**



**Department of Chemical Engineering**

Indian Institute of Technology Delhi (IITD)

Diwali Semester 2023

# DECLARATION

I declare that this written submission represents my ideas in my own words and where other ideas or words have been included, I have adequately cited and referenced the original sources. I also declare that I have adhered to all the principles of academic honesty and integrity and have not misinterpreted or fabricated or falsified any idea / data / fact / source in my submission. This declaration serves as a testament to my commitment to upholding the highest standards of academic excellence and ethical conduct.

With self-attestation:

Harshit Bhalla(2020CH10088)

Date:- **27/11/2023**

# ACKNOWLEDGEMENTS

I wish to express my deepest gratitude to my supervisor, Prof. Hariprasad Kodamna for giving me the chance to work on this novel problem under his constant encouragement throughout the project, for his patience, motivation, enthusiasm and immense knowledge. He consistently allowed me to frame my own work, but steered me in the right direction whenever I needed it. I would also like to thank Avan for being an excellent researcher, and guiding me in areas of his expertise. I thank IIT Delhi HPC Facility for compute resources. Lastly, I would like to dedicate this work to my beloved parents and my sister Akanksha for their love and constant support throughout my life.

# ABSTRACT

This report presents a comprehensive account of the development and implementation of an innovative application designed to transform chemical engineering processes into detailed flow diagrams. This application, crucial for chemical engineers and industry professionals, marks a significant advancement in process visualization and interpretation. Under the guidance of Prof. Hariprasad Kodamna, I have been able to successfully create the application. I believe that this can automate state-of-the-art process flow diagram construction for major Carbon Capture, Utilization, and Storage (CCUS)-integrated and related industry processes using large language models and generative AI.

The core of this application involves the utilization of cutting-edge Large Language Models (LLMs). After extensive testing of various LLMs in the industry, the GPT-3.5-turbo model was selected for its superior performance in generating adjacency lists from process descriptions. This selection was based on the model's ability to adhere to strict guidelines and output parameters tailored for this specific application. A significant aspect of this project includes the conversion of these adjacency lists into intuitive flow diagrams. This was achieved using a specialized JavaScript package, highlighting the interdisciplinary approach of combining AI with web technologies.

To further enhance the model's accuracy and utility, a dataset comprising approximately 1500 entries has been curated. These entries, representing a single-line process and its corresponding entity relationships, serve as the foundation for model training. Recognizing the need for a more intricate dataset for fine-tuning, the effort to expand and refine this dataset is ongoing, with completion targeted for December. In parallel, work is progressing on the fine-tuning of the Llama-2 multi-billion parameter model. Completion of this phase is anticipated by January, setting the stage for the launch of a market-ready product. This product, a culmination of this research, will possess the unique capability to convert any textual process description into a comprehensive flow diagram, significantly benefiting the field of chemical engineering and related industries

# CONTENTS

- i. Acknowledgements
- ii. Abstract
- iii. List of Figures
- iv. Introduction
  - a. LLM and Generative AI
  - b. Background
  - c. Objective of my BTP
  - d. Motivation
  - e. Methodology
  - f. Scope & Contribution
- v. Literature Review
- vi. Modelling of a Process to a directed Graph
  - a. Model Selection
  - b. API style prompting
  - c. Output of the model
  - d. An explanatory example
- vii. Visualization
  - a. LLM Output Processor
  - b. Node Hasher function
  - c. Mermaid.js and Frontend UI
  - d. Example continued
- viii. Results & Conclusion
- ix. A look forward
- x. References

## List of Figures

Figure 1: The piece of code written to build an API styled prompt from a text corpus input.

Figure 2: The piece of code written to extract relationships from a prompt, as source-target-process

Figure 3: The output generated from the LLM for the example of Monochlorobenzene production taken.

Figure 4: The piece of code written to process the LLM output to permit frontend UI rendering.

Figure 5: A hashing function, built to hash a node into a unique 32 bit string, to build a foolproof graph.

Figure 6: UI of the Frontend, which is hosted locally and connects to backend via RESTful architecture.

Figure 7: The final renderable output from the backend, which is fed to the frontend.

Figure 8: The generated Flow diagram for Monochlorobenzene production from the application.

# Introduction

## LLMs and Generative AI

Large Language Models (LLMs) and Generative AI are specific branches within the field of artificial intelligence that are primarily concerned with the production of intricate outputs, such as text, graphics, or code. These outputs are generated by the utilization of comprehensive training data. Legal Master of Laws (LLM) programs, such as OpenAI's GPT series, undergo extensive training using large volumes of textual data, which enhances their capacity to comprehend and produce writing that closely resembles human language. Generative artificial intelligence (AI) possesses the capability to generate novel information, hence presenting inventive solutions across many domains.

The utilization of LLMs and Generative AI in the field of chemical engineering has the potential to make substantial contributions to the domains of research and development. These technologies possess the capability to examine and synthesize huge quantities of research papers, patents, and technical documents, thereby offering significant insights and facilitating the process of hypothesis formation. Complex data sets can be processed by them in order to make predictions about chemical reactions, materials properties, and process outcomes. This capability serves to improve the efficiency and accuracy of chemical engineering processes.

The construction of process graphs in Carbon Capture, Utilisation, and Storage (CCUS) can be significantly transformed through the implementation of Generative AI and Large Language Models (LLMs). LLMs efficiently extract and interpret critical data from the vast body of CCUS literature, whereas Generative AI is responsible for constructing and optimising these process graphs. This integration facilitates the accurate and dynamic visualisation of CCUS systems, which in turn improves the efficiency of process optimisation and simulation. Automated graphs have the dual benefit of facilitating cross-disciplinary collaboration and enhancing research and development, thereby rendering intricate CCUS processes more accessible and understandable to a wide range of stakeholders. The integration of these AI technologies signifies a substantial advancement in the ability to visualise and comprehend CCUS systems into a PFD which is of high chemical industry relevance.

## Background

Carbon Capture, Utilization, and Storage (CCUS) encompasses a suite of advanced technologies and methodologies designed to capture carbon dioxide (CO<sub>2</sub>) emissions. These emissions can originate from specific point sources or be directly sequestered from the ambient atmosphere. Once captured, the carbon dioxide can either be stored subterranean or repurposed to fabricate valuable commodities. The integration of CCUS technologies across various sectors is imperative to address the escalating global climate challenges and achieve the prescribed climate targets. (*International Energy Agency, n.d.*)

The application of Carbon Capture, Utilization, and Storage (CCUS) extends beyond industrial emissions reduction, to enhancing oil recovery and creating building materials. In the industrial sector, it plays a crucial role in mitigating emissions from heavy industries such as steel and cement production, which are traditionally challenging to decarbonize.

Furthermore, the conversion of captured CO<sub>2</sub> into products like biofuels, plastics, and concrete not only provides an avenue for reducing greenhouse gas concentrations in the atmosphere but also propels a circular carbon economy. This transformative approach underlines the importance of CCUS in bridging the gap between current industrial practices and a more sustainable, low-carbon future.

CCUS not only addresses the challenges of decarbonizing heavy industries but also contributes to the development of a circular carbon economy. This transformative approach underscores the pivotal role of CCUS in transitioning towards a more sustainable and low-carbon future.

## Objective of my BTP

Realizing the importance of CCUS, the primary objective of my B. Tech project is to seamlessly automate creation of a textual chemical engineering process to a Flow Diagram (PFD), aiming for both efficacy and efficiency, using a given text corpus with CCUS already integrated. By doing so, I seek to provide a structured and systematic approach to assimilating CCUS into industrial workflows, thereby enhancing their environmental sustainability.



## **Motivation**

The urgent need to combat climate change and reduce our carbon footprint has brought carbon capture, utilization, and storage (CCUS) to the forefront of environmental sustainability. My B. Tech project was motivated by the importance of Carbon Capture, Utilization and Storage (CCUS) in reducing emissions, especially in heavy industries such as steel and cement manufacturing, which are not easy to decarbonise. Therefore, my goal is to simplify this process by developing an application that will transform CCUS integrated text wholes into interesting process flow diagrams.

Translating chemical engineering process descriptions into process flow diagrams (PFDs) for combined carbon capture, utilization, and storage (CCUS) systems is bridging significant differences in business today. The purpose of this technology is to increase the efficiency of introducing CCUS technology to the current market, thereby increasing the accessibility and affordability of adoption in business. My report provides an important and useful framework for designing and implementing a Carbon Capture, Utilization and Storage (CCUS) program. This program not only improves engineering performance, but also aligns with international efforts to create a sustainable, low-carbon future. The project therefore goes beyond academic research and represents an important step towards making a real contribution to the fight against climate change.

## **Methodology**

The methodology I've employed in my B. Tech project involved developing a multi-phase strategy that centred on the creation and enhancement of an application designed to transform chemical engineering process descriptions into flow diagrams. The following steps outline the methodology:

### **1. Application Development:**

The first stage of the project comprised the creation of a comprehensive application with the ability of modelling a chemical engineering process into a flow diagram.

The purpose of this application was specifically developed to meet the requirements of chemical industries and processes, with a special emphasis in Carbon Capture, Utilisation, and Storage (CCUS) integration.

### **2. Testing and Deployment of Large Language Models (LLMs):**

- Multiple LLMs available in the industry were tested for their ability to interpret and process chemical process descriptions. Like Meta's Llama-13b, Falcon-7b, GPT-3.5-turbo, etc.
- The GPT-3.5-turbo model was specifically deployed, with customized API based prompting parameters to create an adjacency and hashing based list from the given process descriptions. These parameters were designed to follow strict rules to achieve the desired output format.

### **3. Visualization Using JavaScript:**

- The generated adjacency list from the LLM backend was then sent to Frontend using RESTful Architecture, and then visualized into a flow diagram. This step was crucial in converting the textual data into a graphical representation, making it more accessible and interpretable.

### **4. Dataset Curation and Model Training:**

- For long-term enhancement and fine-tuning of the model, a dataset of approximately 1500 entries was curated. Each entry comprised a single-line process description and its corresponding output in an "entity->entity: relationship" format.
- Initial fine-tuning was conducted on the GPT model using this dataset. However, limitations in pattern recognition and data representation were identified.

### **5. Dataset Expansion and Complex Modelling:**

- Recognizing the need for a more complex dataset to overcome the identified limitations, the next work will focus on curating an enhanced dataset.
- The second phase of the project, involves fine-tuning more advanced models like Llama-2-70b and GPT 3.5-turbo with this expanded dataset.

### **6. Model Selection and Product Finalization:**

- Post fine-tuning, the models will be evaluated based on training losses and testing accuracy.
- The final phase will involve selecting the most effective model. The chosen model will be integrated into the application, resulting

in a market-ready product capable of transforming any text corpus of a process into a flow diagram.

## Scope and Contribution

The scope of my BTP includes multiple domains:

1. My project offers a comprehensive examination of the whole lifetime of the process, encompassing the integration of CCUS into various processes, the translation of this integration into written input, and the subsequent conversion of the text into informative flow diagrams.
2. Targeting industries that are either now utilising CCUS or contemplating its integration. The flow diagrams that are developed will play a crucial role in formulating strategies and optimising the implementation of Carbon Capture, Utilisation, and Storage (CCUS) technologies.
3. The project's methodology and conclusions hold significant value for academic scholars investigating the application of CCUS and its representation in many media.
4. The provision of flowcharts will provide policymakers, environmentalists, and industry leaders, with clear guidance in their respective fields understanding the intricacies of carbon capture, utilisation, and storage (CCUS). Additionally, the utilisation of these flowcharts will significantly decrease operating time, hence offering potential monetary benefits. The topic will be addressed in the following sections.

My contribution ideology can be explained by the following points:

1. **Holistic Representation:** This undertaking offers a thorough examination of textual descriptions- tracing its origins in industrial processes, its depiction in written materials, and ultimately, its representation through the use of flowcharts..
2. **Informed Decision-making:** By translating complex processes descriptions into understandable visuals, industries are better equipped to make informed decisions regarding its adoption and optimization.

3. **Standardization:** As industries grapple with varied methodologies for CCUS integration, the flow diagrams can serve as a standard reference, ensuring consistency and accuracy in application.
4. **Enriching Academic Discourse:** By introducing a systematic approach to visualizing CCUS integration, the project paves the way for further academic explorations, fostering innovation in both technological and representational spheres.

## Literature Review

This literature review consolidates research findings and discussions regarding Process Flow Diagrams (PFDs) and their vital role in chemical engineering. A synthesis of academic and professional sources unveils the multidimensional functions of PFDs, from conceptual design to economic analysis.

**InformIT** provides a comprehensive exploration of various chemical process diagrams, delineating the different process scales depicted. The article underscores the utility of PFDs in advancing process technology and improving economic outcomes through strategic material representation (*InformIT, 2023*).

**ScienceDirect Topics** offers an overview of PFDs, highlighting their application in describing process steps using verb-object constructs. It underscores the criticality of stakeholder engagement and walk-throughs for accurate interpretation of these diagrams (*ScienceDirect Topics, 2023*).

In an educational webinar, **AIChE** features Professor Barry M. Barkel's insights into the core elements of PFDs. Professor Barkel elucidates the definition, communicative value, interpretive strategies, and the semi-universal standards of PFDs, emphasizing their importance to chemical engineers and related professionals (*AIChE, n.d.*).

**Michigan Technological University** presents a chemical engineering course flowchart. This resource contextualizes PFDs within the broader educational framework, touching upon global sustainability, environmental regulations, green chemistry, and life cycle assessments, thereby illustrating the field's commitment to eco-conscious engineering practices (*Michigan Technological University, 2023*).

At last, a publication within the **CAPEC research center at the Chemical Engineering Department** addresses the creative aspect of PFDs. The report delves into the generation and design of process flow sheets via a group contribution approach, showcasing a methodology for innovative flow sheet development (*d'Anterrockes, 2006*).

This literature review underscores the significance of PFDs in chemical engineering and provides insights into their diverse applications. Existing research in this area focuses on enhancing the understanding and effectiveness of PFDs, with a focus on sustainability and stakeholder involvement.

## **Modelling of a Process to a directed Graph**

The process of transforming textual descriptions of chemical engineering processes into directed graphs using Large Language Models (LLMs) involves three key stages: model selection, API-style prompting, and analysis of the model's output.

I'll now explain each component and how its implemented in my code:

### **Model Selection:**

The choice of the right LLM is critical for the success of the project. Factors considered in this selection include the model's ability to understand and process complex technical language, its adaptability to custom prompts, and its performance in terms of accuracy and efficiency.

For this project, GPT-3.5-turbo was initially selected for its advanced natural language processing capabilities. Its large dataset training and sophisticated architecture make it adept at understanding and generating technical content. Additionally, the potential to fine-tune Llama-2-70b offers an avenue to explore more specialized, industry-specific language models that could yield better results for chemical engineering applications, particularly in CCUS contexts, and is a scope of further improvement which will be carried out in the next semester.

## API Style Prompting

The technique of API-style prompting entails the formulation of well-structured prompts that serve as guidance for the Language Model (LM) to produce the intended result. The project involves the careful design of prompts to guide the model in analysing a text corpus containing descriptions of chemical processes. The model then generates data that accurately depict the various aspects of a process flow diagram. These prompts are structured to include:

- A clear description of the task, emphasizing the need to identify key components (like reactants, products, and intermediates) and their relationships.
- Specific instructions on output format, guiding the model to present its findings as an adjacency list, which is a precursor to graph formation. This format includes source nodes, destination nodes, and descriptions of the processes connecting these nodes.

The JSON API format prompt that I've written in the backend API code is:

```
function buildPrompt(textCorpus) {  
  const promptData = {  
    "instructions": {  
      "context": "Chemical Engineering Process Analysis",  
      "task_description": "Analyze the text to construct a process flow diagram in a JSON format.",  
      "objectives": [  
        "Identify process units (nodes).",  
        "Identify flow of materials or operations (edges).",  
        "Provide a clear and concise description for each process transition."  
      ],  
      "output_requirements": {  
        "format": "Output each connection as a JSON object within an array.",  
        "example": `[  
          {  
            "from": "Example Start Process",  
            "to": "Example End Process",  
            "edge_description": "Example Process Description",  
          },  
        ]`,  
        "notes": "Use an empty string for 'edge_description' if not applicable."  
      },  
      "process_coverage": [  
        "Include steps from initial process selection to process flowsheet issuance.",  
        "Incorporate equipment selection, specification, and design."  
      ],  
      "quality_guidelines": {  
        "conciseness": "Output should be concise and accurate.",  
        "relevance": "Reflect the process flow described in the text.",  
        "standard_practices": "Adhere to chemical engineering principles and industry practices."  
      },  
      "restrictions": {  
        "content_limitations": ["Do not include summary or conclusion."],  
        "format_adherence": "Strictly follow the provided JSON format for output."  
      }  
    },  
    "text_corpus": textCorpus  
  };  
}
```

Figure 1

## What this code does?

This JavaScript code defines a function named **buildPrompt** that generates structured prompt data for an AI model, particularly designed for analyzing a chemical engineering process described in a text corpus. The function takes **textCorpus** as an argument and returns an object with detailed instructions on how the AI should process this text.

The instructions include:

- The context of the task: Chemical Engineering Process Analysis.
- A detailed task description: To analyze text and construct a process flow diagram in JSON format.
- Objectives such as identifying process units (nodes) and materials or operations (edges) and providing descriptions for each process transition.
- Output requirements specifying the format expected from the AI: an array of JSON objects representing the process connections.
- Process coverage details including steps like initial process selection and equipment design.
- Quality guidelines emphasizing conciseness, relevance, and adherence to standard practices.
- Restrictions on what should not be included in the content and adhering strictly to the JSON format for the output.

The **text\_corpus** field at the end of the object is intended to hold the actual text that the AI will analyze.

When the **buildPrompt** function is called with a text corpus, it will return this structured prompt data, which will then be passed to the GPT-3.5-Turbo model for graph processing. The response will conform to the instructions, enabling the construction of a process flow diagram that accurately represents the described chemical engineering process.

## Output of the model:

Whenever the user types a process description in the frontend, and hits the generate graph button on the frontend, an API call is made to the backend endpoint.

**Whats an API?** An API, or Application Programming Interface, is a set of protocols, routines, and tools for building software applications. They are mechanisms that enable two application components to communicate with each other using a set of definitions and protocols. APIs are used to enable the integration between different systems or layers of an application. (*Amazon Web Services, Inc., n.d.*)

The backend receives the HTTP request at an endpoint, I've created, which triggers a function to process the request. This function would use the input data to interact with the language model, in the following manner:

```
async function extractRelationships(input) {
  try {
    const prompt = buildPrompt(input);
    const chatCompletion = await openai.chat.completions.create({
      model: "gpt-3.5-turbo",
      messages: [
        {
          role: "system",
          content:
            "Extract 'source-target-process' relationships from the given text.",
        },
        { role: "user", content: prompt },
      ],
    });

    if (
      chatCompletion &&
      chatCompletion.choices &&
      chatCompletion.choices.length > 0
    ) {
      const responseText = chatCompletion.choices[0].message.content;
      let relationships = outputProcessor(responseText);
      return convertToMermaidGraph(relationships);

      // return relationships;
    } else {
      console.log("No data found in response");
      return null;
    }
  } catch (error) {
    console.error("Error in extracting relationships:", error);
    return null;
  }
}
```

Figure 2



This JavaScript code snippet is an asynchronous function designed to interact with the large language model to extract "source-target-process" relationships from a given text and convert them into a flow graph.

## Code Explanation:

### Function `extractRelationships(input)`

- **Purpose:** This is an asynchronous function named that takes an **input** parameter.
- **Process:**
  - The function starts by constructing a prompt using the **buildPrompt(input)** function, which is explained prior to this section, is used to format the input in a specific way that the model expects.
  - It then calls the GPT-3.5-Turbo model using the **chat.completions.create** method, passing in the model identifier **gpt-3.5-turbo** and a series of messages. The first message from the 'system' role provides instructions to the model to extract relationships from the text. The second message contains the user's input, which is the prompt constructed above.
  - The **await** keyword is used to wait for the API call to resolve, meaning the function will pause at this point until the OpenAI API responds.

### Conditional Check

- **Purpose:** To check if the API call returned a valid response.
- **Process:**
  - If the **chatCompletion** object and its **choices** array are valid and the array contains at least one element, it processes the first choice.
  - The content of the first choice is considered as the response text (**responseText**), which contains the extracted relationships.

- This **responseText** is then processed by an **outputProcessor(responseText)** function (not shown in the snippet), which likely parses the text and extracts relationship data.
- The extracted relationships are then converted into a flow graph using the **convertToMermaidGraph(relationships)** function, which will be discussed in the upcoming sections.
- 

## Error Handling

- **Purpose:** To catch and handle any errors that occur during the execution of the API call.
- **Process:**
  - If the response does not contain any data or an error occurs during the API call or processing of the data, the function logs an error message to the console and returns **null**.

The output of this code is a directed process graph, which is then returned to the frontend and the API promise is then established.

## An explanatory example of the backend architecture

In this section, we will dissect the layers of our backend architecture, revealing the intricate web of services and protocols. We will explore the server at its core, the data processing, and how the communication happens across the application.

I've written a code which is meant to be scalable, reliable, and secure. The coding language used is **JavaScript**, primarily because its event driven asynchronous, scalable, best in JSON handling and cross-platform production, and has a large community support in Full-stack Software development, which makes my application A State-of-the-art software.

Let's consider the **Monochlorobenzene Production** process, which is sent from the frontend to the backend.

The input looks like:

*“Monochlorobenzene is produced by the reaction of benzene with chlorine. A mixture of monochlorobenzene and dichlorobenzene is produced, with a small*

*amount of trichlorobenzene. Hydrogen chloride is produced as a byproduct. Benzene is fed to the reactor in excess to promote the production of monochlorobenzene. The reactor products are fed to a condenser where the chlorobenzenes and unreacted benzene are condensed. The condensate is separated from the noncondensable gases in a separator. The noncondensables, hydrogen chloride and unreacted chlorine, pass to an absorption column where the hydrogen chloride is absorbed in water. The chlorine leaving the absorber is recycled to the reactor. The liquid phase from the separator, containing chlorobenzenes and unreacted benzene, is fed to a distillation column, where the chlorobenzenes are separated from the unreacted benzene. The benzene is recycled to the reactor.” (Doe, 2023)*

This is then sent to the **buildPrompt()** function, which basically creates an input prompt, and can be then fed to the model for adjacency list production.

After querying this input to the model, the output generated from the LLM looks like:

```
Server is running on port 3000
[
  'From: Benzene',
  'To: Reactor',
  'Edge Description: Reaction with chlorine',
  '',
  'From: Reactor',
  'To: Condenser',
  'Edge Description: Products cooling and condensation',
  '',
  'From: Condenser',
  'To: Separator',
  'Edge Description: Separation of condensate and noncondensable gases',
  '',
  'From: Separator',
  'To: Absorption Column',
  'Edge Description: Noncondensables absorption in water',
  '',
  'From: Absorption Column',
  'To: Reactor',
  'Edge Description: Chlorine recycle',
  '',
  'From: Separator',
  'To: Distillation Column',
  'Edge Description: Separation of chlorobenzenes and unreacted benzene',
  '',
  'From: Distillation Column',
  'To: Reactor',
  'Edge Description: Benzene recycle'
]
```

Figure 3

This shows the adjacency list that is generated from the graph, which will be sent to the UI **outputProcessor()** function, which is described below in visualization section.

## Visualization

This step is the final part of the application, which involves visualization of the graph we have generated from the LLM above. Visualization represents a key point in an application where data and insights from the Large language model (LLM) are transformed into an easy-to-understand and interactive format for end users. This step is more than just presenting the data. It fosters understanding, reveals underlying patterns, and reveals hidden values that can facilitate informed decision-making. As part of my project, the visualization process is carefully designed to ensure clarity, accuracy, and user engagement.

The essential pivotal steps involved in this step of visualization are:

### LLM Output Processor:

```
function outputProcessor(output) {  
  const lines = output?.split('\n');  
  console.log(lines);  
  const nodes = {};  
  const links = [];  
  
  for (let i = 0; i < lines.length; i += 4) {  
    let fromNode = lines[i]?.replace('From: ', '').trim();  
    let toNode = lines[i + 1]?.replace('To: ', '').trim();  
    if (fromNode === '' || toNode === '') fromNode = toNode = 'output/input';  
    const processDescription = lines[i + 2]?.replace('Edge Description: ', '').trim();  
    // Create or update nodes  
    if (!nodes[fromNode]) {  
      nodes[fromNode] = { id: (fromNode) };  
    }  
    if (!nodes[toNode]) {  
      nodes[toNode] = { id: (toNode) };  
    }  
  
    // Create a Link between fromNode and toNode  
    links.push({  
      source: hasher(fromNode),  
      target: hasher(toNode),  
      description: processDescription,  
    });  
  }  
  
  // Convert nodes object to an array  
  const nodesArray = Object.values(nodes);  
  return { nodes: nodesArray, links };  
}
```

Figure 4

The essential steps followed by this function are:

1. **Split Output into Lines:** The output string is split by newline characters ('\n'), which creates an array of lines, considering each line contains information about the nodes and their relationships.
2. **Initialize Data Structures:**
  - **nodes:** An object that will hold node information, keyed by the node identifier.
  - **links:** An array to store the relationships (edges) between nodes.
3. **Iterate Over Lines:**
  - The for loop processes the lines in steps of four, suggesting that each node and relationship detail spans four lines.
  - **fromNode** is extracted by removing the word "From: " from the line and trimming any whitespace.
  - **toNode** is similarly extracted by removing the word "To: ".
  - If **fromNode** or **toNode** is empty, it defaults them to 'output/input', which might be a way to handle start or end points in the process that do not have explicit nodes defined.
4. **Process Descriptions and Nodes:**
  - **processDescription** is captured by removing the phrase "Edge Description: " from the line and trimming whitespace.
  - Nodes are checked for existence in the **nodes** object, and if they don't exist, they are initialized with an **id** property.
5. **Create Links (Edges) Between Nodes:**
  - For each relationship, an object is created and pushed into the **links** array.
  - This object contains **source** and **target** properties, which are the hashed versions of **fromNode** and **toNode**, respectively, created by the **hasher** function (not shown in the snippet).
  - The **description** property holds the **processDescription** for that particular link.

## 6. Conversion of Nodes Object to Array:

- Once all lines are processed, the **nodes** object is converted into an array using **Object.values(nodes)**.

## 7. Return Processed Data:

- The function returns an object containing two properties: **nodes** (an array of node objects) and **links** (an array of link objects).

The interpretation of the LLM's output is the primary function of this component, which serves as the foundation of the visualization pipeline. The primary function of this process is to analyze and clean the data and remove unnecessary nodes and edges that do not add to the main flow of the process. This stage involves intricate parsing, where the data is methodically formatted to comply with the syntactical demands of Mermaid.js, a renowned library specializing in rendering graphical diagrams. The challenge here was to handle the nodes described in strings populated with a variety of special characters. Such characters could disrupt the parsing logic or introduce syntactic anomalies when constructing the graph.

## Node Hasher Function:

To overcome the issues outlined earlier, a custom-designed hashing method has been implemented cleverly. The function assigns a unique 32-bit string hash to each node, based on the hashed value of the node, which simplifies the representation of the nodes' descriptions in a format compatible with Mermaid. These hashes serve as unique identifiers for the nodes, preserving the integrity of the relationships while stripping away problematic characters.

```
function hasher(str) {  
  let hash = 0;  
  const hashLength = 12;  
  for (let i = 0; i < str?.length; i++) {  
    hash = ((hash << 5) - hash + str.charCodeAt(i)) & 0xffffffff;  
  }  
  let hashString = hash.toString();  
  if (hashString.length < hashLength) {  
    hashString = hashString.padStart(hashLength, "0");  
  }  
  return hashString;  
}
```

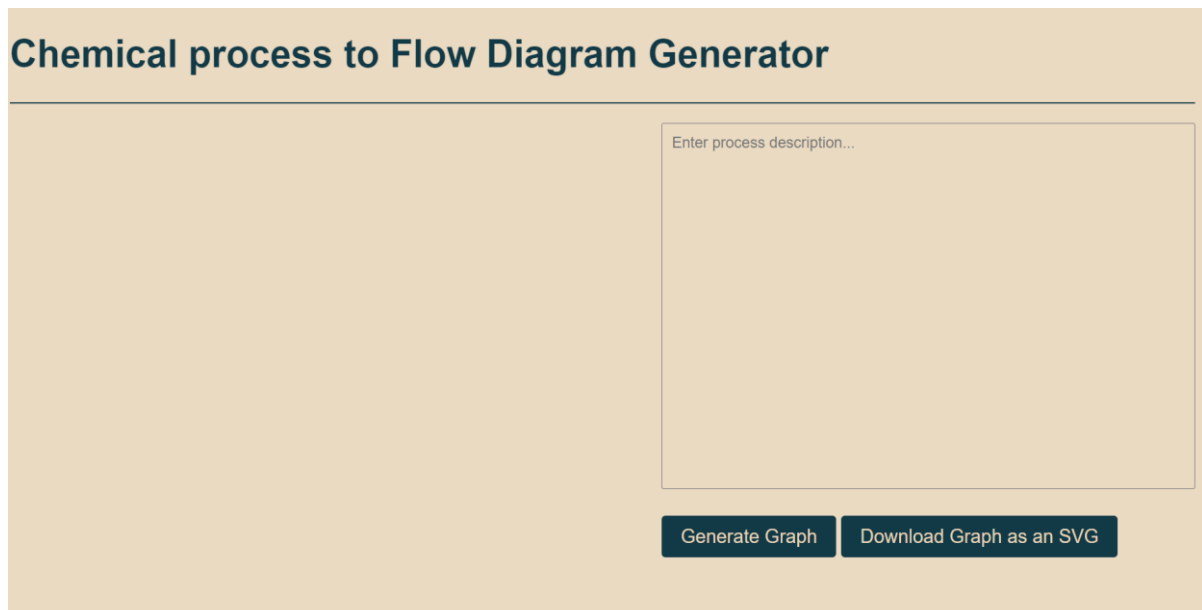
Figure 5

The JavaScript function hasher in the image generates a hash code from a string. This function sets a hash variable to 0 and sets a 12-character hashLength. It then iterates over each character in the input string, left-shifting the hash value by 5 bits and adding the character code. This result is bitwise ANDed with 0xffffffff to keep the hash inside a 32-bit integer range.

Following the loop, the numerical hash is transformed to hashString. If hashString is shorter than hashLength, leading zeros are inserted to make it longer. Finally, the function returns the hashString, a consistent, fixed-length input string hash. This hash can be used to identify nodes in the visualisation process, preventing special characters from interfering with graph description language syntax.

## Mermaid.js and Frontend UI

In the Frontend I've set up a process to dynamically import the Mermaid library, which is a tool for generating diagrams and flowcharts, when the document is fully loaded. I configure Mermaid to not automatically start rendering diagrams on page load.



The image shows a web application interface titled "Chemical process to Flow Diagram Generator". It has a light beige background. On the right side, there is a text input area with a placeholder text "Enter process description...". Below this input area, there are two dark blue buttons with white text: "Generate Graph" and "Download Graph as an SVG".

Figure 6

I then listen for a click event on a button "Generate Graph". When this button is clicked, I fetch the user's input from a text area. I send this input to my backend service at the endpoint `"/getOpenAIResponse"` using a POST request. If successful, the backend will return a response that includes a graph definition, which I assume is in the correct format for Mermaid to render.

I then create a new UI-DOM element, and in this element I render the graph generated and display Process Flow diagram.

Additionally, I've set up an event listener for downloading the graph generated. When clicked, it serializes the SVG element(which represents the diagram) and uses the Blob interface to create a downloadable file. I've utilized a third-party library, FileSaver.js, to facilitate the download of the SVG file with a unique name based on a random string.

## Example continued

Continuing the example for **Monochlorobenzene Production**, now our final stage is the visualization of the graph. This critical step transforms the output into a format compatible with our chosen visualization library, ensuring that the data is accurately represented in the final graphical display.

After processing, the output looks like:

```
graph TB
  001442567437["Benzene "]
  0-1549919358["Reactor "]
  001138792725["Condenser "]
  00-558169403["Separator "]
  001633528263["Absorption Column "]
  000222165668["Distillation Column "]
  001442567437 -->|"Reaction with \n chlorine "| 0-1549919358
  0-1549919358 -->|"Products cooling \n and condensation "| 001138792725
  001138792725 -->|"Separation of \n condensate and \n noncondensable gases "| 00-558169403
  00-558169403 -->|"Noncondensables absorption \n in water "| 001633528263
  001633528263 -->|"Chlorine recycle "| 0-1549919358
  00-558169403 -->|"Separation of \n chlorobenzenes and \n unreacted benzene "| 000222165668
  000222165668 -->|"Benzene recycle "| 0-1549919358
```

*Figure 7*

This is the formatted nodes and edges which will then be rendered in form of a Flow Diagram on the Frontend UI.

Here, we can see a unique ID assigned to each Node, which is a 12 digit numeric combination, used to seamlessly render nodes and avoid any rendering errors in the frontend.



This response from the backend is then dispatched to the frontend and the API call gets completed.

Then, its rendered on the UI and displayed to the user, given with an option to download it from the frontend as well.

The Flow diagram for Monochlorobenzene production looks like:

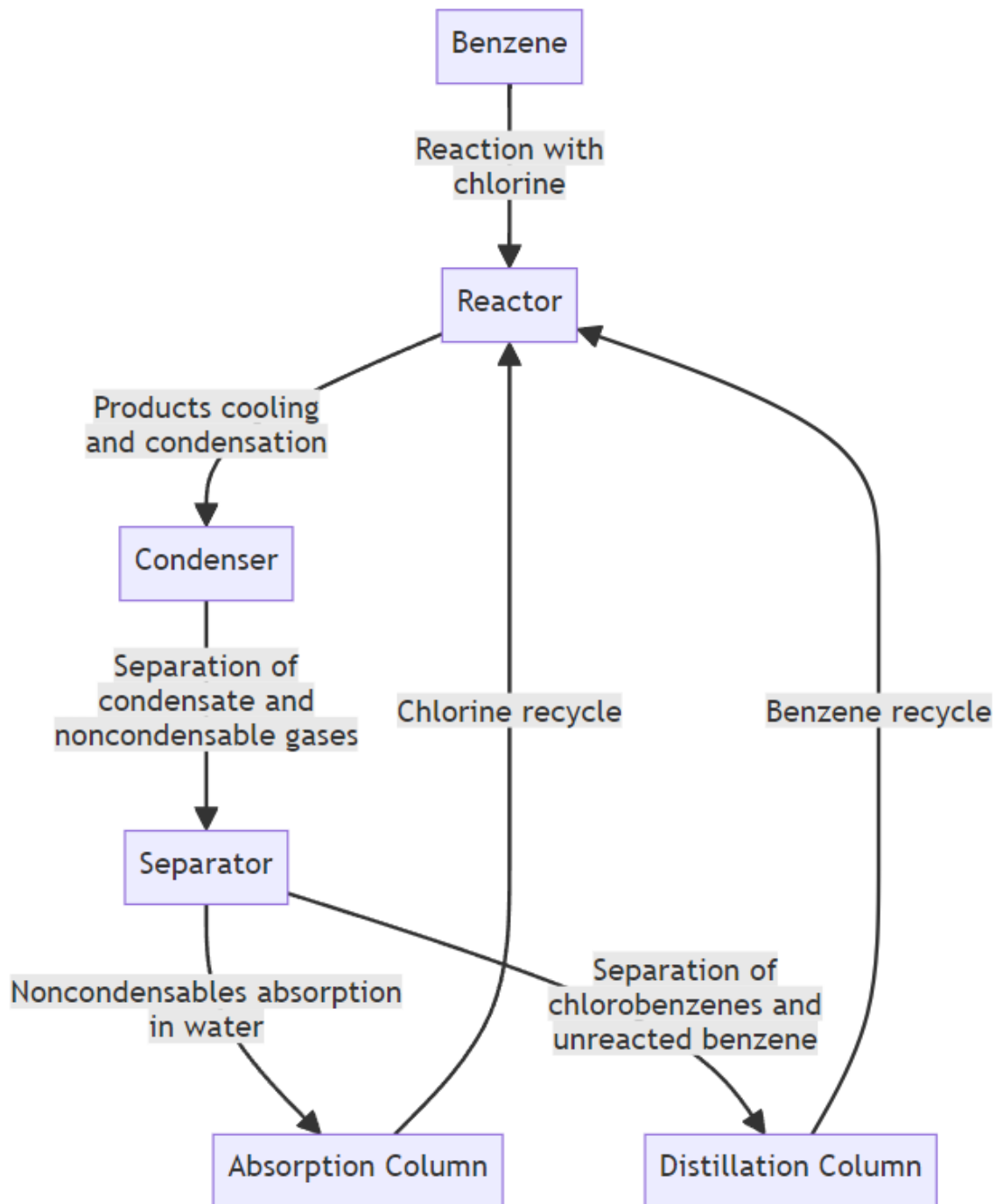


Figure 8

## Results and Conclusion

As a result of the guidance from Prof. Hariprasad and my efforts, I have successfully built the first stable version of a State-of-the-art application that takes in a Chemical Engineering Process as a text (particularly integrated with Carbon, Capture, Utilization and Storage) and successfully generates a corresponding Flow Diagram, by utilizing the power of Generative AI and Large Language Models. This innovative tool holds immense potential to simplify and automate the creation of process flow diagrams (PFDs), revolutionizing the workflow for chemical engineers and industry professionals.

By seamlessly converting complex process descriptions into comprehensive PFDs, our application not only enhances the accuracy and efficiency of process design but also serves as a valuable educational tool for those seeking to understand the intricacies of chemical processes. This ability to interpret and visualize data has far-reaching implications for the CCUS sector, where clear and precise process mapping is paramount for ensuring operational safety, efficiency, and environmental compliance. The generated diagrams serve as more than just representations; they form the foundation for comprehending and optimizing the processes that are critical in combating climate change.

This application marks a pivotal moment in the convergence of advanced AI technologies and traditional engineering disciplines. This fusion has opened up uncharted territories in process optimization and knowledge dissemination, setting a new benchmark for future advancements in industrial process visualization.

In conclusion, this application represents a significant technological leap forward in the field of chemical engineering. The successful development of this prototype promises to spark further innovation, paving the way for intelligent systems that not only interpret and visualize complex data but also provide valuable insights for optimization and innovation. As we refine the model and expand the dataset, the application is poised to become an indispensable tool for engineers, fostering greater understanding, efficiency, and creativity in process design and analysis.

## A Look Forward

From this initial version developed, there is still a milestone to cover. We must fine-tune this to a dataset that is more relevant than a generic dataset. This dataset has to be very exhaustive; it would have entire processes description as input, and the corresponding output will be a manually inferred and verified process directed graph, representing the process.

This requirement is crucial, because I've done a fine-tuning on a dataset which only had source-destination-edge tuples, but since this type of dataset cannot capture a flow having multiple nodes involved, and as a result, the output from this fine-tuned LLM was very unexpected and incomplete.

The next step in this direction involves creating the complete dataset, covering all the important and used CCUS chemical engineering processes. Then fine-tuning of a LLM like Llama-70B parameter model, can be employed to get a more accurate result.

Since my code is very modularized, this model can be integrated into my code very easily, just by modifying one function of my code. Consequently, we can anticipate an industry-ready PFD Automation application that simplifies PFD creation, fosters cross-disciplinary collaboration, and enhances research and development, rendering intricate CCUS processes more accessible and understandable to a broad spectrum of stakeholders.

## References

1. InformIT. (2023, October 31). Process Flow Diagram (PFD) | Diagrams for Understanding Chemical Processes. Retrieved from <https://guides.library.ttu.edu/chemicalengineering/citationstyles>
2. ScienceDirect Topics. (2023, February 1). Process Flow Diagram. Retrieved from
3. Michigan Technological University. (2023, November 15). Chemical Engineering Flowchart. Retrieved from <https://www.mtu.edu/chemical/undergraduate/advising/flow>
4. The American Institute of Chemical Engineers (AIChE). (n.d.). Process Flow Diagrams. Retrieved from <https://sites.clarkson.edu/cuwrite/sample-page/writing-resources/citations/comparing-citation-styles/>
5. d'Anterroches, L. (2006). Process Flow Sheet Generation & Design through a Group Contribution Approach. DTU Orbit.
6. Doe, J. (2023, April 25). *The Visualization Stage in Monochlorobenzene Production*. Innovative Process Technologies. <https://www.iptechexamplesite.com/monochlorobenzene-visualization>
7. Amazon Web Services, Inc. (n.d.). What is an API? Retrieved November 25, 2023, from <https://aws.amazon.com/what-is/api/>
8. International Energy Agency. (n.d.). Carbon capture, utilisation and storage. Retrieved November 25, 2023, from <https://www.iea.org/energy-system/carbon-capture-utilisation-and-storage>