Prompt Engineering For Water Network Design and Analysis

A Project Report
submitted by

GLEN PHILIP SEQUEIRA (DA24C005)

Under the guidance of

PROF. SRIDHARAKUMAR NARASIMHAN

in partial fulfillment of the requirements
for the award of the degree of

MASTER OF SCIENCE

DEPARTMENT OF DATA SCIENCE AND AI
INDIAN INSTITUTE OF TECHNOLOGY MADRAS

January, 2025

Abstract

The project is based on the design and development of an interactive chatbot system for a Designer app. The app helps to design any given water distribution network, branched or looped, using the successive quadratic optimization technique. The app allows the user to import, visualize, solve and modify any given water distribution network. Users can estimate the cost of a water network by providing inputs to the designer app through the chatbot system. The numerical values are extracted from the user's input by doing LLM-assisted input parsing. Data validation is handled using a Pydantic data model, ensuring that the extracted inputs conform to the defined data-type. Users can also ask the chatbot questions about the functionality and features of the app, and it will provide appropriate answers. The chatbot system is powered mainly by a Large Language Model (LLM), the Llama3 70B 8192 model, accessed using the Groq API. Parameters like `temperature` and `max_tokens` are varied in the LLM request and well-structured customized prompts are used to obtain suitable results from the LLM. The LLM is used to generate user-friendly conversations and to handle user's questions about the app. It is also used to determine the intent of the user in various cases like confirming the final data, specifying materials, determining whether user wants to use default value or re-enter the input, etc. The project concludes with the implementation of a chatbot system for the designer app, which estimates the cost of the network, compares the cost by replacing the material of the pipes, handles queries about the app, performs invalid value handling and provides the user the option to confirm the data or edit any input.

Acknowledgements

I would like to express my heartfelt gratitude to my project guide, Prof. Sridharakumar Narasimhan, for providing me with the winter project opportunity. I sincerely thank him for guiding and supporting me throughout the course of the project. I am deeply thankful to Mr. Sam Mathew for his constant support, which allowed me to complete my winter project smoothly. Also, I would like to extend my sincere thanks to Ms. Hemasree, Mr. Sumanth and Mr. Tarak for their support and collaboration. Finally, I am grateful to everyone who contributed, directly or indirectly, to this project, making it a great learning experience for me.

Contents

1. Introduction

With the rise in popularity of AI powered chatbots like ChatGPT, Gemini, Perplexity,etc., people are increasingly interested to interact with applications in a conversational manner. These chatbots leverage Natural Language Processing (NLP) to bridge the gap between human communication and computational problem solving. Chatbot systems can utilize Large Language Models (LLMs) to understand and respond to human queries in a natural and context-aware manner.

The project I will be working on is part of an ongoing project to develop applications for the analysis and design of water distribution networks. There are five applications which are under development currently:

- Sensor Placement: It helps in placing sensors to measure flow and pressure in a water network.
- Scheduler: This app is used to schedule valve operations in an optimized manner to meet the required demands.
- Designer: It is used to design a given branched or looped water distribution network.
- Calibration: It is used to calibrate a water network.
- Leak Detection: This app is used to detect leaks in a water network.

The designer app helps to design any given water distribution network - branched and looped networks. Users can import the network by uploading the input file as an IPANET native format. The network can handle multiple reservoirs and demand junctions. The lengths and diameters of the water network are determined through the design. The app designs the network by assigning diameters to the whole pipe or split pipes from a user-defined set of diameter options. While designing the network, the app ensures that minimum pressure at each junction is maintained. Users have the option to visualize the network to gain a better understanding about the design of the network. While solving the network i.e. estimating the cost of the network, users can specify the approach (diameter or lengths of the pipes) and number of iterations.

2. Goal

The main goal of the project is to design and develop a chatbot system for the Designer app. The possible use cases of the chatbot are:

- Users can estimate the cost of the network by providing the inputs through the chatbot system
- Users can compare the cost of the network by replacing the material of the pipes.

After designing an interactive chatbot system for the designer app, this idea can be extended to other apps to improve user experience.

3. Approach

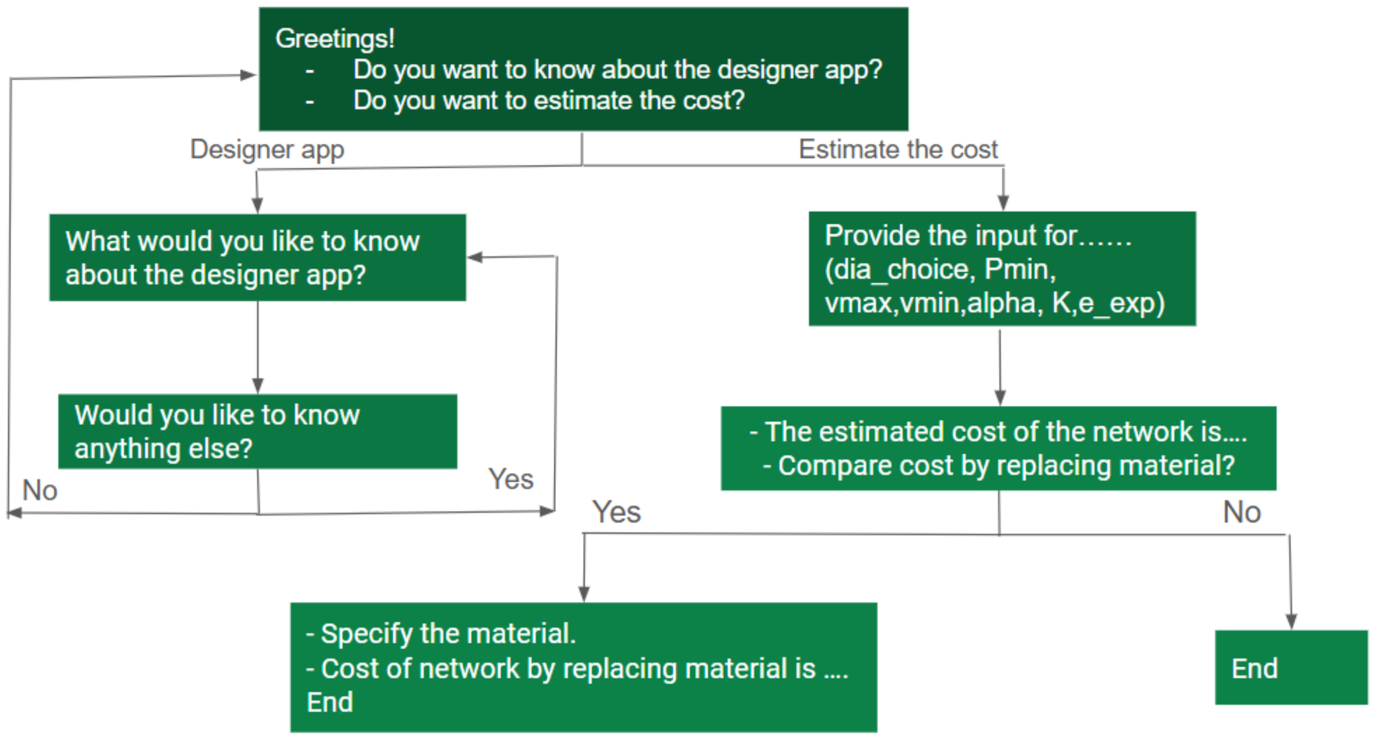

Greetings!
- Do you want to know about the designer app?
- Do you want to estimate the cost?
Designer app
Estimate the cost
What would you like to know about the designer app?
Provide the input for...... (dia_choice, Pmin, vmax,vmin,alpha, K,e_exp)
Would you like to know anything else?
No
Yes
- The estimated cost of the network is....
- Compare cost by replacing material?
Yes
No
- Specify the material.
- Cost of network by replacing material is ....
End
End

FIGURE 1.

The chatbot system is designed based on the logic flow illustrated in Figure 1. When the user clicks the chatbot icon, a chatbot window opens, greeting the user and offering two options:

(1) Do you want to know about the designer app?
(2) Do you want to estimate the cost of a water network?

Option 1: Know about the designer app:
If the user chooses to learn about the designer app, the chatbot asks what specific information the user would like to know. Based on the user's query, the chatbot provides relevant answers. After answering, the user is asked if they would like to know anything else.

- If the user says yes, the chatbot again asks what they would like to know about the app.
- If the user says no, the chatbot returns to the main page.

Option 2: Estimate the Cost of the Water Network
If the user opts to estimate the cost of the water network, the chatbot prompts the

user to input values for various parameters. Inputs are categorized as mandatory or non-mandatory:

Mandatory Fields:

- Choice of diameters (dia_choice)
- Minimum pressure (P_min)
- Minimum velocity (v_min)
- Maximum velocity (v_max)

If the user enters an invalid value for any mandatory field, the chatbot will request the user to re-enter the value.

Non-Mandatory Fields:

- Alpha: Exponent of the diameters in the objective function
- K: Constant in the objective function for cost
- E_exp: Exponent in the Hazen-Williams equation

If the user enters an invalid value for a non-mandatory field, the chatbot offers the option to either re-enter the value or use the default value. Additionally, the user can specify material-specific constants by providing the name of the material.

Data Confirmation and Cost estimation

Once all input data is collected, the chatbot displays the final collected data and asks the user to confirm or update any field:

- If the user confirms the data, the chatbot calculates and displays the estimated cost of the water network.
- If the user chooses to update any field, they can modify the specific values before proceeding.

Material Comparison

After displaying the estimated cost, the chatbot provides the user the option to compare the cost of the water network using a different material. The user provides the name of the new material, and only the material-specific constants are replaced. The chatbot then recalculates and displays the cost of the network with the replaced material.

Implementation

To implement this chatbot system, the LLM model `llama3-70b-819` is utilized. The model is employed across various tasks throughout the chatbot implementation. Efficient use of the LLM is achieved through prompt engineering, which involves creating customized prompts and choosing suitable parameters for LLM requests to ensure desirable outputs. To ensure data extraction and validation, the chatbot incorporates LLM-assisted input parsing and data validation using Pydantic models. These aspects are discussed in detail in the following sections.

4. Prompt Engineering

Prompt engineering is a crucial aspect while working with LLMs because it decides the behaviour and accuracy of the model's response. A well-structured prompt makes sure that the model understands the task, adheres to specific instructions and produces suitable, contextually relevant responses. Precise and customized prompts guide the model to focus on the specific requirements of a task. Customized prompts help to optimize the format, tone and content of the output, ensuring that it aligns with the expectation of the user. In the chatbot system that I have implemented, prompt engineering is done in several ways:

Customized Prompts

Throughout the code, well-structured and tailored prompts are used for various use cases to generate specific and relevant outputs. Precise instructions are provided in the prompt so that the LLM generates desirable outputs.

For example, if the user enters an invalid value, the chatbot provides the option to either re-enter the value or use the default value. When choosing an option, the user may respond flexibly, expressing their intent without necessarily using specific keywords. To determine whether the user's intent is to use the default value or re-enter a new one, the following prompt is used:

Prompt A: "You are an assistant helping to interpret user input. The user may respond in various ways. Analyze the response and determine whether the user wants to use the default value or re-enter the input. Provide one word as output: 'default' if the user intends to use the default value, or 'reenter' if the user wants to re-enter the input. Input: "

Suitable Choice of Parameters

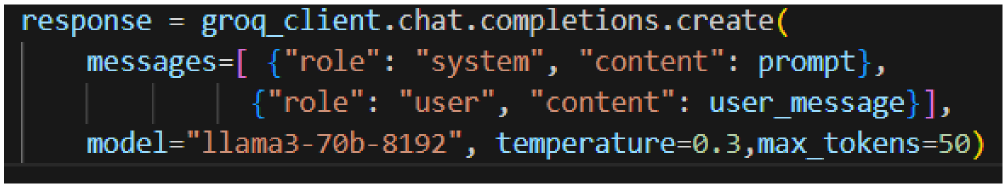
The parameters in the LLM request like `temperature` and `max_tokens` can be chosen appropriately based on the type of task and expected response style:

- `temperature` controls the randomness of the model's output. Lower values (closer to 0) make the responses more consistent and deterministic, while higher values (closer to 1) increase creativity and variability in the generated text.
- `max_tokens` specifies the maximum number of tokens the model can generate in its response. Lower values generate shorter responses while higher values lead to detailed and lengthier outputs.

For example: For Prompt A mentioned above, a suitable value of `temperature` and `max_tokens` is 0.3 and 10 because we expect to get consistent responses which is short, either 'default' or 'reenter'. The use of these parameters is shown in Figure 2

Use of role assignments

In the LLM request, the parameter `role` specifies the role of the entity sending the message. `role: system` represents instructions or setup information provided to the model, used to establish context, define behavior, or provide rules to the LLM. `role:user` contains the actual question, request, or command the user provides, prompting the model to generate a response. The use of role assignments in the LLM request is shown in Figure 2



```
response = groq_client.chat.completions.create(
    messages=[ {"role": "system", "content": prompt},
               {"role": "user", "content": user_message}],
    model="llama3-70b-8192", temperature=0.3,max_tokens=50)
```

FIGURE 2.

Using chat history in context management
Throughout the implementation of the chatbot, the interaction between the user and the chatbot is recorded as chat history. We can include the last 2-5 exchanges in the prompt to balance contextual balance and response latency. Including chat history in the prompt helps maintain context and continuity across user interactions, making the LLMs responses more coherent and contextually relevant.

5. Data Extraction and Validation

Data validation prevents errors and inconsistencies by verifying that the input data adheres to the expected format, type, and constraints. Ensuring a robust data extraction and validation process is essential, as the data provided by users through the chatbot will directly impact the functioning of the designer app. For instance, when a user enters the minimum velocity, the chatbot system should ensure that the value is converted into a standard unit, such as meters per second (m/s), which the designer app can process. Additionally, the chatbot should validate that the entered value is within a reasonable range and does not exceed the maximum velocity, ensuring consistency and accuracy for further use in the designer app.

In the designer app, some fields like the choice of diameter, minimum pressure, minimum velocity, and maximum velocity are marked as mandatory. If the user enters an invalid value for these fields, an error message is displayed, and the user is prompted to reenter a valid value, as these fields are required. For non-mandatory fields, the user has the option to either reenter a value or accept the default value. Additionally, for material-specific constants, the user can specify the name of the material, after which the chatbot will refer to a lookup table to retrieve and use the relevant values for the constants.

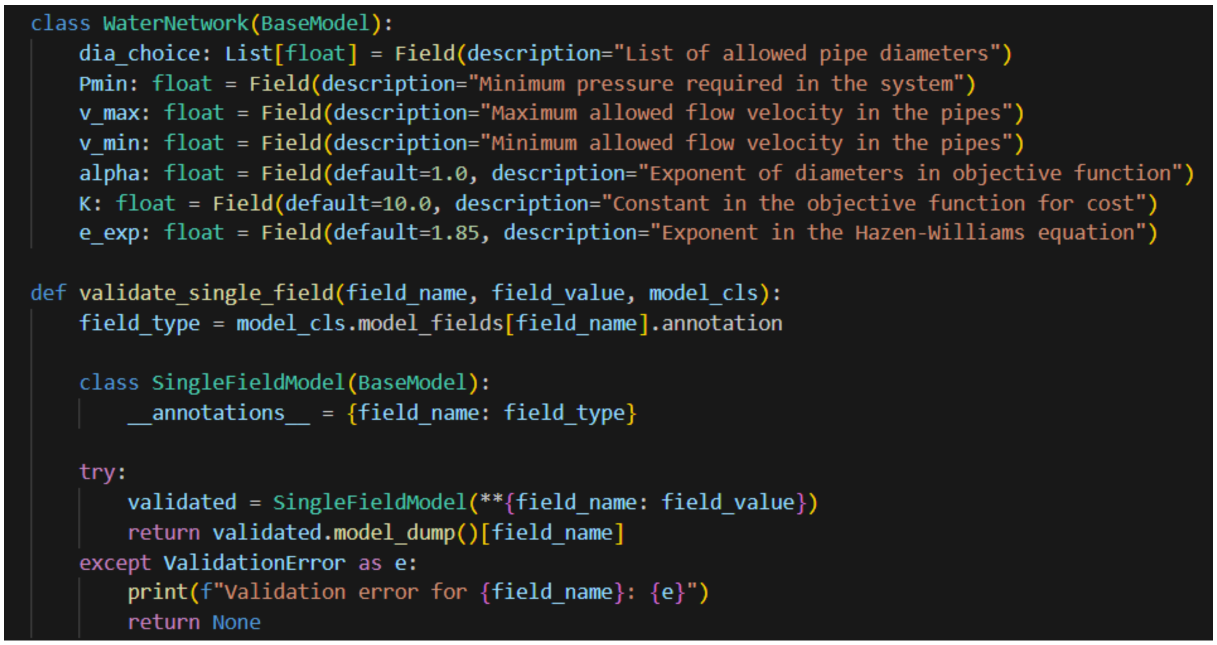
The chatbot asks the user to provide input values for various parameters of the water network. The user enters the input text, from which the LLM extracts the numerical values. Customized prompts are used in the LLM request to ensure that the value is converted to standard units and the numeric value is returned in JSON format. The extracted value is then validated using Pydantic model, which specifies the description, data type and description of the parameter which is being extracted.

LLM assisted input parsing

The LLM, `llama3-70b-8192` is used to extract numerical values from the user's input text based on customized prompts. In the LLM request, `system` and `user` roles are specified.

- The prompt of `role:system` provides context to the LLM by instructing it to assist in a water network analysis task. It ensures the extraction and conversion of numerical values from user's input, whether expressed as digits or words, into numeric form. If input is invalid, the LLM should return `"invalid": true`
- The `role:user` specifies a field-specific instruction prompt. For example, for maximum velocity, the LLM is instructed to convert the user's input into meters per second (m/s) and return the value in this unit. If the user does not specify a unit, the LLM assumes the input is in the standard unit (m/s in this case). The output is formatted strictly as a JSON object to ensure seamless data validation.

The LLM is queried using the `groq_client.chat.completions.create()`. In the LLM request, `temperature` is set as 0.2 because we intend to get a deterministic output. The response from the LLM is processed to extract a JSON object using regular expressions and parsed into a Python dictionary. The extracted value is validated using a Pydantic model. If the output of the LLM is `invalid` or the validation fails, the user is provided with the option to reenter or to use the default value, depending on whether it's a mandatory field or not.



```python
class WaterNetwork(BaseModel):
    dia_choice: List[float] = Field(description="List of allowed pipe diameters")
    Pmin: float = Field(description="Minimum pressure required in the system")
    v_max: float = Field(description="Maximum allowed flow velocity in the pipes")
    v_min: float = Field(description="Minimum allowed flow velocity in the pipes")
    alpha: float = Field(default=1.0, description="Exponent of diameters in objective function")
    K: float = Field(default=10.0, description="Constant in the objective function for cost")
    e_exp: float = Field(default=1.85, description="Exponent in the Hazen-Williams equation")

def validate_single_field(field_name, field_value, model_cls):
    field_type = model_cls.model_fields[field_name].annotation

    class SingleFieldModel(BaseModel):
        __annotations__ = {field_name: field_type}

    try:
        validated = SingleFieldModel(**{field_name: field_value})
        return validated.model_dump()[field_name]
    except ValidationError as e:
        print(f"Validation error for {field_name}: {e}")
        return None
```

FIGURE 3.

Data Validation using Pydantic model

Pydantic is a Python library for data validation that leverages Python's type annotations. It ensures that data matches specified types and automatically converts inputs to the expected format when possible. By raising detailed validation errors, it simplifies debugging and enforces data integrity. Pydantic models are easy to customize using fields with default values, constraints, and descriptions. With easy JSON and dictionary conversion, it works well with APIs and configuration management, especially in frameworks like FastAPI.

The `WaterNetwork` Pydantic model is a structured data model, which specifies the description, default values and data types of the input parameters of the Designer app. Each input field is defined with a clear and descriptive description, which helps in understanding the purpose of each parameter. Certain fields, especially that of constants are defined with default values. TThe WaterNetwork model performs data validation by ensuring that the data provided for a specific field adheres to the data type and constraints defined for that field in the model. Another advantage of using the Pydantic model is its ability to validate multiple values for a single input field. For example, defining `dia_choice` as `List[float]` allows flexibility in specifying and validating multiple allowed diameters in a single input.

Since the input parameters are provided by the user one at a time, it is more preferable to dynamically create a temporary model (`SingleFieldModel`) to validate a single field from the `WaterNetwork` model. Figure 3 illustrates the creation of single field Pydantic models from the `WaterNetwork` model. The `SingleFieldModel` is created using the type annotation of a specific field from the `WaterNetwork` model. By assigning the `field_type` to the `SingleFieldModel`, the single-field validator applies the same validation rules defined in the `WaterNetwork` model for that specific field. If the input value extracted fails to meet the requirements of the field, `ValidationError` will be raised.

6. Large Language Model (LLM)

6.1. Choice of LLM. :

Several LLMs are available through API platforms such as HuggingFace and Groq. These models differ in key aspects, including the number of parameters, context size, performance, and the number of free requests provided. Initially, I attempted to implement the chatbot system using Mistral-7B-v0.1 and Mixtral-8x7B-Instruct-v0.1. However, these LLMs did not deliver desirable results, even after using customized prompts and fine-tuning hyperparameters. Then, I used llama3-70b-8192 via Groq API to design the chatbot system. This LLM model produced efficient results through customized prompts and fine-tuned parameters. This model consists of 70 billion parameters and has a context length of 8,192 tokens. Moreover, the API allows for 30 free requests per minute and 14,400 free requests per day, which is sufficient for our use-case.

6.2. Application of the LLM in the chatbot system. :

The Large Language Model, llama3-70b-8192 is used throughout the functioning of the chatbot system to deliver an interactive user experience. The LLM ensures that a wide variety of users can use the designer app easily through the chatbot system through user-friendly conversations. The LLM is used for the following tasks in the chatbot system:

Extracting Numerical values with unit handling

The chatbot system is designed to handle user inputs flexibly for various fields, such as diameter, minimum pressure, minimum velocity,etc. Users can provide inputs in any format,for example: if the chatbot asks to provide the input for maximum velocity, the user can enter the input as 'sixteen', '15 inch/s', or 'the velocity should not exceed 10 metres per second'. To process these inputs, the chatbot sends them along with a customized prompt to the LLM. The prompt explicitly instructs the LLM to extract the numerical value from the user's input after converting it into SI units. For instance, if the input is "15 inch/s", the system converts it to meters per second (m/s) and returns the value in the format `v_max`: 0.381 m/s. If no units are specified in the user's input, the value is assumed to already be in SI units.

Determing the intent of the user's utterances:

When a chatbot asks the user a question that requires selecting from several options, the user may choose one of the options but might not respond using the exact keywords the chatbot anticipates. To address this, the LLM is used to determine which option the user wants to choose by analyzing the intent of the user's responses. The user's response is combined with the original question and the LLM is instructed to return specific keywords by analyzing the intent of the user's response. The keyword returned depends on what option the user likely intended to choose.For instance, if the chatbot asks, "Would you like to provide a new value or shall I proceed with the default value since the input provided is not valid?" the user may respond in various ways, such as "I would like to enter again," "Proceed with the default value," "I want to reenter," or "Use default value." The prompt sent in the LLM request is customized to analyze the user's intentions and return the keyword "reenter" if the user wishes to input a new value, or "default" if they prefer to proceed with the default value.

Generating variations in the chatbot's dialogues

To ensure that chatbot conversations feel natural and varied, repetitive dialogues should be avoided, as they can negatively impact user experience. To address this, the chatbot leverages an LLM to introduce variability in its responses. A 'base sentence' is first defined, and a custom prompt is then used to instruct the LLM to generate variations of the sentence while preserving its original intent.

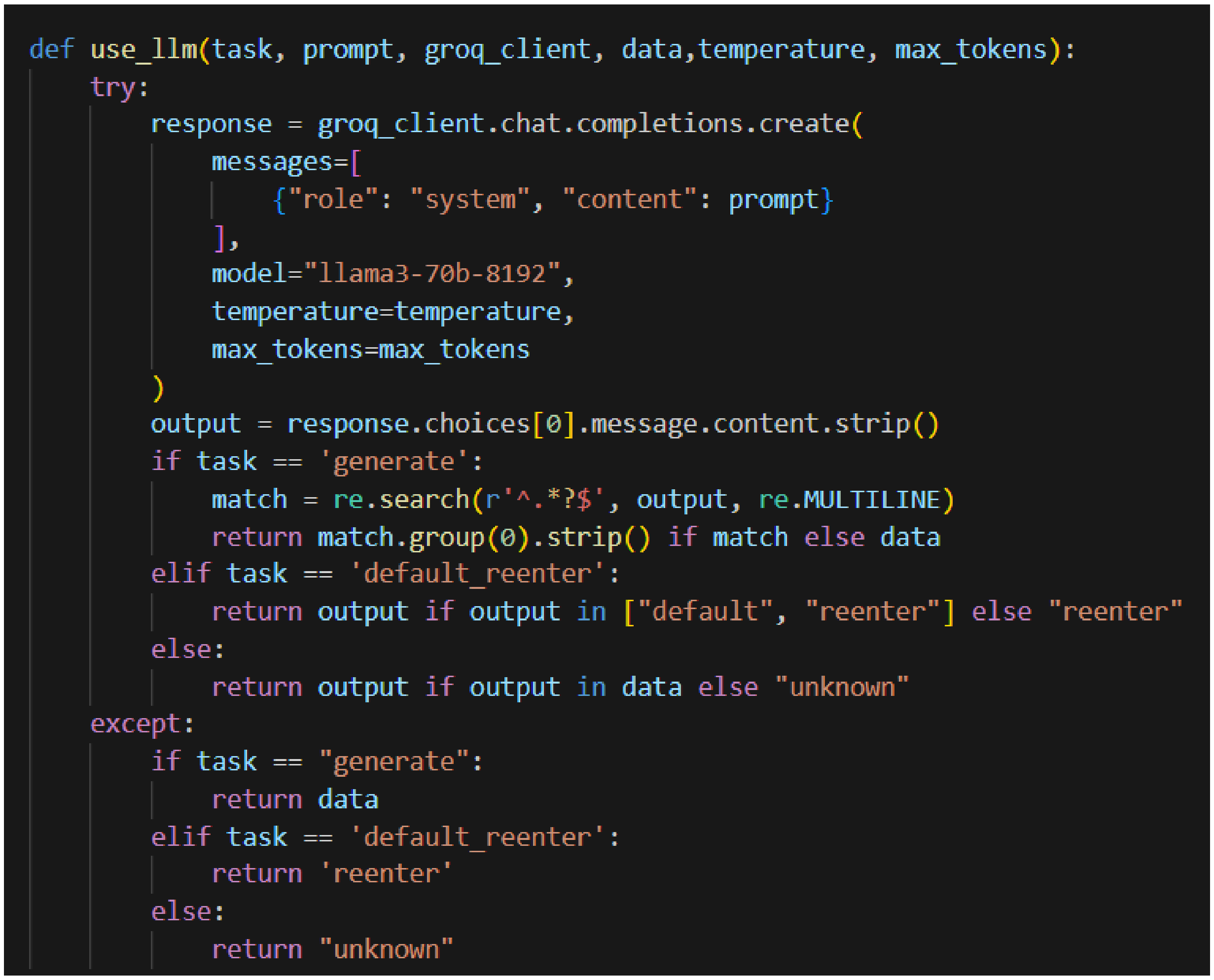
For example, if the user provides invalid input, the chatbot uses the LLM to create a variation of the sentence:

"The input provided is invalid. Since this input is not mandatory, should I use the default value, or would you like to provide a new value?"

Handling user's queries regarding the designer app

To ensure the chatbot accommodates a diverse range of users, including those unfamiliar with the app's features and functionality, the system is designed to answer user questions about the designer app. A file named designer.txt contains a detailed description of the app, its features, and its functionality.

When a user asks a question, the contents of designer.txt are provided as context to the LLM. The user's question is appended to the LLM prompt, and the LLM is instructed to generate a response based on the information in the file. This setup allows users to ask questions like: "What is alpha?", "How does the app work?,"What inputs does the app take?".



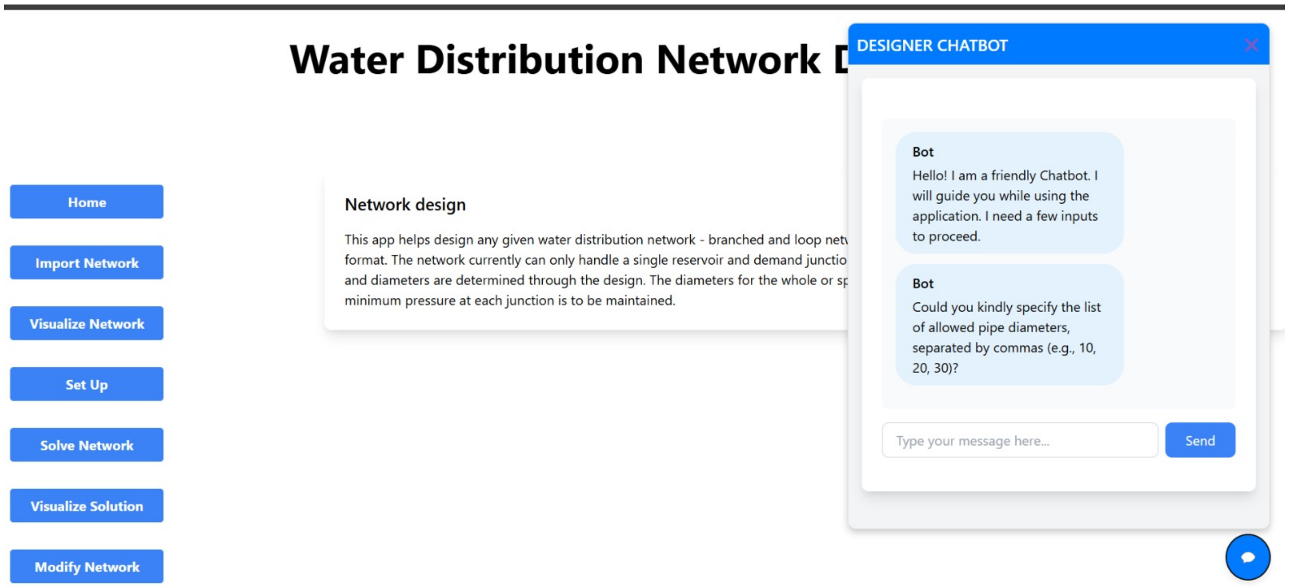
```
def use_llm(task, prompt, groq_client, data,temperature, max_tokens):
    try:
        response = groq_client.chat.completions.create(
            messages=[
                {"role": "system", "content": prompt}
            ],
            model="llama3-70b-8192",
            temperature=temperature,
            max_tokens=max_tokens
        )
        output = response.choices[0].message.content.strip()
        if task == 'generate':
            match = re.search(r'^.*?$', output, re.MULTILINE)
            return match.group(0).strip() if match else data
        elif task == 'default_reenter':
            return output if output in ["default", "reenter"] else "reenter"
        else:
            return output if output in data else "unknown"
    except:
        if task == "generate":
            return data
        elif task == 'default_reenter':
            return 'reenter'
        else:
            return "unknown"
```

FIGURE 4.

7. Conclusions and Future Work

A chatbot system has been designed and integrated with the designer app. Through the chatbot system, users can provide the inputs of the water network in a conversational manner. Data validation is handled smoothly by using a Pydantic data model and LLM assisted input parsing and validation. Invalid values are handled by the chatbot by providing the user the option to re-enter or use the default value. For material specific inputs, the chatbot provides the user the otpiton to specify the material name, after which the chatbot will use a look-up table and use the corresponding constant values.The chatbot can also handle user's queries about the app. The user can ask questions about the working, features or any specific detail of the app, to which the chatbot will provide an appropriate answer. After all the inputs are collected, the chatbot provides the user to confirm the final collected data or edit any input field. After providing confirmation, the cost of the network is calculated and displayed. After this, the user has the option to compare the cost of the network by replacing all the pipes of the network with another material.

There is scope to develop the project further by introducing additional features in the chatbot system. For instance, users can have the option to modify the water network, say changing the demand at a particular node or all the nodes. Additionally, through the chatbot users could specify their preferred approach for solving the water network, such as defining the number of iterations or choosing whether to solve based on the diameters or lengths of the pipes. Also, chatbot systems can be introduced in the other water design and analysis apps, namely Scheduler, Sensor Placement, Calibration and Leak Detection.



Network design

This app helps design any given water distribution network - branched and loop netw
format. The network currently can only handle a single reservoir and demand junctio
and diameters are determined through the design. The diameters for the whole or sp
minimum pressure at each junction is to be maintained.

Bot
Hello! I am a friendly Chatbot. I will guide you while using the application. I need a few inputs to proceed.

Bot
Could you kindly specify the list of allowed pipe diameters, separated by commas (e.g., 10, 20, 30)?

Figure 5.

8. Learning Outcomes

The implementation of the chatbot system for the Designer app helped me to improve my skills and knowledge. Through this project, I gained insights into key concepts related to water distribution networks, the effective use of large language models (LLMs) and techniques for data validation and extraction:

- I gained knowledge about water distribution networks, including their design and analysis. I learned that the design problem of water networks can be approached in several ways. Specifically, I worked on building a chatbot for the Designer app, which uses a non-linear programming technique called the Successive Quadratic Optimization method to solve the network design problem. I learnt about the different input parameters required for a water distribution network.

- I explored different LLM models and observed differences in their behavior and responses. I also learned how to utilize LLMs for various tasks using APIs. I was able to select a suitable LLM model, llama3 70B 8921 via Groq API, to implement the chatbot system.
- I realized the importance of using customized and well-formatted prompts in generating suitable outputs from the models. I learned that more elaborate and context-rich prompts give desirable outputs. I learned how to select appropriate values of parameters like `temperature` and `max_tokens` in the LLM request.

- I learned about data validation and extraction. I realized that data can be efficiently extracted from a user's input using LLM assisted input parsing by specifying customized prompts. The extracted values can be validated with the help of Pydantic models which help to make sure that the values extracted conform to the schema(data-type) defined in the Pydantic model.

References

[1] K. Vasant Kumar Varma, Shankar Narasimhan, and S. Murty Bhallamudi. Optimal Design of Water Distribution Systems Using an NLP Method. *Journal of Environmental Engineering, ASCE*, vol. 123, no. 4, 1997, pp. 381–388.

[2] Pydantic documentation. https://docs.pydantic.dev/latest/.

[3] Llama Documentation. https://www.llama.com/.

[4] Pydantic model documentation. https://pypi.org/project/pydantic.

[5] Building an Intelligent chatbot system with LLMs. https://mrmaheshrajput.medium.com/how-to-build-an-intelligent-qa-chatbot-on-your-data-with-llm-or-chatgpt-d0009d256dce.

[6] Using the Llama3 70B model. https://groq.com/introducing-llama-3-groq-tool-use-models/.

[7] Groq-API. https://console.groq.com/docs/overview.

[8] Python JSON documentation. https://docs.python.org/3/library/json.html.

[9] Mistral-7B-v0.1 model, Hugging Face. https://huggingface.co/mistralai/Mistral-7B-v0.1.

[10] Building an LLM chatbot with Langchain https://roluquec.medium.com/building-a-context-aware-llm-chatbot-with-langchain-996d372cedbb.

[11] Langchain documentation. https://python.langchain.com/docs/introduction/

[12] Building a RAG based chatbot with langchain. https://medium.com/credera-engineering/build-a-simple-rag-chatbot-with-langchain-b96b233e1b2a

[13] Chatbot implementation with Langchain. https://github.com/shashankdeshpande/langchain-chatbot