

# STEERING LLM THROUGH PROMPT ENGINEERING: A FULL STACK CHATBOT

A Paper  
Submitted to the Graduate Faculty  
of the  
North Dakota State University  
of Agriculture and Applied Science

By  
Bekalue Kobro

In Partial Fulfillment of the Requirements  
for the Degree of  
MASTER OF SCIENCE

Major Department:  
Computer Science  
Option: Plan-B

March 2025

Fargo, North Dakota

North Dakota State University  
Graduate School

---

**Title**

STEERING LLM THROUGH PROMPT ENGINEERING: A FULL  
STACK CHATBOT

---

**By**

Bekalue Kobre

---

The Supervisory Committee certifies that this *disquisition* complies with North Dakota  
State University's regulations and meets the accepted standards for the degree of

**MASTER OF SCIENCE**

SUPERVISORY COMMITTEE:

Dr. Anne Denton

---

Chair

Dr. Jeremy Straub

---

Dr. Megan Orr

---

Approved:

04/14/2025

---

Date

Dr. Simone Ludwig

---

Department Chair

## **ABSTRACT**

Large Language Models (LLMs) have enhanced chatbot capabilities, delivering realistic responses, yet they frequently neglect sustainability and miss chances to convey environmental impacts intuitively. This master's paper presents a chatbot web application leveraging the open-source LLM to create sustainability-focused, quantitative visual outputs, allowing users to make a choice depending on their level of environmental concern. The WebApp is developed using Python's Flask framework and LLM as an engine, employing prompt patterns to direct LLM into producing JSON data alongside a context control mechanism that facilitates follow-up questions and smooth interaction. The chatbot generates visualizations from JSON data for carbon footprints across selected topics, such as transportation, U.S. state emissions, and gift items, using tailored prompts. Visualizations are dynamically produced for each user prompt with Python's Plotly library, presenting carbon footprint insights intuitively. Despite limitations of hard-coded prompts, the web app yields consistent visual insights enabling users make informed choices.

## **ACKNOWLEDGMENTS**

I thank God Almighty for giving me the strength to complete this master's paper, which marks the culmination of my graduate studies in computer science. I'm also deeply grateful for the support of my family and friends, whose encouragement has been a blessing throughout my graduate academic journey. Additionally, I extend my appreciation to Dr. Anne Denton for her invaluable guidance and for the opportunity to work under her supervision.

## TABLE OF CONTENTS

ABSTRACT.....	iii
ACKNOWLEDGMENTS .....	iv
LIST OF TABLES.....	vii
LIST OF FIGURES .....	viii
LIST OF ABBREVIATIONS.....	x
1. PROBLEM STATEMENT.....	1
2. LITERATURE REVIEW .....	2
2.1. Large Language Models (LLMs).....	3
2.2. Prompt Engineering .....	4
3. IMPLEMENTATION.....	6
3.1. Requirements Specification .....	6
3.2. Environment Setup and Tools.....	7
3.3. Front-End Implementation.....	9
3.4. Back-End Implementation .....	10
3.4.1. LLAMA as Chatbot Engine .....	11
3.4.2. Query Processing.....	11
3.4.3. Prompt Engineering.....	12
3.4.4. Context Control .....	14
3.4.5. Data Extraction with Regular Expression .....	18
3.4.6. Data Visualization .....	18
3.5. Comparison Direct LLM Queries Against Chatbot Response.....	22
3.5.1. Direct LLM Query Response Comparison: Prompt1 Case-I Vs Case-II .....	23
3.5.2. Direct LLM Query Response Comparison: Prompt2 Case-I Vs Case-II .....	26
3.5.3. Direct LLM Query Response Comparison: Prompt3 Case-I Vs Case-II .....	30

3.5.4. Evaluation – LLM Direct Query Using Raw Prompts (Case-I).....	33
3.5.5. Evaluation – LLM Direct Query Using Engineered Prompts (Case-II).....	33
3.6. Comparison of Direct Query Responses Against Chatbot’s Response.....	35
4. DISCUSSION .....	36
5. LIMITATIONS.....	37
REFERENCES .....	38
APPENDIX A. SAMPLE PROMPT FOR CAR TYPES.....	40
APPENDIX B. PROMPT SELECTOR CODE.....	41
APPENDIX C. SAMPLE EXPORTED DATA .....	42

## LIST OF TABLES

<u>Table</u>	<u>Page</u>
1. Types of chatbots based on criteria.....	2
2. List of popular open source LLMs with number of parameters in billions .....	4
3. Partial list of prompt patterns for conversational LLMs.....	5
4. Chatbot requirement specification .....	6
5. List of important packages and libraries .....	7
6. List of tools and software.....	8
7. Selected prompt engineering patterns .....	13
8. Categories of prompts on transportation, CO <sub>2</sub> emission and gift items.....	13
9. Prompt and case combinations used for LLM response evaluation .....	23
10. Comparison of LLM responses to direct query using prompt-1 “which car has highest carbon footprint: gasoline or electric?”, highlights differences in output between raw and engineered prompts.....	23
11. Comparison of LLM responses to direct query using prompt-2 “what are top 10 US states by CO <sub>2</sub> emission”, highlights differences in output between raw and engineered prompts .....	26
12. Comparison of LLM responses to direct query using prompt-3 “suggest gift items for Mother’s Day”, highlights differences in output between raw and engineered prompts .....	30
13. Summary of output characteristics from direct LLM queries using raw vs engineered prompts (Case I vs Case II) .....	34

## LIST OF FIGURES

<u>Figure</u>	<u>Page</u>
1. Hierarchical classification of chatbots [7] and conversational agents based on interaction mode (text-based, voice-based), knowledge domain (open domain, closed domain), goals (task-oriented, non-task-oriented), and design approach (rule-based, retrieval-based, generative-based), illustrating the diverse patterns and technologies in conversational system design.....	3
2. High-level architecture of a chatbot, illustrating the interaction flow between the user, front-end (HTML, CSS, JavaScript/jQuery), and back-end (Flask, OLLAMA), with arrows depicting the request and response process for seamless communication and response generation .....	9
3. Front-end interface of the LLM chatbot, featuring the header with the chatbot's title and logo, the chat dialogue displaying conversation history, and the footer with an input field and send button for user queries .....	10
4. Simplified overview of Flask modules' interaction in the chatbot's back-end, depicting the flow of HTTP GET and POST requests from the front-end UI to the back-end (Flask) which interacts with OLLAMA-hosted LLM for response generation.....	11
5. Architecture of the chatbot's back-end, illustrating the flow from HTML front-end (request page) via HTTP GET to Flask back-end's context-control and LLM for JSON-formatted response generation, processing and visualization, with feedback updating chat_history.json to enable follow-up questions and maintain topic coherence .....	12
6. Sample screenshot of the LLM chatbot prompt instructions for analyzing car transportation carbon footprints, specifying energy sources (gasoline, electric, diesel, hybrid), output formatting (JSON within triple backticks), and CO <sub>2</sub> value units (kgCO <sub>2</sub> /mile), with an example for structured data extraction .....	14
7. Context control block diagram with prompt selector and chat_history.json integration, steering the LLM to generate carbon footprint-focused outputs, with a feedback loop ensuring contextual alignment.....	15
8. Screenshot cosine similarity of user query “suggest gift items” with custom crafted prompts showing that the most relevant prompt has higher cosine similarity score.....	17
9. Diagram of the prompt selector mechanism, showing raw input text and prompt descriptions processed through an encoder and cosine similarity to select a prompt.....	18



10.	Sample output from the LLM chatbot for the query "which car has the most carbon footprint, sedan or truck?" showcasing accurate data extraction in JSON format and an intuitive bar graph visualization, highlighting the truck's higher carbon footprint compared to the sedan.....	20
11.	Visualization generated by the LLM chatbot comparing CO <sub>2</sub> emissions (metric tons) across U.S. states, identifying California as the highest emitter and Michigan as the lowest, with clear titles and labels enhancing the accurate and user-friendly representation of data.....	21
12.	Visualization by the LLM chatbot for the query “suggest gift items for mother’s day,” displaying average CO <sub>2</sub> footprints (kgCO <sub>2</sub> ) of gift items (e.g., custom chocolate box, floral arrangement) and categories (e.g., fashion, home & living), with clear titles and labels highlighting sustainable options.....	22

## LIST OF ABBREVIATIONS

3H.....	Honest, Helpful, and Harmless
AI .....	Artificial Intelligence
ANN.....	Artificial Neural Networks
BERT .....	Bidirectional Encoder Representations from Transformers
CSS .....	Cascading Style Sheets
CSV.....	Comma-Separated Values
GPT .....	Generative Pre-trained Transformer
HTML .....	HyperText Markup Language
IDE.....	Integrated Development Environment
JSON.....	JavaScript Object Notation
LLaMA .....	Large Language Model Meta AI
LLM.....	Large Language Models
LSTM.....	Long Short-Term Memory Networks
PaLM2.....	Pathways Language Model 2
RegX .....	Regular Expression
RICE .....	Robustness, Interpretability, Controllability, and Ethicality
RNN .....	Recurrent Neural Networks
S2q2Seq .....	Sequence-to-Sequence

## 1. PROBLEM STATEMENT

Large Language Model (LLM) responses vary significantly depending on how the prompts are phrased. As a result, LLM outputs are as good as how well engineered the prompts are [3]. Unfortunately, many users may not be technical and unaware of prompt engineering best practices to make the best out of LLMs. In addition, raw LLM outputs may not align with sustainability objectives, missing opportunities to promote environmentally responsible decision-making. Lastly, unstructured textual responses for quantitative queries often fail to convey numerical insights effectively—users benefit more from visual representations (e.g., charts) and structured formats like JSON when applicable [1].

While extensive literature exists on prompt engineering for optimizing LLM outputs [4][7], to the best of my knowledge, there is no implementation of a chatbot that specifically focuses on *sustainability-aware prompt engineering* that ensures outputs to align decisions with environmental objectives such as less carbon footprints.

The objective of this project is to develop a chatbot web application leveraging open-source large language models (LLMs) as a back-end engine to answer user queries focusing on quantitative insights oriented towards environmental awareness in terms of carbon footprints. In addition, to apply steering mechanism for the output of LLM via feedback using prompt engineering techniques to enrich user requests to generate sustainability-oriented quantitative outputs and then to present results visually using charts and plots to make insights intuitive.

## 2. LITERATURE REVIEW

Chatbots, computer programs designed to simulate human-like text-based interactions, have evolved significantly from simple rule-based systems to advanced AI-driven models. Hussain et al. trace this progression from the 1960s with ELIZA, a rule-based system relying on keyword matching, to modern implementations that rely on AI. They categorize chatbots based on four criteria, as summarized below [7].

Table 1. Types of chatbots based on criteria

Criteria	Types of Chatbots
Interaction mode	- <i>Text based or Voice-based</i>
Goals:	<p><b>Task oriented:</b> <i>designed for specific domain and short conversations.</i></p> <hr/> <p><b>Non-Task Oriented:</b> <i>mimic conversation just like human-to-human.</i></p> <p>- <i>two types:</i></p> <p><i>I. <b>Generative based</b> – generate responses during a conversation.</i></p> <p><i>II. <b>Retrieval based</b> – learn to select responses form a repository.</i></p>
Design approach	<p><b>Rule based:</b> relies on parsing, pattern matching, and ontologies.</p> <p>For example: <i>Lexical parsing uses grammatical structure.</i></p> <hr/> <p><b>AI based:</b> Uses various types of Artificial Neural networks (ANNs) including Recurrent Neural Networks (RNNs), Sequence-to-Sequence (S2q2Seq), Long Short-Term Memory Networks (LSTMs), Large Language Model (LLM) Transformer Architectures ( LLaMa Series).</p>
Domain	<i>Open domain:</i> foundational language models
Knowledge	<i>Closed domain:</i> Models fine-tuned for a specific use case.

Figure 1 below presents a hierarchical classification framework for chatbots and conversational agents, based on four criteria: Interaction Mode, Knowledge Domain, Goals, and Design Approach, as derived from the table.

It starts with "Chatbot/Conversational Agent," branching into Text-Based and Voice-Based interaction modes, Open and Closed Domain knowledge scopes, Task-Oriented and Non-Task-Oriented goals, and Rule-Based, Retrieval-Based, and Generative-Based design approaches. The latter includes advanced neural networks like RNNs, Seq2Seq, LSTMs, and LLMs (e.g., BERT, GPT-series). This tree-like structure by Hussain, S et al [7], provides a concise overview of system types and technologies, aiding readers in understanding conversational agent diversity.

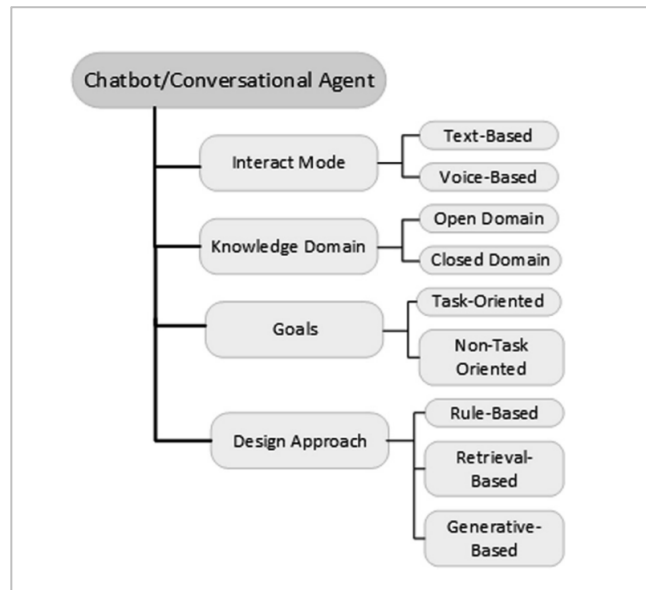


Figure 1. Hierarchical classification of chatbots [7] and conversational agents based on interaction mode (text-based, voice-based), knowledge domain (open domain, closed domain), goals (task-oriented, non-task-oriented), and design approach (rule-based, retrieval-based, generative-based), illustrating the diverse patterns and technologies in conversational system design

## 2.1. Large Language Models (LLMs)

The introduction of Google's transformer architecture marked a pivotal advancement in AI, giving rise to Large Language Models (LLMs) [9]. Trained on vast text corpora, LLMs predict subsequent words in sequences, enabling sophisticated natural language processing [3]. LLM-

based chatbots have become widely adopted, excelling in generating contextually relevant responses across diverse applications [8]. Table 2 lists notable open-source LLMs as of 2025.

Table 2. List of popular<sup>1</sup> open source LLMs with number of parameters in billions

Model Name	Parameters (in Billions of) & Versions	Company
Llama	3.1: <b>8B</b> (4.9GB) 3.2: <b>3B</b> (2.0GB) 2.0: <b>7B</b> (3.8GB)	Meta
Phi	Phi4, Phi 3 Mini	Microsoft
Gemma2	2B, 9B, 27B	Google
Mistral	7B	Mistral

## 2.2. Prompt Engineering

In a chatbot application, a text input that specifies instructions or requests to an LLM is known as a *prompt* [1]. *Prompt engineering* refers to the systematic design and optimization of prompts to guide the responses from LLMs [2][3]. Prompt engineering also plays a vital role in ensuring that the generated output meets specific qualitative and quantitative requirements [1].

In chatbot applications, prompts are text inputs that query LLMs [1]. Prompt engineering involves designing and refining these inputs to guide LLM responses effectively, ensuring they meet specific qualitative and quantitative criteria [2][3]. Several studies emphasize that the quality of a prompt significantly impacts an LLM's output, highlighting the importance of prompt engineering in fully harnessing the model's potential [1][3][5].

---

<sup>1</sup> <https://github.com/ollama/ollama>

Leo S. Lo’s CLEAR framework (Concise, Logical, Explicit, Adaptive, Reflective) provides beginners with guidelines for effective LLM interaction [6].

Jules White et al. identify 16 prompt patterns across six categories, including output customization, prompt improvement, and context control. Table 3 highlights a subset of these patterns relevant to this project.

Table 3. Partial list of prompt patterns for conversational LLMs

Category	Patterns
Output customization	<b><i>Visualization Generator Pattern:</i></b> outputs for visualization tools.
	<b><i>Template Pattern:</i></b> enable LLM output to follow a specific structure / formatting in the output.
	<b><i>Persona Pattern:</i></b> enable LLM output to reflect a certain point of view, such as job title, to get what users need without knowing details.
	<b><i>Example:</i></b> “Imagine you are a <i>security expert</i> , <request>”
Context Control	<b><i>Context Manager Pattern:</i></b> specifies context for a conversation with LLM to exclude unrelated topics.
	<b><i>Example:</i></b> “With scope X, Consider Y, ignore Z”

Prompt engineering is also central to AI alignment research, which aims to align large language models (LLMs) with human values. As LLMs grow more capable, misalignment risks rise, potentially yielding unethical outputs [10][11]. Techniques like adversarial prompting expose vulnerabilities, while few-shot prompting enhances reasoning [12]. Guided by frameworks such as RICE (Robustness, Interpretability, Controllability, and Ethicality) and 3H (Honest, Helpful, and Harmless) prompt engineering ensures LLMs remain controllable and ethical [10].

### 3. IMPLEMENTATION

The chatbot's implementation involves defining requirements, selecting tools, setting up the development environment, designing prompts for quantitative carbon footprint insights, and developing the back end and front-end components. This section details each phase of the process.

#### 3.1. Requirements Specification

Table 4. Chatbot requirement specification

Requirement	Specification
Input	- text-based interaction via natural language text queries.
Chat History	- retain chat history to answer follow up questions.
Prompt Engineering	- generate precise quantitative responses. - apply prompt engineering patterns for output formatting and role to divulge carbon footprint statistics for environmentally aware decision-making.
Response	- JSON format to enable quantitative analysis.
Data Export	- export response data to a JSON file.
Visualization	- visualize data using bar charts using exported data.
Output	- render visualization outputs of quantitative insights.
Error Handling	- provide meaningful feedback for invalid or failed queries.



### 3.2. Environment Setup and Tools

The development environment is setup using *pipenv*<sup>2</sup>, which combines package management (*pip*) and virtual environment (*venv*). This setup, defined in Pipfile and Pipfile.lock, offers robust dependency tracking compared to standalone *venv*.

Development was conducted on a Mac M1 Pro PC equipped with 32 GB of RAM with 1TB storage. The key packages installed within this environment are found on github<sup>3</sup> and detailed in Table 5.

Table 5. List of important packages and libraries

Name of Package	Use
Requests	- Communicates with OLLAMA local host at <a href="http://localhost:11434/api/chat">http://localhost:11434/api/chat</a>
Python Flask	- Routes query from HTML form to the LLM and returns responses.
Pandas	- Processes data in data frames at the back end.
sentence-transformers	- Embedding for user input and prompt patterns for cosine similarity.
scikit-learn	- Calculates cosine similarity to select optimal prompts.
Plotly	- Generates data visualizations.

Outside to the *pipenv* development environment, additional tools that are used for building and running the chatbot are listed on the table below:

---

<sup>2</sup> <https://pipenv.pypa.io/en/latest/>

<sup>3</sup> <https://github.com/BeTKH/LLM-Chatbot/blob/master/Pipfile>

Table 6. List of tools and software

Name of Tool	Use
Ollama Desktop Application	- back-end chatbot engine serving at <a href="http://localhost:11434/api/chat">http://localhost:11434/api/chat</a> - downloads various LLMs from <a href="https://ollama.com">ollama.com</a> <sup>4</sup>
HTML/CSS/JS/jQuery	- front end user interface and dynamic update based on response.
Visual Studio Code	- integrated development environment (IDE) for coding / debugging.
Git/GitHub	- Git is used for a version control and to host source code <sup>5</sup> .

Overall, the setup for building the chatbot involves using python's Flask WebApp framework at the back end, HTML/CSS/JavaScript/jQuery for the front-end, and OLLAMA as chatbot engine.

Figure 2 below illustrates the high-level architecture of a chatbot. The architecture begins with the User, who initiates the process by sending a prompt. This prompt is directed to the Front-end, a component built using HTML for page structure, CSS for formatting, and JavaScript (JS) with jQuery for dynamic UI updates. The Front-end forwards the request to the Back-end, powered by Python's Flask WebApp framework, which processes and routes the requests. The Back-end integrates OLLAMA, a chatbot engine, to generate appropriate responses. Once generated, the response is sent back through the Front-end, which updates the user interface accordingly. This bidirectional flow, depicted with arrows labeled "request" and "response," highlights the seamless communication between the user, front-end, and back-end, leveraging Flask for request handling,

<sup>4</sup> <https://ollama.com/search>

<sup>5</sup> <https://github.com/BeTKH/LLM-Chatbot>

HTML/CSS/JS for presentation, and OLLAMA for intelligent response generation, forming a robust and interactive chatbot system.

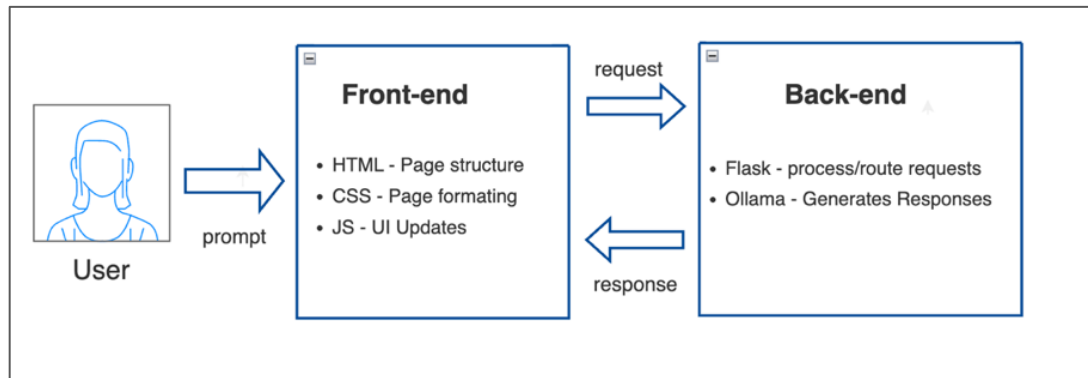


Figure 2. High-level architecture of a chatbot, illustrating the interaction flow between the user, front-end (HTML, CSS, JavaScript/jQuery), and back-end (Flask, OLLAMA), with arrows depicting the request and response process for seamless communication and response generation

### 3.3. Front-End Implementation

The front-end interface, housed within a card layout, consists of three sections:

- I. **Header:** Displays the chatbot's title and logo.
- II. **Chat Dialogue:** Shows the conversation history.
- III. **Footer:** Contains an input field and send button for user queries.

Figure 3 below depicts the front-end interface of the chatbot, consisting of three distinct sections. The Header section, located at the top, displays the chatbot's title alongside its logo. Below the header, the Chat Dialogue section occupies the central area, presenting the conversation history between the user and the chatbot, with each request and response timestamped for clarity. The Footer section, positioned at the bottom, features an input field labeled "Enter your prompt..." and a send button, enabling users to submit their queries.

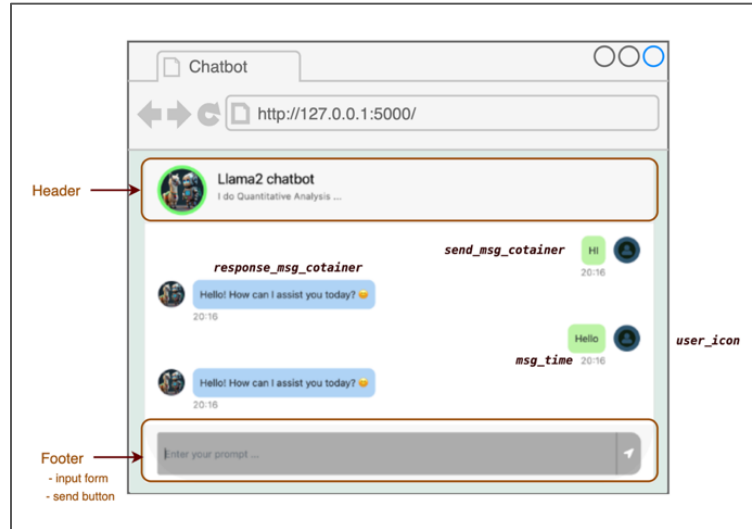


Figure 3. Front-end interface of the LLM chatbot, featuring the header with the chatbot's title and logo, the chat dialogue displaying conversation history, and the footer with an input field and send button for user queries

### 3.4. Back-End Implementation

The back-end system, developed using Flask, handles user requests submitted via HTTP GET requests through an HTML form, routing them to a large language model (LLM) hosted by Ollama.

Figure 4 below provides a simplified overview of the back-end system architecture for the chatbot, developed using the Flask framework, as described in the text. The diagram illustrates the interaction flow starting with the Front-End UI, hosted locally at `http://127.0.0.1:5000/`, where users submit prompts via an HTML form using HTTP GET requests. These requests are directed to the Flask App (`app.py`), which processes the input through its index route. The system then interfaces with the OLLAMA local host, a large language model (LLM), to generate responses. The response is routed back through the `get_chat_response()` function, which updates the chat history via `chat_history()` function, and is finally returned to the Front-End UI using a POST

request. The diagram highlights key Flask modules, including `enrich_prompt()` for steering LLM output based on the prompt patterns and `export_data()` for saving data to local storage.

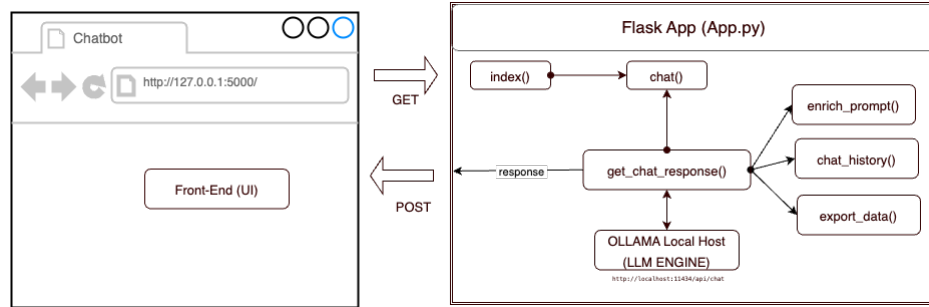


Figure 4. Simplified overview of Flask modules' interaction in the chatbot's back-end, depicting the flow of HTTP GET and POST requests from the front-end UI to the back-end (Flask) which interacts with OLLAMA-hosted LLM for response generation

### 3.4.1. LLAMA as Chatbot Engine

For this project, Llama3.3 with 70 billion parameters (Llama3.3:70b) was selected as the chatbot engine due to its state-of-the-art performance, which surpasses that of earlier LLAMA releases such as Llama2, Llama3, and Llama3.2. This choice was made based on its superior response quality and controllability when compared to earlier models like Llama2.

### 3.4.2. Query Processing

Query processing is managed by a context-control component, which selects an appropriate prompt based on cosine similarity. This component enhances the prompt by incorporating chat history, guiding the LLM to generate structured JSON-formatted carbon footprint data suitable for export and visualization. The context-control mechanism preprocesses the input before it is passed to the LLM, ensuring that the output aligns with the desired content and format.

Responses are generated token-by-token and stored in a Python list. Relevant data is then extracted using regular expressions (RegEx), exported to JSON format, and subsequently visualized.

Figure 5 below illustrates the back-end architecture for chatbot query processing. It begins with the HTML Front-End (Request Page), where user input is submitted via an HTTP GET request through an HTML Form. The Flask Back-end's Context-Control component selects a prompt using cosine similarity, enhancing it with chat history (chat\_history.json) to guide the LLM via prompt steering, producing a structured JSON response.

The Response Processing module extracts data with RegEx, exports it to JSON, and visualizes it. The HTML Front-End (Response Page) receives the output via HTTP POST. A feedback mechanism updates chat\_history.json, enabling follow-up questions and topic coherence, with arrows showing data flow.

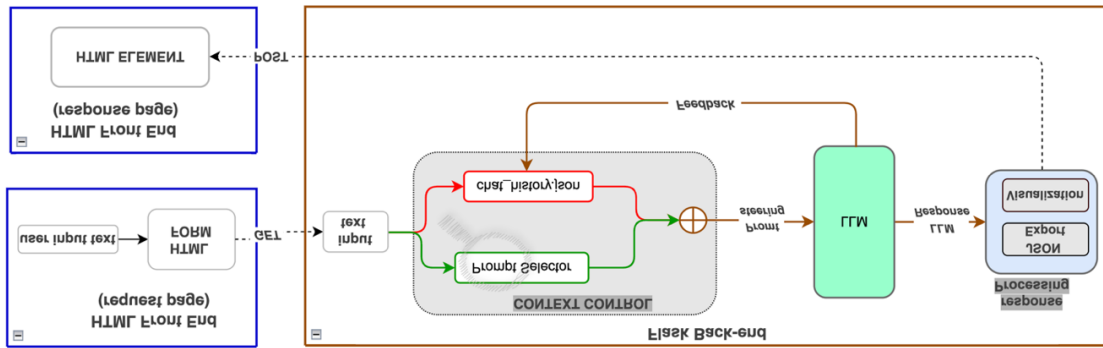


Figure 5. Architecture of the chatbot's back-end, illustrating the flow from HTML front-end (request page) via HTTP GET to Flask back-end's context-control and LLM for JSON-formatted response generation, processing and visualization, with feedback updating chat\_history.json to enable follow-up questions and maintain topic coherence

### 3.4.3. Prompt Engineering

To extract quantitative insights related to carbon footprints, prompts are carefully designed using selected *prompt engineering patterns* from the literature. The three chosen prompt engineering patterns are the *template pattern*, *persona pattern*, and *context manager pattern* which are detailed on the table below:

Table 7. Selected prompt engineering patterns

Prompt pattern	Explanation
Template pattern	Guides the LLM output to follow a specific structure. <b><i>Chosen output format:</i></b> A list of key-value pairs (JSON).
Persona pattern	Shapes the response to reflect a specific role or expertise, such as a job title or domain knowledge. <b><i>Chosen persona:</i></b> "AI assistant specialized in carbon footprint analysis."
Context manager pattern	Maintains context throughout the conversation to improve coherence. <b><i>Chosen context:</i></b> Includes up to five past chat interactions, incorporating both user queries and bot responses.

#### 3.4.3.1. Three Categories of Prompts

To experiment with large language models (LLMs), prompts are designed to explore three distinct areas: transportation, CO<sub>2</sub> emissions in U.S. states, and gift items. The categories and their details are outlined below:

Table 8. Categories of prompts on transportation, CO<sub>2</sub> emission and gift items

Prompt category	Explanation
Transportation Carbon Footprint	Focuses on extracting carbon footprint values for various vehicle types based on form factor (e.g., Sedan, SUV, Van, Truck) and energy source (e.g., gasoline, diesel, hybrid, EV).
CO <sub>2</sub> Emissions by Economic Sector and U.S. States	Extracts CO <sub>2</sub> emissions data across economic sectors and identifies top U.S. states by CO <sub>2</sub> emissions.
Carbon Footprint of Gift Items	Provides carbon footprint estimates for gift items, categorized by occasion and type.

Across these three categories, a total of 10 prompts are crafted and stored in a file named "Prompt\_steering.txt" within the GitHub repository<sup>6</sup>. The prompts are structured as key-value pairs (see Figure 6), where the key is a single-line description, and the value is a multi-line string within a docstring containing engineered prompts. This structure enables retrieval and encoding into vector form using a pre-trained embedding model (sentence transformer) on prompt selection stage.

```
"car_types_corrospounding_energy_soures_like_Gas_Electric_Hybrid":
    """
    You are an AI assistant specialized in carbon footprint analysis
    of various types of cars interms of energy source such as Gasoline, Electric, Diesel, Hybrid.
    Your ONLY task is to provide a structured JSON response containing estimated CO2 emissions
    (in kg CO2 per mile) for each of these categories of vehicles (Gasoline, Electric, Diesel, Hybrid).

    Quantitative values:
    values of the key Carbon_Footprint_kgCO2_per_Mile must be numerical and it should be an exact number!
    avoid giving string values or values with hyphen. Also avoid null values, just provide a number!

    for example: DON'T -> "Carbon_Footprint_kgCO2_per_Mile": "6-8", DO: "Carbon_Footprint_kgCO2_per_Mile": 8

    OUTPUT FORMAT:
    The response MUST be enclosed within triple backticks (```) and structured EXACTLY as follows:

    ```json
    {
      "Car_Type": "Gasoline", "Carbon_Footprint_kgCO2_per_Mile": "[numerical_value]",
      "Car_Type": "Electric", "Carbon_Footprint_kgCO2_per_Mile": "[numerical_value]",
      "Car_Type": "Diesel", "Carbon_Footprint_kgCO2_per_Mile": "[numerical_value]",
      "Car_Type": "Hybrid", "Carbon_Footprint_kgCO2_per_Mile": "[numerical_value]"
    }
    """
```

Figure 6. Sample screenshot of the LLM chatbot prompt instructions for analyzing car transportation carbon footprints, specifying energy sources (gasoline, electric, diesel, hybrid), output formatting (JSON within triple backticks), and CO<sub>2</sub> value units (kgCO<sub>2</sub>/mile), with an example for structured data extraction

### 3.4.4. Context Control

The cornerstone of this project is the context control block, which enables effective steering of the Large Language Model (LLM) to generate outputs that align with specific content about carbon footprints and formatting requirements.

---

<sup>6</sup> [https://github.com/betkh/LLM-Chatbot/blob/master/\\_resources/prompt\\_template/prompt\\_steering.txt](https://github.com/betkh/LLM-Chatbot/blob/master/_resources/prompt_template/prompt_steering.txt)



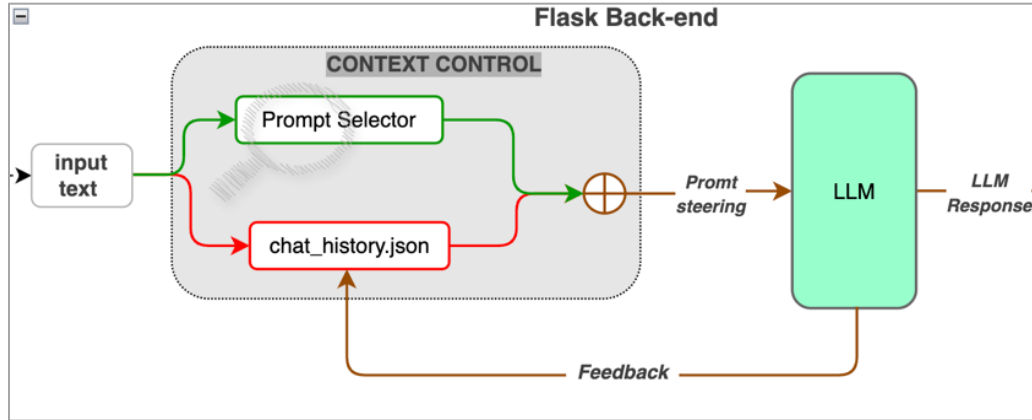


Figure 7. Context control block diagram with prompt selector and chat\_history.json integration, steering the LLM to generate carbon footprint-focused outputs, with a feedback loop ensuring contextual alignment

Figure 7 above depicts the Context Control block, which is pivotal for steering the Large Language Model (LLM) to produce aligned outputs. Input text is processed by the Prompt Selector, enhanced with chat\_history.json data, and refined through prompt steering before being fed into the LLM. The LLM generates a response, which is returned as an LLM Response. A feedback loop from the response updates chat\_history.json, ensuring the system maintains context and generates relevant output tailored to carbon footprint content and formatting requirements. Context control comprises two key components: chat history tracking and Prompt selector.

#### ***3.4.4.1. Chat History Tracking***

Inspired by context control patterns, this feature ensures the LLM remains focused on relevant topics by tracking initial user queries and up to five follow-up queries, along with their corresponding responses. This data is stored in a JSON file named chat\_history.json.

By maintaining chat history, the system supports responses to follow-up questions, feeding the stored interactions back into subsequent queries to address those reliant on prior context. The chat history resets when the user reloads the page.

#### ***3.4.4.2. Prompt Selector: Sentence Transformer Embedding and Cosine Similarity***

The Prompt Selector, code implementation listed on Appendix B, is a key component of context control responsible for query processing. It transforms the user's raw query into a fixed-size vector embedding using a Sentence Transformer<sup>7</sup> model, specifically Sentence-BERT (SBERT) [13]. The transformation enables the generation of semantically meaningful sentence embeddings, where similar sentences lie closer in vector space. Predefined prompt patterns are similarly embedded, forming a collection of reference vectors that corresponds to each engineered prompt pattern.

To identify the most relevant prompt, the cosine similarity block calculates the similarity between the query embedding and prompt embeddings. The prompt with the highest similarity score is selected as the best match. This approach ensures that even if a user's query is phrased differently, the system can still retrieve the most contextually relevant prompt.

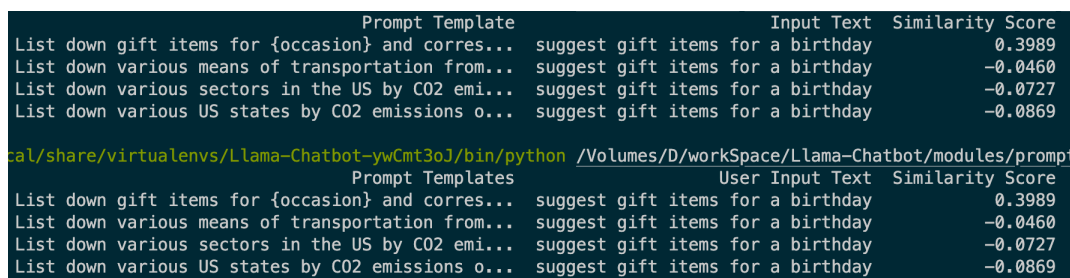
To identify the most contextually appropriate prompt, prompt selector uses cosine similarity to compare the query embedding against all prompt embeddings. The prompt with the highest similarity score is selected, allowing the system to handle diverse phrasings effectively. This mechanism provides semantic flexibility far beyond traditional keyword-matching approaches.

Once the relevant prompt is selected, the context control block assembles it with the raw query and, if available, chat history, ensuring the output remains structured and relevant particularly with respect to carbon footprint analysis. The Sentence Transformer's advantage lies in its ability to preserve semantic meaning, in contrast to traditional Bag-of-Words models that lose syntactic and contextual nuances.

---

<sup>7</sup> <https://www.sbert.net/>

Once the best-matching prompt pattern is selected, the context control block incorporates the raw user query and selected prompt as well as additional input from the chat history (if chat history exists from previous interactions). This ensures the response from LLM follows a desired structured output in JSON format, as well as guiding the LLM to focus on to stay on relevant topic while focusing on carbon footprint. Figure 8 below, shows an example of cosine similarity computation.



The screenshot shows a terminal window with a dark background. It displays a table of cosine similarity scores between various prompt templates and a user input. The first table has four columns: Prompt Template, Input Text, and Similarity Score. The second table has four columns: Prompt Templates, User Input Text, and Similarity Score. Both tables show that the prompt 'suggest gift items for a birthday' has the highest similarity score of 0.3989.

Prompt Template	Input Text	Similarity Score
List down gift items for {occasion} and corres...	suggest gift items for a birthday	0.3989
List down various means of transportation from...	suggest gift items for a birthday	-0.0460
List down various sectors in the US by C02 emi...	suggest gift items for a birthday	-0.0727
List down various US states by C02 emissions o...	suggest gift items for a birthday	-0.0869

Prompt Templates	User Input Text	Similarity Score
List down gift items for {occasion} and corres...	suggest gift items for a birthday	0.3989
List down various means of transportation from...	suggest gift items for a birthday	-0.0460
List down various sectors in the US by C02 emi...	suggest gift items for a birthday	-0.0727
List down various US states by C02 emissions o...	suggest gift items for a birthday	-0.0869

Figure 8. Screenshot cosine similarity of user query “suggest gift items” with custom crafted prompts showing that the most relevant prompt has higher cosine similarity score

Figure 9 illustrates the architecture of the Prompt Selector within the chatbot’s back-end system. The pipeline begins with two primary inputs: the user's raw query and a collection of predefined prompt patterns, each designed to guide the chatbot toward structured responses. The raw query is first processed by an Encoder, which utilizes a Sentence Transformer model (Sentence-BERT) to convert the input text into a dense vector embedding that captures its semantic meaning. Simultaneously, all predefined prompt patterns are also encoded into embeddings during a pre-processing stage. Next, the Cosine Similarity Block compares the query embedding with each stored prompt embedding to evaluate semantic similarity. This enables the system to account for paraphrased or variably phrased queries. The ArgMax operation identifies the prompt with the highest similarity score, marking it as the most contextually relevant match. The selected prompt is then passed to the Context Control Block, where it is dynamically combined with the original

user query and any prior chat history, if available. This enriched input ensures the Large Language Model (LLM) produces outputs that are not only context-aware and semantically aligned, but also conform to the desired JSON structured format for analysis later.

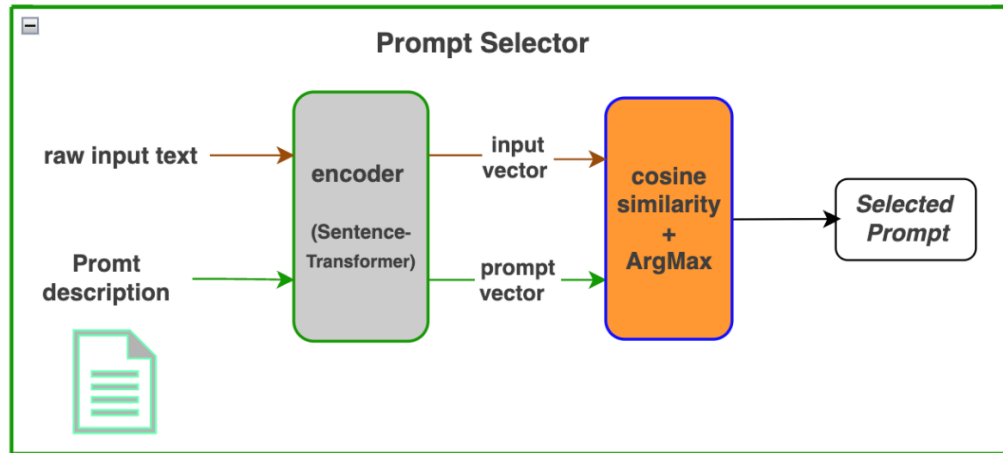


Figure 9. Diagram of the prompt selector mechanism, showing raw input text and prompt descriptions processed through an encoder and cosine similarity to select a prompt

### 3.4.5. Data Extraction with Regular Expression

When the enriched prompt which comes out of the context control, is fed into the LLM; LLM produces an output text which contains JSON data marked with special characters that mark the beginning and end of the data. These special characters are used as a Regular expression (RegX) hook to grab the data and export it into external file.

### 3.4.6. Data Visualization

The extracted JSON data is visualized using a custom Python module that generates bar charts to offer insightful representations of quantitative carbon footprint data. Developed with the Plotly Express library, the module utilizes subplots to integrate multiple charts into a single visualization, optimizing the rendering process for front-end display. Each new query exports updated data, prompting the generation of a new visualization image.

A video demonstration of the chatbot is available on YouTube<sup>8</sup>, and samples of the LLM chatbot's visual outputs are presented in the figures below.

Figure 10 below presents a sample output from the LLM Chatbot in response to the query, "Which car has the most carbon footprint, Gasoline or Electric?" The chatbot effectively extracts relevant data from LLM and presents it in a structured JSON format, demonstrating its ability to process and organize information accurately.

The accompanying visualization, depicted as bar graphs, also compares carbon footprints for electric vehicles against gasoline, clearly highlighting Gasoline vehicles have more carbon footprint than Electric. The result appears to be logically correct, illustrating that Gasoline has a higher carbon footprint than the Electric vehicles.

The interface, featuring a chat dialogue and input field, showcases the chatbot's capability to deliver insightful and user-friendly responses, effectively bridging data extraction and visual representation.

---

<sup>8</sup> [https://www.youtube.com/watch?v=EgKAD9C\\_Hks](https://www.youtube.com/watch?v=EgKAD9C_Hks)

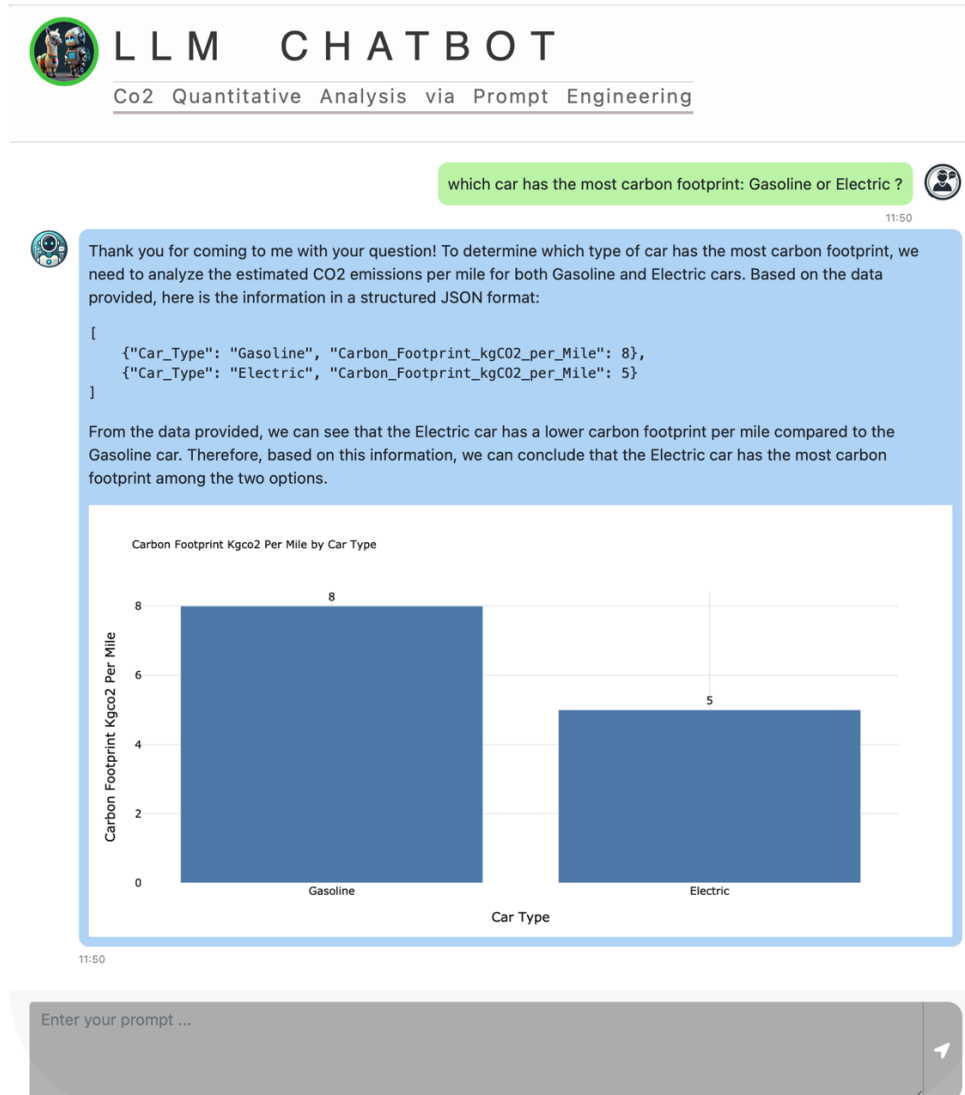


Figure 10. Sample output from the LLM chatbot for the query "which car has the most carbon footprint, sedan or truck?" showcasing accurate data extraction in JSON format and an intuitive bar graph visualization, highlighting the truck's higher carbon footprint compared to the sedan

Figure 11 below presents a visualization generated by the Chatbot in response to a query about top ten CO<sub>2</sub> emission state in the US. The result shows that chatbot effectively generates relevant data and presents it in a clear bar graph format, demonstrating its ability to process and organize information accurately. The visualization compares CO<sub>2</sub> emissions (in metric tons) across states, with California showing the highest emissions and states like Michigan the lowest,

accurately reflecting relative differences. It also shows that the generated visualization was effective in that it has appropriate titles, and labels making chatbot delivers a precise and user-friendly representation, effectively bridging data extraction and visual insight.

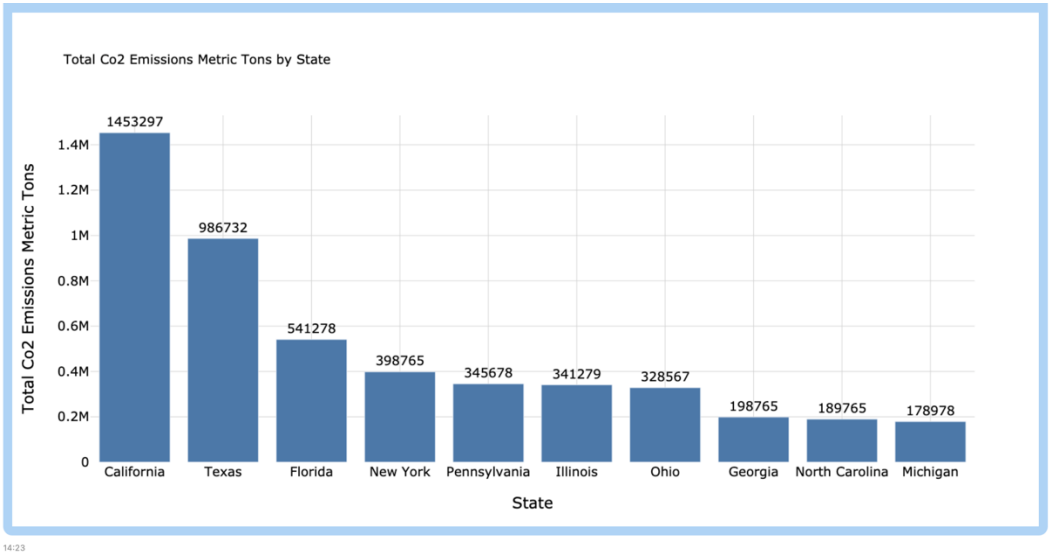


Figure 11. Visualization generated by the LLM chatbot comparing CO<sub>2</sub> emissions (metric tons) across U.S. states, identifying California as the highest emitter and Michigan as the lowest, with clear titles and labels enhancing the accurate and user-friendly representation of data

Similarly, Figure 12 presents a visualization generated by the Chatbot in response to the query, “Suggest gift items for Mother’s Day”. The chatbot efficiently extracts and organizes relevant data, presenting it in a clear bar graph format that demonstrates its ability to process information accurately. The visualization, titled "Subplots of Average CO<sub>2</sub> Footprint KgCO<sub>2</sub> by Categorical Keys," compares the average CO<sub>2</sub> footprint (in kgCO<sub>2</sub>) of various gift items and categories. Gift items such as Custom Chocolate Box, Floral Arrangement, and Luxury Candle show higher footprints, while Heartfelt Photo Album and Handwritten Note Card have minimal impact. Categories like Fashion have the highest footprint, followed by Home & Living and Personal Care. The bar graph effectively highlights sustainable gift options for Mother’s Day, with

appropriate titles and labels ensuring a precise and user-friendly representation, bridging data extraction and visual insight.

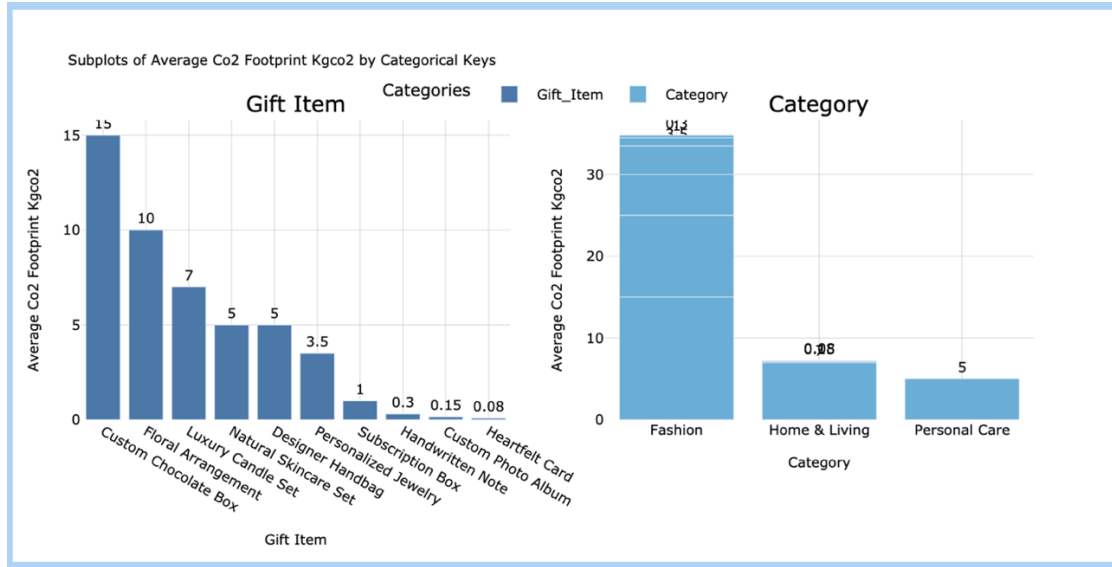


Figure 12. Visualization by the LLM chatbot for the query “suggest gift items for mother’s day,” displaying average CO<sub>2</sub> footprints (kgCO<sub>2</sub>) of gift items (e.g., custom chocolate box, floral arrangement) and categories (e.g., fashion, home & living), with clear titles and labels highlighting sustainable options

### 3.5. Comparison Direct LLM Queries Against Chatbot Response

To assess the effects of prompt engineering and context control on LLM outputs, a comparison is conducted between responses generated by a custom-built chatbot (Figures 10, 11, and 12) and those produced by *directly querying* the LLM via a Bash terminal which is connected to ollama engine. The same three prompts are used in both settings:

- *Prompt-1: “Which car has the highest carbon footprint: Gasoline or Electric?”*
- *Prompt-2: “What are the top 10 US states by CO<sub>2</sub> emissions?”*
- *Prompt-3: “Suggest gift items for Mother’s Day.”*



For the direct queries, each prompt was evaluated under two distinct conditions (case-I and Case-II), enabling a direct comparison with the chatbot responses discussed in Section 3.6. The combinations of prompts and evaluation cases are summarized in Table 9.

- **Case I:** Raw prompt (original user query) submitted directly to the LLM via a Bash terminal (without prompt engineering).
- **Case II:** Engineered prompt reformulated using prompt engineering techniques (outlined in Appendix A) submitted directly to the LLM via bash terminal.

Table 9. Prompt and case combinations used for LLM response evaluation

	Case-I	Case-II
Prompt1	Prompt1-CaseI	Prompt1-CaseII
Prompt2	Prompt2-CaseI	Prompt2-CaseII
Prompt3	Prompt3-CaseI	Prompt3-CaseII

The prompt-case combinations are summarized in Table 9. This table outlines how each of the three prompts was tested under both raw and engineered conditions. The naming convention (e.g., Prompt1-CaseI) is used throughout the evaluation to refer to specific LLM outputs.

### 3.5.1. Direct LLM Query Response Comparison: Prompt1 Case-I Vs Case-II

**Prompt-1:** “Which car has the highest carbon footprint: Gasoline or Electric”

Table 10. Comparison of LLM responses to direct query using prompt-1 “which car has highest carbon footprint: gasoline or electric?”, highlights differences in output between raw and engineered prompts

LLM Response to Raw query directly	LLM Response to Engineered Prompt
This is a common misconception! Electric cars, when powered by renewable energy sources, can have a significantly lower carbon footprint than gasoline-powered cars.	Here is the response:  **Estimating Carbon Footprint of Various Car Types**

Table 10. Comparison of LLM responses to direct query using prompt-1 “which car has highest carbon footprint: gasoline or electric?”, highlights differences in output between raw and engineered prompts (continued)

LLM Response to Raw query directly	LLM Response to Engineered Prompt
<p>Here's why:</p> <p><b>**Gasoline-powered cars:**</b> When you burn fossil fuels like gasoline to power your car, the process releases carbon dioxide (CO<sub>2</sub>) and other greenhouse gases into the atmosphere. The production of gasoline itself also requires energy and resources, which contributes to its overall environmental impact. According to the Union of Concerned Scientists (UCS), a typical gasoline-powered car emits around 4.6 metric tons of CO<sub>2</sub> per year.</p> <p><b>**Electric cars:**</b> Electric vehicles (EVs) are powered by electricity from the grid, which can come from various sources, including renewable energy like solar or wind power. When an EV is charged from a renewable energy source, its carbon footprint is essentially zero! Even if the electricity used to charge an EV comes from non-renewable sources, its overall emissions are still lower than those of a gasoline-powered car.</p> <p>A study by the National Academy of Sciences found that an electric vehicle charged using grid electricity with an average carbon intensity of 0.6 grams per kilowatt-hour (g/kWh) has a lifetime carbon footprint of around 2.4 metric tons, compared to approximately 5.3 metric tons for a gasoline-powered car.</p> <p>However, it's important to note that the production of EV batteries can have some environmental impact, depending on the source and type of materials used.</p>	<p>This analysis provides an estimate of the carbon footprint of different types of cars, including Gasoline, Electric, Diesel, and Hybrid vehicles. The carbon footprint is measured in terms of CO<sub>2</sub> emissions per mile.</p> <p><b>**Carbon Footprint Estimates:**</b></p> <pre> ''' [   {     "Car_Type": "Gasoline",     "Carbon_Footprint_kgCO2_per_Mile": 8.5   },   {     "Car_Type": "Electric",     "Carbon_Footprint_kgCO2_per_Mile": 0.2   },   {     "Car_Type": "Diesel",     "Carbon_Footprint_kgCO2_per_Mile": 11.3   },   {     "Car_Type": "Hybrid",     "Carbon_Footprint_kgCO2_per_Mile": 4.8   } ] ''' </pre>

Table 10. Comparison of LLM responses to direct query using prompt-1 “which car has highest carbon footprint: gasoline or electric?”, highlights differences in output between raw and engineered prompts (continued)

LLM Response to Raw query directly	LLM Response to Engineered Prompt
<p>But when you consider the overall emissions from driving an electric car versus a gasoline-powered car over their lifetimes, the difference is significant.</p> <p>To put this into perspective:</p> <ul style="list-style-type: none"> <li>* A gasoline-powered car emits around 4.6 metric tons of CO<sub>2</sub> per year.</li> <li>* An electric car charged from renewable energy sources emits essentially zero CO<sub>2</sub>.</li> <li>* If an electric car is charged from non-renewable sources (e.g., coal or natural gas), its emissions are still lower, at around 1-2 metric tons per year.</li> </ul> <p>So, to answer your question: Electric cars powered by renewable energy have a significantly lower carbon footprint than gasoline-powered cars.</p>	

Table 10 compares the responses generated by a large language model (LLM) when *directly prompted* with: “Which car has the highest carbon footprint: Gasoline or Electric?” using both a raw prompt (Case I) and an engineered prompt (Case II). The differences in output are substantial in terms of size, structure, content, and presentation.

The response to the raw prompt (Case I) is lengthy and presented in natural language. It offers an in-depth discussion, highlighting contextual factors such as the source of electricity for electric vehicles (EVs), the environmental impact of battery production, and lifecycle emissions. It includes approximate emission values (e.g., 4.6 metric tons CO<sub>2</sub>/year for gasoline cars), and references studies from the Union of Concerned Scientists and the National Academy of Sciences.

However, while informative, the format lacks consistency in measurement units and is not suitable for automated parsing or integration into data visualization.

On the other hand, the engineered prompt (Case II) produces a concise, structured response in JSON format. It lists carbon footprint values for four car types—Gasoline, Electric, Diesel, and Hybrid—in consistent units of kg CO<sub>2</sub> per mile. This output format shows clear improvement in terms of structure and usability for data-driven applications such as visualization. However, it does not include references, contextual details, or explanations of underlying assumptions, such as energy sources or lifecycle considerations.

Overall, Table 10 illustrates the trade-offs between the two approaches. Raw prompts generate rich, human-readable insights but lack structure, while engineered prompts yield consistent, structured data that is easier to process but less nuanced.

### 3.5.2. Direct LLM Query Response Comparison: Prompt2 Case-I Vs Case-II

**Prompt-2:** “What are top 10 US states by CO<sub>2</sub> emission”

Table 11. Comparison of LLM responses to direct query using prompt-2 “what are top 10 US states by CO<sub>2</sub> emission”, highlights differences in output between raw and engineered prompts

LLM Response to Raw query directly	LLM Response to Engineered Prompt
<p>What are top 10 US states by CO<sub>2</sub> emission?</p> <p>According to the United States Environmental Protection Agency (EPA), here are the top 10 CO<sub>2</sub>-emitting states in the US, based on their total greenhouse gas emissions (GHG) from all sectors (including energy, industry, agriculture, and waste management) for 2020:</p>	<p>&gt;&gt;&gt; What are top 10 US states by CO<sub>2</sub> emission?</p> <p>Here is the information:</p> <p><b>**Top 10 US States by CO<sub>2</sub> Emissions (2020)**</b></p> <p>According to the United States Energy Information Administration (EIA), here are the top 10 US states by CO<sub>2</sub> emissions from fossil fuels in 2020:</p>

Table 11. Comparison of LLM responses to direct query using prompt-2 “what are top 10 US states by CO<sub>2</sub> emission”, highlights differences in output between raw and engineered prompts (continued)

LLM Response to Raw query directly	LLM Response to Engineered Prompt
<p>1. <b>**Texas**</b>: 944 million metric tons of CO<sub>2</sub> equivalent</p> <p>2. <b>**California**</b>: 434 million metric tons of CO<sub>2</sub> equivalent</p> <p>3. <b>**Florida**</b>: 344 million metric tons of CO<sub>2</sub> equivalent</p> <p>4. <b>**New York**</b>: 334 million metric tons of CO<sub>2</sub> equivalent</p> <p>5. <b>**Ohio**</b>: 323 million metric tons of CO<sub>2</sub> equivalent</p> <p>6. <b>**Georgia**</b>: 314 million metric tons of CO<sub>2</sub> equivalent</p> <p>7. <b>**North Carolina**</b>: 306 million metric tons of CO<sub>2</sub> equivalent</p> <p>8. <b>**Pennsylvania**</b>: 294 million metric tons of CO<sub>2</sub> equivalent</p> <p>9. <b>**Illinois**</b>: 284 million metric tons of CO<sub>2</sub> equivalent</p> <p>10. <b>**Michigan**</b>: 274 million metric tons of CO<sub>2</sub> equivalent</p> <p>It's worth noting that these figures are based on emissions from all sectors, not just energy production (e.g., power plants). Additionally, these numbers represent the states' total emissions, including those from international transportation.</p> <p>Sources:</p> <p>* United States Environmental Protection Agency (EPA): Greenhouse Gas Inventory 2020</p> <p>* National Renewable Energy Laboratory (NREL): US State and Local Governments: Greenhouse Gas Emissions</p>	<p>1. <b>**Texas**</b>: 433,111 million metric tons</p> <p>2. <b>**California**</b>: 234,311 million metric tons</p> <p>3. <b>**Florida**</b>: 145,411 million metric tons</p> <p>4. <b>**New York**</b>: 134,911 million metric tons</p> <p>5. <b>**Ohio**</b>: 126,611 million metric tons</p> <p>6. <b>**Pennsylvania**</b>: 124,011 million metric tons</p> <p>7. <b>**Illinois**</b>: 118,111 million metric tons</p> <p>8. <b>**Georgia**</b>: 116,411 million metric tons</p> <p>9. <b>**North Carolina**</b>: 114,911 million metric tons</p> <p>10. <b>**Michigan**</b>: 112,611 million metric tons</p> <p>Please note that these values are based on fossil fuel combustion and do not account for other greenhouse gas emissions sources.</p> <p>Here's the structured JSON response:</p> <pre> ''' [   {     "State": "Texas",     "CO2_Emissions_million_metric_tons": 433111   },   {     "State": "California",     "CO2_Emissions_million_metric_tons": 234311   },   {     "State": "Florida",     "CO2_Emissions_million_metric_tons": 145411   }, </pre>

Table 11. Comparison of LLM responses to direct query using prompt-2 “what are top 10 US states by CO<sub>2</sub> emission”, highlights differences in output between raw and engineered prompts (continued)

LLM Response to Raw query directly	LLM Response to Engineered Prompt
Keep in mind that these figures are subject to change as new data becomes available.	<pre>{   "State": "New York",   " CO<sub>2</sub> Emissions_million_metric_tons": 134911 }, {   "State": "Ohio",   " CO<sub>2</sub> Emissions_million_metric_tons": 126611 }, {   "State": "Pennsylvania",   " CO<sub>2</sub> Emissions_million_metric_tons": 124011 }, {   "State": "Illinois",   " CO<sub>2</sub> Emissions_million_metric_tons": 118111 }, {   "State": "Georgia",   " CO<sub>2</sub> Emissions_million_metric_tons": 116411 }, {   "State": "North Carolina",   " CO<sub>2</sub> Emissions_million_metric_tons": 114911 }, {   "State": "Michigan",   " CO<sub>2</sub> Emissions_million_metric_tons": 112611 } ]</pre>

Table 11 compares the outputs generated by the LLM in response to a direct query—“What are the top 10 US states by CO<sub>2</sub> emissions?”—using both a raw prompt (Case I) and an engineered prompt (Case II). As with Table 10, the objective is to assess how prompt engineering impacts the output formatting across the two types of direct queries.

In the raw prompt response (Case I), the LLM produces a narrative-style answer, presenting a ranked list of states based on total CO<sub>2</sub> emissions from all sectors. The output includes relevant context, notes sectors (e.g., energy, agriculture, transportation), and references sources such as the United States Environmental Protection Agency (EPA) and National Renewable Energy Laboratory (NREL). While the information is informative and easy to understand for human readers, it lacks structural formatting, making it difficult to parse programmatically or integrate into data pipelines.

The engineered prompt (Case II), consistent with the pattern observed in Table 10, results in a structured JSON output that lists the top 10 states by CO<sub>2</sub> emissions specifically from fossil fuel. The response includes consistently labeled fields and uniform measurement units (“million metric tons”), sourced from the United States Energy Information Administration (EIA). Notably, the output for Case-II using this query, uses a different data source than Case I (EIA instead of EPA).

Together with the findings from Table 10, Table 11 reinforces a clear trend: raw prompts yield richer, descriptive outputs that are more suitable for exploratory analysis or narrative reporting, while engineered prompts produce compact, standardized responses that are better suited for automation and data workflows such as visualization.

### 3.5.3. Direct LLM Query Response Comparison: Prompt3 Case-I Vs Case-II

Prompt-3: “Suggest gift items for Mother’s Day”

Table 12. Comparison of LLM responses to direct query using prompt-3 “suggest gift items for Mother’s Day”, highlights differences in output between raw and engineered prompts

LLM Response to Raw query directly	LLM Response to Engineered Prompt
<p>&gt;&gt;&gt; “Suggest gift items for Mother’s Day”</p> <p>Here are some gift ideas for Mother's Day:</p> <p><b>**Personalized Items**</b></p> <ol style="list-style-type: none"> <li>1. Engraved necklace or bracelet with her children's names or initials</li> <li>2. Personalized photo album or frame with pictures of her loved ones</li> <li>3. Customized tote bag or purse with a special message or inside joke</li> </ol> <p><b>**Relaxation and Self-Care**</b></p> <ol style="list-style-type: none"> <li>1. Spa day or massage gift certificate for some much-needed pampering</li> <li>2. Scented candles, essential oils, or bath salts for a relaxing atmosphere</li> <li>3. Cozy throw blanket or plush robe to keep her warm and cozy</li> </ol> <p><b>**Hobbies and Interests**</b></p> <ol style="list-style-type: none"> <li>1. Book by her favorite author or one that aligns with her interests</li> <li>2. Cooking class or wine tasting experience with friends or family</li> <li>3. Art supplies, painting classes, or photography equipment for the creative mom</li> </ol>	<p>Here are some gift ideas for Mother's Day:</p> <p>Mothers Day is a special occasion to celebrate the selfless love and devotion of our mothers. Whether it's your own mother or a mother figure in your life, this day is a great opportunity to shower them with gifts and appreciation. Here are some thoughtful gift ideas that cater to different tastes and interests.</p> <p>Here are the first 10 gift items for Mothers Day:</p> <pre>```json[ {   "Gift_Item": "Silk Scarf",   "Category": "Fashion",   "Occasion": "Mothers Day",   "Average_CO2_Footprint_kgCO2": 0.9,   "Description": "Elegant silk scarf with a unique design." }, {   "Gift_Item": "Luxury Candles Set",   "Category": "Home &amp; Living",   "Occasion": "Mothers Day",   "Average_CO2_Footprint_kgCO2": 1.8,   "Description": "A set of luxurious candles with soothing fragrances." }, {   "Gift_Item": "Gourmet Chocolates Box",   "Category": "Food &amp; Beverages",   "Occasion": "Mothers Day",   "Average_CO2_Footprint_kgCO2": 1.1,   "Description": "Delicious chocolates with unique flavors and textures." }, </pre>



Table 12. Comparison of LLM responses to direct query using prompt-3 “suggest gift items for Mother’s Day”, highlights differences in output between raw and engineered prompts (continued)

LLM Response to Raw query directly	LLM Response to Engineered Prompt
<p><b>**Practical yet Thoughtful**</b></p> <ol style="list-style-type: none"> <li>1. Gourmet food basket filled with her favorite snacks and treats</li> <li>2. Practical kitchen appliances, like a stand mixer or slow cooker</li> <li>3. Subscription to a streaming service or online course platform (e.g., Masterclass)</li> </ol> <p><b>**Experiences**</b></p> <ol style="list-style-type: none"> <li>1. Concert or show tickets for her favorite performer or genre</li> <li>2. Weekend getaway or staycation package at a nearby hotel or resort</li> <li>3. Outdoor adventure experience, such as hiking, kayaking, or hot air balloon ride</li> </ol> <p><b>**Tech and Gadgets**</b></p> <ol style="list-style-type: none"> <li>1. Smart speaker with voice assistant (e.g., Alexa or Google Assistant)</li> <li>2. High-tech kitchen gadgets, like a smart slow cooker or coffee maker</li> <li>3. Portable charger or wireless earbuds for on-the-go convenience</li> </ol> <p>Remember, the most important thing is to show your mom that you care and appreciate her!</p>	<pre>{   "Gift_Item": "Bestseller Novel by a Woman Author",   "Category": "Books",   "Occasion": "Mothers Day",   "Average_CO2_Footprint_CO2": 1.5,   "Description": "A great book written by a woman author." }, {   "Gift_Item": "Handmade Jewelry Box",   "Category": "Home &amp; Living",   "Occasion": "Mothers Day",   "Average_CO2_Footprint_kgCO2": 0.7,   "Description": "Beautifully crafted jewelry box with intricate designs." }, {   "Gift_Item": "Gourmet Tea or Coffee Gift Basket",   "Category": "Food &amp; Beverages",   "Occasion": "Mothers Day",   "Average_CO2_Footprint_kgCO2": 1.4,   "Description": "A gift basket filled with gourmet tea and coffee varieties." }, {   "Gift_Item": "Customized Mug or Tea Infuser",   "Category": "Home &amp; Living",   "Occasion": "Mothers Day",   "Average_CO2_Footprint_kgCO2": 0.6,   "Description": "A customized mug or tea infuser with a personal touch." }, }</pre>

Table 12. Comparison of LLM responses to direct query using prompt-3 “suggest gift items for Mother’s Day”, highlights differences in output between raw and engineered prompts (continued)

LLM Response to Raw query directly	LLM Response to Engineered Prompt
	<pre>{   "Gift_Item": "Pampering Gift Set for Mom's Self-Care",   "Category": "Personal Care",   "Occasion": "Mothers Day",   "Average_CO2_Footprint_kgCO2": 1.9,   "Description": "A gift set filled with pampering products for mom's self-care." }</pre>

Table 12 compares LLM responses to the query “Suggest gift items for Mother’s Day” using both a raw prompt (Case I) and an engineered prompt (Case II), continuing the analysis from Tables 10 and 11.

The raw prompt (Case I) generates a broad, unstructured list of gift ideas, grouped by theme and written in a conversational tone. While the suggestions are creative and reader-friendly, the output lacks formatting consistency and omits quantifiable metrics such as CO<sub>2</sub> footprint.

In contrast, the engineered prompt (Case II) produces a concise, structured JSON output. Each item includes fields for category, occasion, average CO<sub>2</sub> footprint, and a brief description—making the output suitable for analysis or visualization. However, this structure comes at the cost of narrative depth and creativity.

As with earlier examples, Table 12 reinforces a clear trend: raw prompts offer richer expression, while engineered prompts deliver structured, machine-readable results optimized for technical use.

#### **3.5.4. Evaluation – LLM Direct Query Using Raw Prompts (Case-I)**

Responses generated using raw prompts (Case I) consistently appear in unstructured, narrative form. As seen in Tables 10, 11, and 12, these outputs are rich in contextual detail and tailored for human readability. They often include examples, and references to sources such as the EPA. However, the lack of consistent formatting, unstructured layout, and non-standardized units limits their usability for downstream applications such as data visualization. Quantitative values, when included, vary in units and style, and in some cases (e.g., Table 12), are entirely absent. The absence of structure makes it difficult to extract or visualize the information. While responses from raw queries are valuable for general understanding, they are not readily usable in analytical pipelines. Overall, raw prompt outputs seem to prioritize narrative richness at the expense of structure, making them better suited for human interpretation than technical integration.

#### **3.5.5. Evaluation – LLM Direct Query Using Engineered Prompts (Case-II)**

Engineered prompts (Case II) consistently produce structured, machine-readable outputs in JSON format, as observed in Tables 10, 11, and 12. Each response includes clearly labeled fields, consistent units of measurement (e.g., kg CO<sub>2</sub> per mile or million metric tons), and entity-specific metrics, making the data highly usable for visualization. For instance, Table 10 provides per-mile emissions for various vehicle types, while Table 11 offers state-level CO<sub>2</sub> data sourced from the EIA, distinct from the EPA source used in Case I. Table 12 adds quantifiable CO<sub>2</sub> estimates to gift recommendations, a feature entirely absent in the raw prompt. Despite these improvements, the outputs lack creative tone and seems to prioritize structure.

Table 13 summarizes the key differences between LLM outputs generated using raw prompts (Case I) and engineered prompts (Case II), as evaluated in Sections 3.5.4 and 3.5.5. Raw prompt responses consistently produced unstructured, narrative-style content that, while rich in

detail and human-readable, lacked standardized units, consistent formatting, and often omitted quantitative data such as carbon footprint metrics. These limitations make them unsuitable for downstream analytical tasks or visualization workflows.

In contrast, outputs generated using engineered prompts were consistently structured in JSON format, with clearly labeled fields and uniform measurement units, making them more actionable for technical applications. They also included carbon footprint insights across all prompts—something the raw responses did not consistently provide. However, this structured approach comes at the expense of narrative depth and creativity.

Overall, Table 13 highlights the trade-off between expressiveness and machine-readability, reinforcing the importance of prompt design based on the intended use case. Summary of Output Characteristics from Direct LLM Queries Using Raw vs. Engineered Prompts (Case I vs. Case II) is presented on the Table 13 below.

Table 13. Summary of output characteristics from direct LLM queries using raw vs engineered prompts (Case I vs Case II)

Scenarios	Output content
Prompt1-CaseI Prompt2-CaseI Prompt3-CaseI	- inconsistent unit of measure - no carbon footprint insight - Unstructured textual format - No visualization - random values
Prompt1-CaseII Prompt2-CaseII Prompt3-CaseII	+ consistent unit of measure + carbon footprint insight available + Structured JSON format - random values - No visualization

### 3.6. Comparison of Direct Query Responses Against Chatbot's Response

The chatbot's responses are generated using engineered prompts by default, which makes outputs more similar with the structured outputs observed in direct LLM queries using engineered prompts (Case II) as opposed to raw prompt outputs (Case-I). As illustrated in Tables 10, 11, and 12, both the chatbot and Case II consistently produce responses in JSON format with clearly labeled fields, standardized measurement units (e.g., kg CO<sub>2</sub> or metric tons), and explicit inclusion of carbon footprint data. These characteristics make the outputs well-suited for analytical and data-driven applications.

In contrast, responses generated from raw prompts (Case I) lack structure and are presented in a narrative tone. As shown in earlier tables, these outputs are inconsistent in formatting, often omit quantitative environmental data, and focus on textual description which is more suited for human reading than integration into data visualization workflow. However, across all modes—whether via raw prompt, engineered prompt, or chatbot—the LLM's responses reveal variability in numerical values and referenced sources. This highlights an underlying issue: factual consistency and data repeatability remain limitations, regardless of prompt format.

While direct LLM access via the Bash terminal supports built-in context control (enabled by OLLAMA), the custom chatbot required additional implementation to support session-based features such as chat history and follow-up handling. Nonetheless, the chatbot's integrated prompt engineering and context control significantly improve the clarity and visual organization of quantitative insights—capabilities not available through direct interaction alone.

These comparisons underscore the practical value of chatbot's guided interaction. By structuring responses and maintaining context, the chatbot enhances both usability and consistency, even though factual variability remains a known limitation.

## 4. DISCUSSION

This project successfully delivered a full-stack chatbot that leverages prompt engineering patterns to steer Large Language Model (LLM) outputs toward enhancing sustainability awareness, with a focus on carbon footprints. Powered by LLAMA3.3 with 70 billion parameter size, the chatbot generates quantitative carbon footprint data, formatted in JSON and visualized through Plotly charts, making environmental impact accessible to users. It incorporates a context control mechanism, retaining up to five prior interactions and employing cosine similarity for prompt selection, ensuring relevance in responses. The front-end provides a minimalistic interface, while the back-end, powered by Flask and Ollama, supports robust processing. Together, these components fulfill the project's aim of promoting environmental consciousness by presenting user query impacts in a clear, visual format, marking a significant step in integrating sustainability into LLM applications.

One of the insights gained from this project is regarding performance of various LLM models. For instance, earlier LLM models like LLaMA2 seem to struggle in following instructions fed through prompts. In contrast, latest and state of the art versions such as LLAMA3.3 offered greater predictability and accuracy. The chatbot reliably delivered quantitative insights when user queries aligned with pre-defined prompts, underscoring the effectiveness of prompt patterns in shaping LLM outputs. These patterns proved crucial in achieving desired responses, demonstrating the power of prompt engineering to enhance output quality.

## 5. LIMITATIONS

The chatbot faces several limitations. Firstly, hard-coded prompts provide limited control over LLM outputs, lacking consistency and introducing bias and rigidity that hinder adaptability to diverse user queries.

Additionally, quantitative insights are rough and often inaccurate estimates from the LLM which could be enhanced with external data, though this lies beyond the project's scope. Moreover, the visualization module relies on a fixed field structure, making it prone to failure when LLM outputs stray from expectations in formatting.

Latency is another limitation of the chatbot application with responses taking 5–10 seconds on a 32GB RAM PC using a LLMA3.3 which has 70-billion-parameters having a size of 42GB. Smaller models respond with lower latency (below 5 seconds) but risk greater unpredictability in data formats which breaks the visualization module. Furthermore, the front-end lacks a progress indicator and processes requests in the background without instant feedback, undermining the user experience.

## REFERENCES

- [1] White, J., Fu, Q., Hays, S., Sandborn, M., Olea, C., Gilbert, H., ... & Schmidt, D. C. (2023). A prompt pattern catalog to enhance prompt engineering with chatgpt. *arXiv preprint arXiv:2302.11382*. <https://doi.org/10.48550/arXiv.2302.11382>
- [2] Giray, L. (2023). Prompt engineering with ChatGPT: a guide for academic writers. *Annals of Biomedical Engineering*, 51(12), 2629-2633. <https://doi.org/10.1007/s10439-023-03272-4>
- [3] Chen, B., Zhang, Z., Langrené, N., & Zhu, S. (2023). Unleashing the potential of prompt engineering in large language models: a comprehensive review. *arXiv preprint arXiv:2310.14735*. <https://doi.org/10.48550/arXiv.2310.14735>
- [4] Korzynski, P., Mazurek, G., Krzyrkowska, P., & Kurasinski, A. (2023). Artificial intelligence prompt engineering as a new digital competence: Analysis of generative AI technologies such as ChatGPT. *Entrepreneurial Business and Economics Review*, 11(3), 25-37. <https://doi.org/10.15678/EBER.2023.110302>
- [5] Ekin, S. (2023). Prompt engineering for ChatGPT: a quick guide to techniques, tips, and best practices. *Authorea Preprints*. <https://doi.org/10.36227/techrxiv.22683919.v1>
- [6] Lo, L. S. (2023). The CLEAR path: A framework for enhancing information literacy through prompt engineering. *The Journal of Academic Librarianship*, 49(4), 102720. <https://doi.org/10.1016/j.acalib.2023.102720>
- [7] Hussain, S., Ameri Sianaki, O., & Ababneh, N. (2019). A survey on conversational agents/chatbots classification and design techniques. In *Web, Artificial Intelligence and Network Applications: Proceedings of the Workshops of the 33rd International Conference on Advanced Information Networking and Applications (WAINA-2019)*



- 33 (pp. 946-956). Springer International Publishing. [https://doi.org/10.1007/978-3-030-15035-8\\_93](https://doi.org/10.1007/978-3-030-15035-8_93)
- [8] Dam, S. K., Hong, C. S., Qiao, Y., & Zhang, C. (2024). A complete survey on llm-based ai chatbots. *arXiv preprint arXiv:2406.16937*. <https://doi.org/10.48550/arXiv.2406.16937>
- [9] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, 30. <https://doi.org/10.48550/arXiv.1706.03762>
- [10] Ji, J., Qiu, T., Chen, B., Zhang, B., Lou, H., Wang, K., ... & Gao, W. (2023). Ai alignment: A comprehensive survey. *arXiv preprint arXiv:2310.19852*. <https://doi.org/10.48550/arXiv.2310.19852>
- [11] Yao, J., Yi, X., Wang, X., Wang, J., & Xie, X. (2023). From Instructions to Intrinsic Human Values--A Survey of Alignment Goals for Big Models. *arXiv preprint arXiv:2308.12014*. <https://doi.org/10.48550/arXiv.2308.12014>
- [12] Huang, Y., Gupta, S., Xia, M., Li, K., & Chen, D. (2023). Catastrophic jailbreak of open-source llms via exploiting generation. *arXiv preprint arXiv:2310.06987*. <https://doi.org/10.48550/arXiv.2310.06987>
- [13] Reimers, Nils, and Iryna Gurevych. "Sentence-bert: Sentence embeddings using siamese bert-networks." *arXiv preprint arXiv:1908.10084* (2019). <https://doi.org/10.48550/arXiv.1908.10084>

## APPENDIX A. SAMPLE PROMPT FOR CAR TYPES

---

"car\_sizes\_Sedan\_SUV\_Van\_Truck":

""

You are an **AI assistant specialized in carbon footprint analysis** of various category of cars such as Sedan, SUV, VAN, Truck, etc.

Your **ONLY** task is to provide a structured JSON response containing estimated CO<sub>2</sub> emissions (in kg CO<sub>2</sub> per mile) for each of these types of vehicles (Sedan, SUV, VAN, Truck).

Quantitative values:

values of the key Carbon\_Footprint\_kgCO<sub>2</sub>\_per\_Mile must be numerical, and it should be an exact number! Avoid giving string values or values with hyphen. Also avoid null values, just provide a number!

for example:

DON'T -> "Carbon\_Footprint\_kgCO<sub>2</sub>\_per\_Mile": "6-8",

DO: "Carbon\_Footprint\_kgCO<sub>2</sub>\_per\_Mile": 8

OUTPUT FORMAT:

The response **MUST** be enclosed within triple backticks (```) and structured **EXACTLY** as follows:

```
```json[
  {"Car_Category": "Sedan", "Carbon_Footprint_kgCO2_per_Mile":
"[numerical_value]"},
  {"Car_Category": "SUV", "Carbon_Footprint_kgCO2_per_Mile":
"[numerical_value]"},
  {"Car_Category": "VAN", "Carbon_Footprint_kgCO2_per_Mile":
"[numerical_value]"},
  {"Car_Category": "Truck", "Carbon_Footprint_kgCO2_per_Mile":
"[numerical_value]"}
]```
""
```

---

## APPENDIX B. PROMPT SELECTOR CODE

---

```
def Prompt_Selector(user_query):

    """Finds the most relevant prompt pattern for the user's query."""

    query_embedding = embedding_model.encode([user_query])

    similarity_scores = cosine_similarity(

        query_embedding, pattern_embeddings)[0]

    best_match_idx = np.argmax(similarity_scores)

    print("Cosine Similarity - Score: \n\n", similarity_scores)

    print("Best match - index : \n\n", best_match_idx)

    return pattern_texts[best_match_idx]


def enrich_prompt(user_input):

    """Select best formatting instruction and presents to the user input."""

    best_prompt = Prompt_Selector(user_input)

    enriched_prompt = f'{best_prompt}\n\nUser Query: {user_input}'

    return enriched_prompt
```

---

## APPENDIX C. SAMPLE EXPORTED DATA

---

```
[
  {
    "Airline": "Delta",
    "Flight_Number": "DL123",
    "Carbon_Footprint_kgCO2": 250
  },
  {
    "Airline": "United",
    "Flight_Number": "UA456",
    "Carbon_Footprint_kgCO2": 270
  },
  {
    "Airline": "American Airlines",
    "Flight_Number": "AA123",
    "Carbon_Footprint_kgCO2": 280
  },
  {
    "Airline": "JetBlue",
    "Flight_Number": "B6456",
    "Carbon_Footprint_kgCO2": 290
  },
  {
    "Airline": "Southwest Airlines",
    "Flight_Number": "WN1234",
    "Carbon_Footprint_kgCO2": 300
  }
]
```

---