

**A  
Project Report  
on**

**CLI Command Generation Using Generative AI**

**Submitted to**

**Sant Gadge Baba Amravati University, Amravati**

**Submitted in partial fulfilment of  
the requirements for the Degree of  
Bachelor of Engineering in  
Computer Science and Engineering**

**Submitted by**

**Atharva Tattu**

(PRN:203120386)

**Rushikesh Dhawne**

(PRN:203120290)

**Prajwal Chitode**

(PRN: 213120427)

**Vedant Chaudhari**

(PRN:203120299)

**Under the Guidance of  
Name of guide  
Dr. P. V. Deshmukh  
Asst. Prof, CSE Dept.**



**Department of Computer Science and Engineering  
Shri Sant Gajanan Maharaj College of Engineering,  
Shegaon – 444203 (M.S.)  
Session 2023-2024**

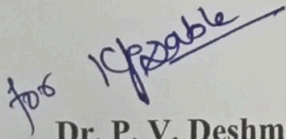
SHRI SANT GAJANAN MAHARAJ COLLEGE OF ENGINEERING,  
SHEGAON – 444203 (M.S.)

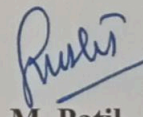
DEPARTMENT OF COMPUTER SCIENCE AND  
ENGINEERING

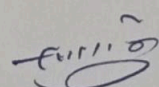


## CERTIFICATE

This is to certify that **Mr. Atharva Tattu, Mr. Prajwal Chitode, Mr. Rushikesh Dhawne, and Mr. Vedant Chaudhari** students of final year Bachelor of Engineering in the academic year 2023-24 of Computer Science and Engineering Department of this institute have completed the project work entitled “**CLI Command Generation using Generative AI**” and submitted a satisfactory work in this report. Hence recommended for the partial fulfillment of degree of Bachelor of Engineering in Computer Science and Engineering.

  
**Dr. P. V. Deshmukh**  
Project Guide

  
**Dr. J. M. Patil**  
Head of Department

  
**Dr. S. B. Somani**  
Principal  
SSGMCE, Shegaon



SHRI SANT GAJANAN MAHARAJ COLLEGE OF ENGINEERING,  
SHEGAON – 444 203 (M.S.)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING



**CERTIFICATE**

This is to certify that **Mr. Atharva Tattu, Mr. Prajwal Chitode, Mr. Rushikesh Dhawne and Mr. Vedant Chaudhari** students of final year Bachelor of Engineering in the academic year 2023-24 of Computer Science and Engineering Department of this institute have completed the project work entitled “**CLI Command Generation Using Generative AI**” and submitted a satisfactory work in this report. Hence recommended for the partial fulfillment of degree of Bachelor of Engineering in Computer Science and Engineering.

*for P. Sable*  
Internal Examiner

*Kalyani P. Sable*  
Name and Signature

Date: 10/5/24.

*K. Sawadkar*  
External Examiner

*Krushna Y. Sawadkar*  
Name and Signature

Date: 10/05/24

## Acknowledgement

It is our utmost duty and desire to express gratitude to various people who have rendered valuable guidance during our project work. We would have never succeeded in completing our task without the cooperation, encouragement and help provided to us by them. There are a number of people who deserve recognition for their unwavering support and guidance throughout this report.

We are highly indebted to our guide **Dr. P. V. Deshmukh** for her guidance and constant supervision as well as for providing necessary information from time to time. We would like to take this opportunity to express our sincere thanks, for her esteemed guidance and encouragement. Her suggestions broaden our vision and guided us to succeed in this work.

We are sincerely thankful to **Dr. J. M. Patil** (HOD, CSE Department, SSGMCE, Shegaon), and to **Dr. S. B. Somani** (Principal, SSGMCE, Shegaon) who always has been kind to extend their support and help whenever needed.

We would like to thank all teaching and non-teaching staff of the department for their cooperation and help. Our deepest thank to our parents and friends who have consistently assisted us towards successful completion of our work.

### **Projectees**

**Atharva Tattu**

**Prajwal Chitode**

**Rushikesh Dhawne**

**Vedant Chaudhari**

## ABSTRACT

The integration of generative Artificial Intelligence (AI) in Command Line Interface (CLI) command generation represents a transformative leap in computing, promising to democratize access to powerful tools and streamline interactions between users and systems. Traditionally, the CLI has been lauded for its efficiency but criticized for its steep learning curve and syntax complexity, hindering its usability and accessibility. However, with the advent of generative AI, this paradigm is shifting. Through machine learning algorithms, generative models analyze vast repositories of CLI commands, learning semantic relationships and syntactic structures to generate coherent and contextually relevant commands in real-time. This breakthrough empowers users of all skill levels to interact with CLI environments more intuitively, reducing the cognitive load associated with command formulation. The benefits of AI-driven CLI command generation are manifold. It adapts to user preferences and contextual cues, automating repetitive tasks and accelerating workflows in software development, system administration, and data analysis. For developers, AI-generated commands streamline debugging processes and improve code quality, while system administrators can automate routine maintenance tasks and enhance system security. In data analysis and machine learning, AI-driven CLI tools expedite data preprocessing, model training, and experimentation workflows, facilitating insights and discoveries. However, the integration of generative AI in CLI environments also presents challenges. Usability, accuracy, and security are paramount considerations, necessitating robust validation mechanisms and safeguards to maintain reliability and safety. Moreover, efforts to enhance the interpretability and explainability of AI-generated commands foster user trust and understanding. In conclusion, the convergence of CLI and generative AI represents a significant advancement in human-computer interaction, with the potential to reshape workflows and accelerate innovation across industries, while ensuring accessibility and usability for users worldwide.

***Keywords: Command Line Interface, Artificial Intelligence, AI-Driven Tools.***

## Contents

Particulars	Page No.
Abstract	i
Contents	ii
List of Abbreviations and Symbols	iii
List of Figures and Tables	iv
<b>Chapter -1: Introduction</b>	<b>01</b>
1.1 Overview	01
1.2 Background and Significance of the Problem	03
1.3 Aims of research work study	03
1.4 Objectives and scope of the work	04
<b>Chapter -2: Literature Review</b>	<b>05</b>
<b>Chapter -3: Methodology</b>	<b>08</b>
3.1 System Architecture	08
3.2 ChatGPT API request and response	10
3.3 How to use Application	12
3.4 Tech Used	13
<b>Chapter -4: Implementation</b>	<b>14</b>
4.1 Code Snippets	14
<b>Chapter -5: Result and Discussion</b>	<b>21</b>
5.1 Results	21
<b>Chapter -6: Conclusion</b>	<b>31</b>
<b>Chapter -7: Future Scope</b>	<b>32</b>
<b>References</b>	<b>34</b>
<b>Dissemination of work</b>	<b>36</b>
<b>Plagiarism Report</b>	

## List of Abbreviations and Symbols

---

Symbol/Abbreviation	Particulars
<i>API</i>	Application Programming Interface
<i>CLI</i>	Command Line Interface
<i>LSTM</i>	Long Short-Term Memory Networks
<i>ONNX</i>	Open Neural Network Exchange
<i>MXNet</i>	Mixed Network
<i>IoT</i>	Internet of Things
<i>AI</i>	AI: Artificial Intelligence
<i>NLP</i>	Natural Language Processing
<i>LLM</i>	Large Language Models
<i>UAVs</i>	Unmanned Aerial Vehicles

## **List of Figures and Tables**

<b>Sr. No.</b>	<b>Particulars</b>	<b>Page No.</b>
3.1	Proposed System Overview	08
3.2	Interaction with ChatGPT	10
3.3	Steps to use Application	12
4.1	Main file Code	14
4.2	Using OraPromise Code	14
4.3	Looping the Responses Code	15
4.4	Sending To ChatGpt Code	16
4.5	Message Specification Code	16
4.6	About Us Code	17
4.7	Accessing Shell Code	18
4.8	Executing Shell Commands Code	19
5.1	Input List all files Input	21
5.2	Main file Code	21
5.2	Output for List all files	23
5.3	Input for Finding Pattern in file	23
5.4	Output for Finding Pattern in file	25
5.5	Creating Html File	25
5.6	Input for downloading Packages	29
5.7	Output for downloading Packages	29



# **CHAPTER 01**

## **INTRODUCTION**

# 1.INTRODUCTION

## 1.1 OVERVIEW

Command-line interfaces serve as a fundamental tool for system administrators, developers, and power users alike, enabling swift and precise control over computing resources. Despite their power and versatility, CLI commands can be intimidating, especially for newcomers to the command-line environment. Generative AI offers a novel approach to alleviate this complexity by leveraging machine learning algorithms to automatically generate CLI commands based on user intent and contextual cues. By harnessing the power of natural language processing (NLP) and pattern recognition, generative AI models can interpret user commands, infer desired actions, and construct syntactically correct CLI instructions, thereby enhancing usability and accessibility.

In contemporary computing environments, the command-line interface (CLI) remains a fundamental tool for interacting with operating systems and software applications. Yet, the manual construction of CLI commands presents a significant hurdle, demanding users to possess extensive knowledge of command syntax and system functionalities. This reliance on manual crafting often serves as a bottleneck, impeding productivity and hindering accessibility for less experienced users. To address this challenge, the integration of generative artificial intelligence (AI) techniques has emerged as a promising solution. By leveraging AI, users can automate and optimize the process of CLI command generation, thereby streamlining interactions with computing systems. Natural language processing (NLP) models, trained on vast repositories of CLI documentation and usage examples, enable users to express their intentions in natural language, with the AI system converting these inputs into precise CLI commands.

Furthermore, machine learning algorithms can analyze patterns in historical CLI commands and user behaviors to offer real-time suggestions or auto-complete commands, enhancing user productivity and reducing errors. Additionally, reinforcement learning methods can continuously refine CLI command generation based on user feedback, ensuring adaptability to evolving user needs and preferences. In essence, the integration of AI into CLI environments holds the potential to

democratize access to command-line functionalities, making them more intuitive and efficient for a broader spectrum of users, regardless of their expertise level.

One of the key advantages of AI-driven CLI command generation is its ability to adapt to user preferences and contextual cues. Through natural language processing (NLP) techniques, generative models can interpret user input, infer intent, and generate commands tailored to specific tasks or objectives. This level of customization not only enhances user productivity but also fosters a more seamless and fluid interaction between humans and machines.

Furthermore, the integration of generative AI in CLI environments has profound implications for various industries and domains. In software development, for instance, developers can leverage AI-generated commands to automate repetitive tasks, accelerate debugging processes, and improve code quality. System administrators can benefit from AI-driven CLI tools by automating routine maintenance tasks, optimizing resource allocation, and enhancing system security. In the realm of data analysis and machine learning, generative AI holds promise for streamlining data preprocessing, model training, and experimentation workflows. By generating complex CLI commands based on data context and analysis requirements, data scientists and researchers can expedite the exploration and interpretation of large datasets, facilitating insights and discoveries.

However, while the potential of AI-driven CLI command generation is undeniable, it also raises important considerations regarding usability, accuracy, and security. As with any AI-powered solution, robust validation mechanisms and safeguards must be implemented to ensure the reliability and safety of generated commands. Additionally, efforts should be made to enhance the interpretability and explainability of AI-generated commands, enabling users to understand the rationale behind each recommendation and fostering trust in the system. In conclusion, the convergence of CLI and generative AI represents a significant milestone in the evolution of human-computer interaction. By automating command generation and enhancing user experiences, AI-driven CLI tools have the potential to democratize access to powerful computing resources and accelerate innovation across industries. This report endeavors to provide a comprehensive exploration of this transformative technology, shedding light on its benefits, challenges, and future directions in the realm of CLI command generation.

## **1.2 BACKGROUND AND SIGNIFICANCE OF THE PROBLEM**

The proliferation of CLI-based tools underscores the need to optimize command generation processes to meet diverse user needs. However, manually crafting CLI commands poses challenges: complexity requires a deep understanding of syntax and configurations, hindering novice users; iterative refinement is time-consuming, especially for complex tasks, leading to productivity issues; and human errors in formulation can result in unintended consequences or security risks, necessitating automated validation. Addressing these challenges is crucial for improving user productivity, fostering CLI adoption, and creating a more inclusive computing environment.

## **1.3 AIMS OF RESEARCH OF WORK STUDY**

The primary focus of this research is to explore the viability and effectiveness of integrating generative AI techniques into CLI command generation, with specific objectives in mind. Firstly, we aim to delve into state-of-the-art generative AI algorithms like recurrent neural networks (RNNs), transformer models, and reinforcement learning methods to identify the most suitable methodologies for CLI command synthesis. Secondly, we intend to curate a diverse dataset of CLI commands sourced from open-source repositories, software documentation, and user contributions, ensuring comprehensive coverage of command syntaxes, operating systems, and software domains. Thirdly, our goal is to develop and train generative AI models capable of comprehending user intents and producing contextually relevant CLI commands, utilizing techniques such as sequence-to-sequence learning, attention mechanisms, and transfer learning. Subsequently, we plan to evaluate the performance of these models rigorously through testing against benchmark datasets, user simulations, and real-world usage scenarios, assessing criteria like command accuracy, syntactic correctness, and user satisfaction. Finally, we aim to explore practical applications of generative AI-generated CLI commands across various use cases, including system administration, software development, data analysis, and automation workflows, to showcase the versatility and utility of our proposed approach.

## 1.4 OBJECTIVES AND SCOPE OF WORK

- ❖ To invent novel generative AI algorithms tailored for CLI command generation, optimizing for accuracy, efficiency, and scalability.
- ❖ To conduct comprehensive evaluations of developed models using standard metrics and user feedback, iteratively refining model architectures and training methodologies.
- ❖ To explore potential applications of generative AI-generated CLI commands in various domains, ranging from cloud computing and DevOps to scientific computing and cybersecurity.
- ❖ To provide practical guidelines and best practices for integrating generative AI-based CLI command generation into existing software ecosystems, fostering adoption and interoperability.

# **CHAPTER 02**

## **LITERATURE REVIEW**



## **2. LITERATURE REVIEW**

Partha Pratim Ray (2023). the author proposes an extensive examination into the transformative influence wielded by ChatGPT, an advanced AI language model, across a diverse array of sectors, including scientific research, customer service, healthcare, and education. In his comprehensive review, Ray meticulously traces the lineage of ChatGPT, delving into its origins, technological intricacies, and key developmental milestones, with a particular emphasis on its evolution from the Generative Pre-trained Transformer (GPT) architecture and its distinctive features compared to models such as Generative Adversarial Network (GAN). Addressing significant challenges inherent in its deployment, Ray navigates ethical concerns and data biases while offering pragmatic strategies for mitigation, thereby paving the way for responsible AI development. Looking towards the future, Ray envisages a landscape where ChatGPT seamlessly integrates with complementary technologies, enhances human-AI interaction, and contributes to narrowing the digital divide. Despite lingering controversies, ChatGPT remains a focal point for its potential to revolutionize research and industry practices, as Ray's insightful analysis underscores the enduring allure and profound implications of its advancement.[1]

Dinesh Kalla's (2023) the author intricately delves into profound impact of ChatGPT, an innovative AI technology developed by OpenAI, across diverse domains. The survey meticulously traces ChatGPT's origins within the Generative Pre-trained Transformer (GPT) architecture, elucidating its operational framework, deep neural network architectures, and iterative training methodologies. Through compelling case studies and real-world applications, it highlights ChatGPT's transformative influence in academia, industry, and beyond, showcasing its role in enhancing productivity, fostering collaboration, and reshaping communication paradigms. While acknowledging its advantages in natural language generation and scalability, the survey candidly addresses challenges such as biases in responses and limitations in emotional intelligence, underscoring the importance of ongoing research and ethical considerations. Looking ahead, the survey envisions promising opportunities for innovation and collaboration, positioning ChatGPT as a catalyst for future

advancements in AI-driven conversational agents, thus serving as a guiding beacon for balanced AI development and deployment.[2]

Devadas Menon and K. Shilpa (2023) .the author publish a qualitative investigation delving into the determinants influencing users' acceptance and utilization of OpenAI's ChatGPT, leveraging the Unified Theory of Acceptance and Usage of Technology (UTAUT) model. Through insightful semi-structured interviews involving 32 users from India, the research discerns pivotal factors dictating users' interactions with ChatGPT. Alongside fundamental UTAUT constructs such as performance expectancy and effort expectancy, the study identifies additional dimensions like perceived interactivity and privacy concerns as influential in users' engagement with ChatGPT. Moreover, the analysis unveils that age and experience play a moderating role, shaping the significance of these factors in users' decision-making processes. Importantly, the implications of this study extend beyond mere user behavior analysis, offering actionable insights for developers to refine ChatGPT's design and functionality, thereby enhancing its usability and adoption. Furthermore, by contributing to the burgeoning literature on AI technology acceptance, this research informs broader discussions surrounding the societal integration of advanced AI-driven solutions like ChatGPT, underscoring its potential impact on various facets of contemporary life.[3]

Tom Fellmann and Manolya Kavakli (2007),the author delve into the comparative utility of Command Line Interfaces (CLIs) versus Graphical User Interfaces (GUIs) in the development of Virtual Reality (VR) systems, with a specific focus on the coding expertise and differing approaches of novice and expert programmers. Their paper meticulously presents a comparative analysis of the advantages and disadvantages inherent in both interfaces, considering factors such as ease of use, productivity, resource consumption, and scriptability. While acknowledging the proficiency of expert programmers in CLI environments, the study underscores the necessity for GUIs, even for skilled developers, in simplifying the complexity of VR systems and enhancing user efficiency through features like pre-initialized settings and logical order presentation. Furthermore, it highlights the importance of adaptability in GUI design to cater to varying levels of programming proficiency. Through this investigation, Fellmann and Kavakli contribute valuable insights into understanding interface

preferences and requirements in VR development, offering guidance on optimizing programming environments to accommodate diverse user groups effectively.[4]

Volker Bilgram and Felix Laarmann (2023) explores the transformative impact of generative Artificial Intelligence (AI) on innovation management, particularly focusing on the early phases of exploration, ideation, and digital prototyping. Through experimentation with large language models (LLMs), such as the generative pretrained transformer (GPT), the authors demonstrate how AI can augment innovation processes, democratizing access to AI tools and reshaping workflows. The research draws on six months of intensive experimentation with LLMs in both internal development and client projects, offering concrete examples and first-hand experiences of AI-assisted approaches. The study highlights the versatility of generative AI in various innovation tasks, from user journey mapping to idea generation and prototyping, and underscores its potential to revolutionize knowledge management systems. Moreover, the authors argue that generative AI, by enabling faster iterations and reducing costs in early prototyping, could fundamentally change the innovation landscape. Despite the initial skepticism about AI's suitability for creative tasks, the emergence of transformer language models has shifted perceptions, positioning AI as a central tool for idea generation and innovation. However, while the potential benefits of AI-augmented innovation management are evident, research on its practical application in corporate settings remains limited, emphasizing the need for further exploration and analysis in this emerging field.[5]

# **CHAPTER 03**

## **METHODOLOGY**

### 3.METHODOLOGY

#### 3.1 SYSTEM ARCHITECTURE OVERVIEW

The system architecture gives an overview of the working of the system. The working of this system is shown below:

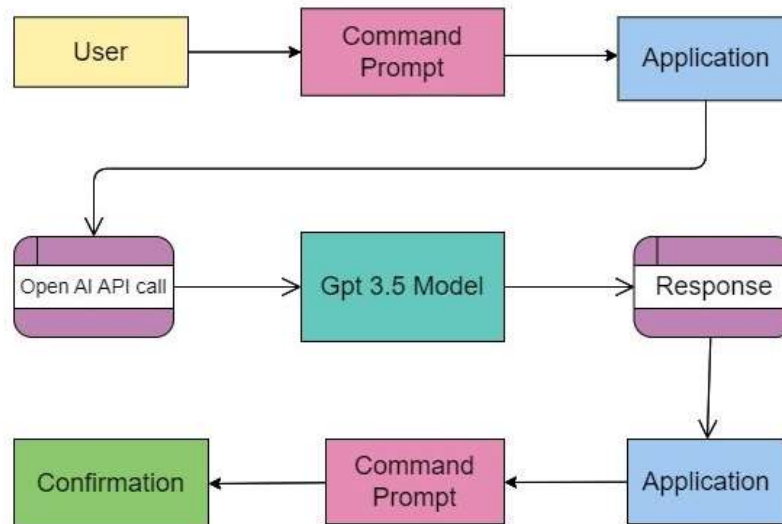


Figure 3.1 Proposed System Overview

##### User Interaction:

The interaction begins when a user engages with your Command Line Interface (CLI) application. This interaction typically involves the user entering a command or providing input through the terminal interface.

##### Command Processing:

Upon receiving the user's command or input, your application starts the processing phase. This step may involve parsing the input, validating it for correctness or completeness, and preparing it for further processing.

##### OpenAI API Call:

After processing the command, your application initiates an API call to the OpenAI API, specifically targeting the GPT-3.5 Turbo model. This call is the bridge between your application and the powerful natural language processing capabilities offered by GPT-3.5 Turbo.

**Input to GPT-3.5 Turbo:**

The processed command or input from your application serves as the prompt for the GPT-3.5 Turbo model. This prompt is formulated to encapsulate the user's intent or question in a way that GPT-3.5 Turbo can understand and generate a meaningful response.

**Response Generation:**

GPT-3.5 Turbo processes the input prompt using its advanced language model and contextual understanding. It generates a response based on the prompt, leveraging its vast knowledge and learning from extensive training data.

**Response Handling:**

Upon receiving the response from GPT-3.5 Turbo, your application proceeds to handle the output. This step involves processing the AI-generated response, which may include filtering out irrelevant information, formatting the output for presentation, or extracting key insights from the response.

**Output Processing:**

The processed output from GPT-3.5 Turbo, now refined by your application, is ready for further processing or presentation. Depending on the nature of the command and response, your application may perform additional logic or transformations to prepare the output for the user.

**Confirmation and Feedback:**

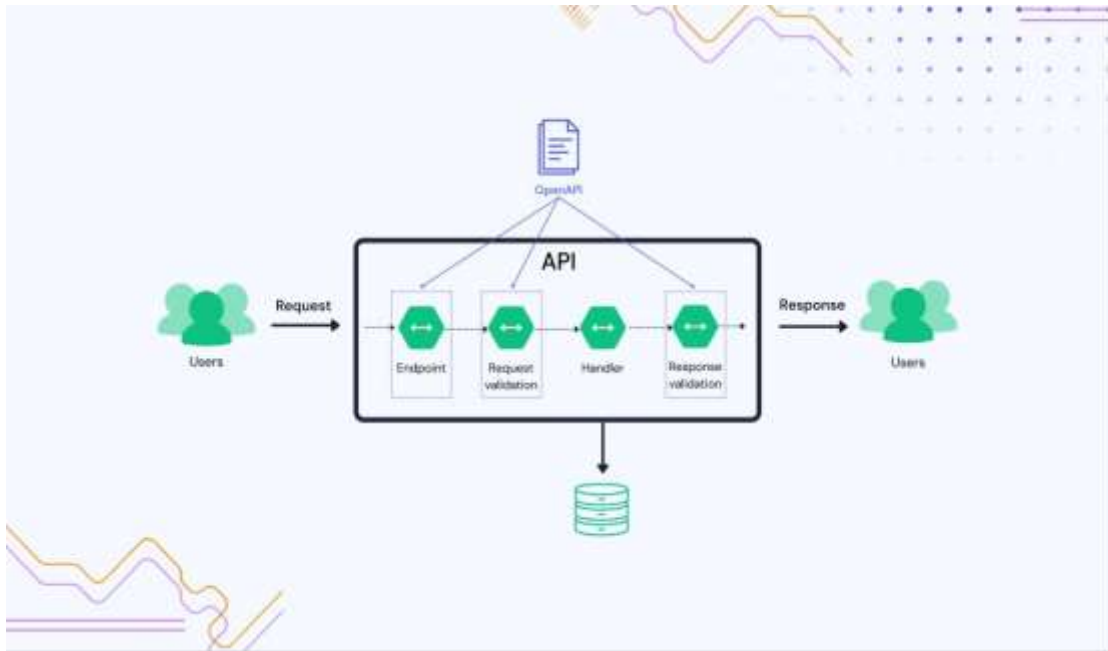
With the processed output in hand, your application generates a meaningful confirmation or feedback message for the user. This message could include the results of a computation, a summary of information, an action taken based on the input, or any other relevant response.

**Display to User:**

Finally, the confirmation or feedback message generated by your application is displayed to the user via the CLI interface. This interaction loop closes the communication cycle, providing the user with the requested information or acknowledging their command's execution.



## 3.2 CHATGPT API REQUEST AND RESPONSE



**Figure 3.2 Interaction with ChatGpt**

The image depicts a typical API (Application Programming Interface) architecture and its flow. The main components represented are:

1. **Users:** The entities that send requests to the API and receive responses from it.
2. **API:** The core component that handles user requests and responses. It consists of the following sub-components:
  - **Endpoint:** The entry point where user requests are received.
  - **Request validation:** This component validates the incoming requests to ensure they are well-formed and meet the required criteria.
  - **Handler:** This component processes the validated requests and performs the necessary operations.
  - **Response validation:** This component validates the responses generated by the handler before sending them back to the users.
3. **OpenAPI:** This component represents the OpenAPI specification, which is a standardized way to define and document APIs.

4. **Database:** The API undertakes execution of its functions by interacting with the database such that the database stores and retrieves data during request processing. The flow illustrated in the diagram is as follows:

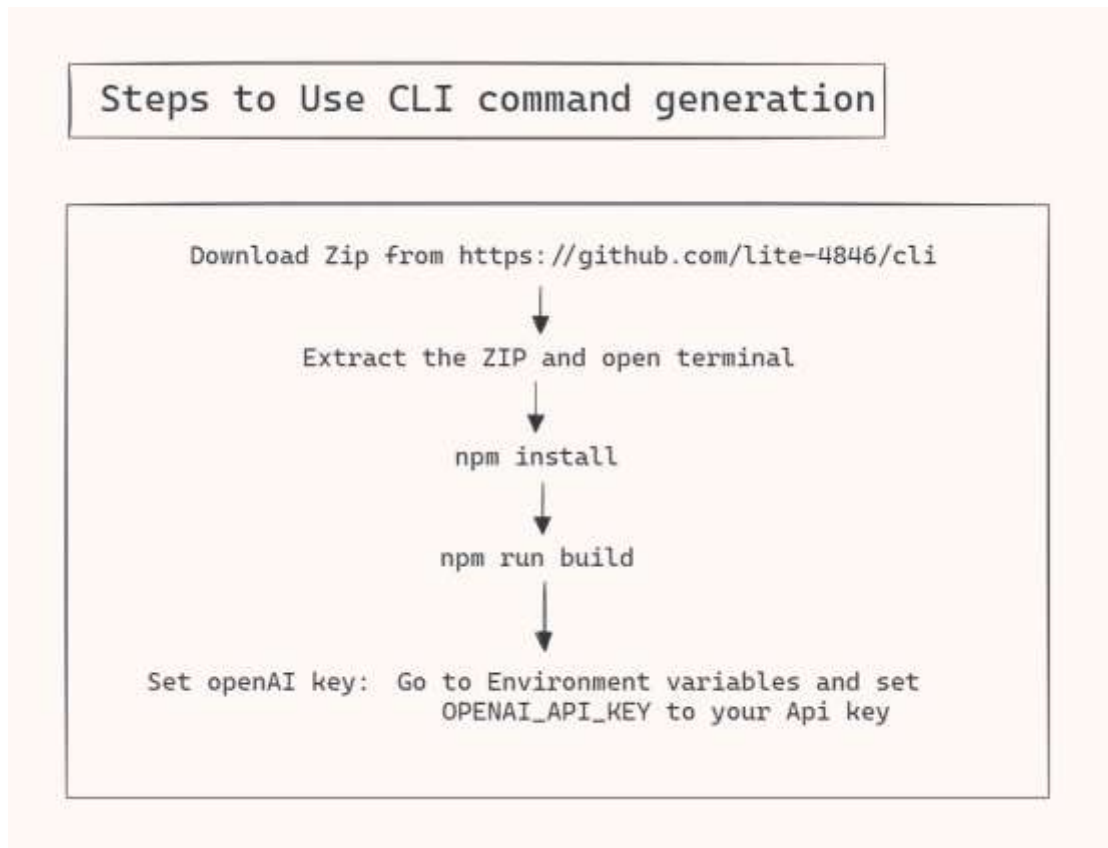
The flow illustrated in the diagram is as follows:

1. The users start making the API calls.
2. The requests are destined to land at the endpoint.
3. Subsequently, the requests use Back-end Validation for authenticating if they are valid.
4. The marked requests then push to the Handler for the processing.
5. The Hands of the Modules which have been demarcated are to be validated during the Response generation.
6. The action is at the discretion of the Users.

This diagram offers a simple yet detailed view of a logical structure of an API and the significant elements within this design: the incoming of user requests, responding to them, interaction with the database and the Open API format for API documentation. Problem Statement: The purpose of the project is to create a Node.js software that connects a command-line interface (CLI) with OpenAI's GPT-3 API. The application should be able to communicate with users via the command-line interface and have the features of the interface so that is easy to access the GPT-3's natural language processing functions and receives replies. System Components: Node.js app with CLI app: The main system that handles the flow of input between the CLI, the application logic, and the API OpenAI GPT-3 API: The service which is bought from outside is dedicated to the consideration of natural language and the generating of replies on a basis of user input.

System Requirements: Functionalities: The system has to offer the user to type in a query text through the CLI interface. Security: Make sure implementation of API keys and data door care. Error Handling: Take care with reverting error handling mechanism to the proper level which is unexpected occurrences. User Experience: The graphical user interface of the CLI interface should help users to perform operations easily in a logical manner as quick responses and direct feedback are expected from users.

### 3.3 HOW TO USE APPLICATION



**Figure 3.3 Step to use Application**

This image outlines the steps to use a command-line interface (CLI) command generation tool. Here's a detailed explanation of the flow:

1. Download Zip from <https://github.com/lite-4846/cli> - The first step is to download a ZIP file containing the CLI tool from the specified GitHub repository.
2. Extract the ZIP and open terminal - After downloading the ZIP file, you need to extract its contents, and then open a terminal or command prompt on your system.
3. npm install - In the terminal, run the command `npm install`. This command installs all the required dependencies and packages for the CLI tool to work correctly.
4. npm run build - After installing the dependencies, run the command `npm run build`. This command builds and compiles the CLI tool, making it ready for use.

5. Set openAI key : Go to Environment variables and set OPENAI\_API\_KEY to your Api key - The CLI tool likely integrates with the OpenAI API for certain functionalities. To use the API, you need to obtain an API key from OpenAI and set it as an environment variable named OPENAI\_API\_KEY. This step ensures that the CLI tool can authenticate with the OpenAI API and access its services.

By following these steps, you will have the CLI command generation tool set up and ready to use on your system. The tool can then be used to generate various commands or perform other tasks related to its functionality.

### 3.4 TECH USED

**Node.js:** Node.js is a runtime environment that allows you to run JavaScript code outside the browser, making it suitable for server-side applications and command-line tools.

**TypeScript:** TypeScript is a superset of JavaScript that adds static typing and other features to help developers write more robust and maintainable code, especially in large-scale projects.

**Enquirer:** Enquirer is a lightweight library for creating interactive command-line prompts in Node.js, providing a simple and elegant way to gather user input.

**oraPromise:** oraPromise is a promise-based wrapper for the ora library, which allows you to display elegant loading spinners and messages in the terminal while waiting for asynchronous operations to complete.

**OpenAI:** OpenAI provides powerful artificial intelligence (AI) models and APIs, such as GPT-3, for natural language understanding and generation tasks, enabling developers to integrate AI capabilities into their applications.

**Chalk:** Chalk is a library for styling and coloring terminal output in Node.js, offering an easy way to add colors, styles, and formatting to text displayed in the command-line interface for improved readability and visual appeal.

# **CHAPTER 04**

## **IMPLEMENTATION**

## 4.IMPLEMENTATION

### 4.1 CODE SNIPPETS

```

import { about, version } from './about.js';
import { commands } from './commands/index.js';
import { whatCommandToRun } from './completion.js';
import { isOpenAiKeyInEnvironment } from './utils.js';
import { oraPromise } from 'ora';

export async function main(argv: string[]) {
  const [, ...args] = argv;
  const command = args[0];

  if (!command || ['help', '--help', '-h'].includes(command)) {
    console.log(about);
    return;
  }

  if (['version', '--version', '-v'].includes(command)) {
    console.log(version);
    return;
  }

  if (!isOpenAiKeyInEnvironment) {
    console.log(
      'OpenAI API key is not set. Please set OPENAI_API_KEY in your environment.'
    );
    return;
  }

  const content = args.join(' ');

```

Figure 4.1 Main File Code

```

1  const response = await oraPromise(
2    whatCommandToRun([
3      {
4        role: "user",
5        content,
6      },
7    ]),
8    "cli is thinking..."
9  );

```

Figure 4.2 Using Orapromise Code





Figure 4.3 Looping the Responses Code

This code is a TypeScript module that serves as the main entry point for a CLI application. Here's a brief description of its functionality:

**Imports:** It imports various modules such as `about`, `version`, `commands`, `whatCommandToRun`, `isOpenAiKeyInEnvironment`, and `oraPromise` from different files.

**Main Function:** The main function is an asynchronous function that takes command-line arguments (`argv`) as input. It extracts the command and arguments from the input, checks for special commands like "help" and "version," and displays relevant information if needed. It checks if the OpenAI API key is set in the environment variables; if not, it prompts the user to set it. It joins the remaining arguments into content and calls `whatCommandToRun` function to determine the appropriate action based on user input. It uses `oraPromise` to display a loading spinner while waiting for the response from `whatCommandToRun`.

**Processing Response:** It loops through the choices in the response from `whatCommandToRun`. If a choice contains a `function_call`, it attempts to execute the

corresponding function from the commands module based on the function name provided. If a choice indicates a stop or length limit, it displays a relevant message.

```
import OpenAI from "openai";
import process from "node:process";
import { cliLLMModel } from "../utils.js";

export async function whatCommandToRun(
  messages: OpenAI.Chat.ChatCompletionMessageParam[]
) {
  const openai = new OpenAI();
  const response = await openai.chat.completions.create({
    model: cliLLMModel,
    function_call: { name: "shell" },
  });

  return response;
}
```

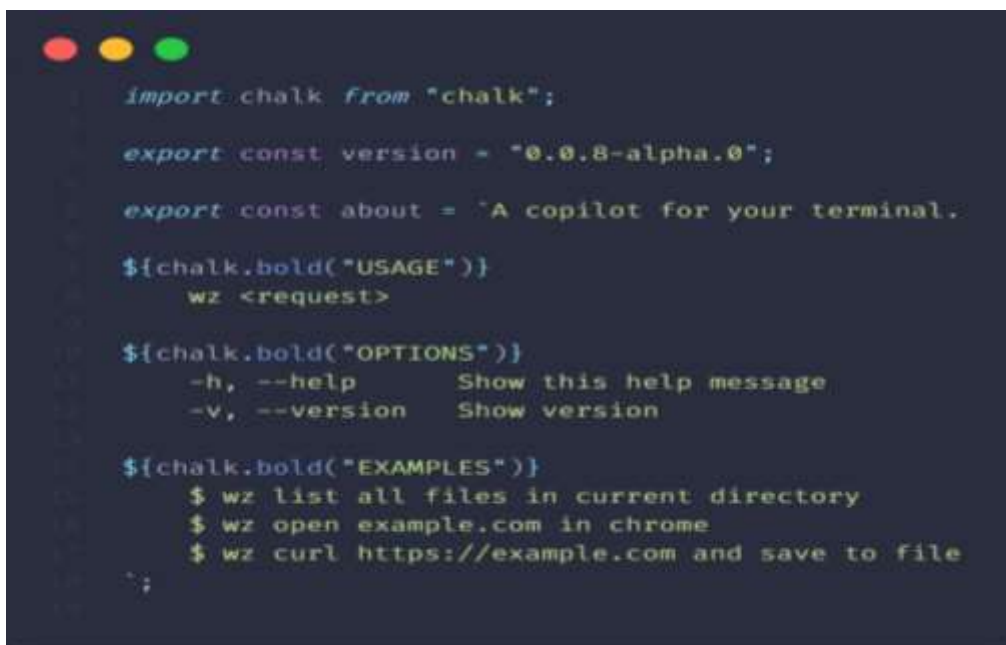
Figure 4.4 Sending to ChatGPT Code

```
messages: [
  {
    role: "system",
    content: `OS: ${process.platform}
Arch: ${process.arch}
shell: ${process.env.SHELL?.split("/").pop()}
You MUST NOT use functions that are not available.`,
  },
  ...messages,
],
temperature: 0,
functions: [
  {
    name: "shell",
    description: "Run a shell command",
    parameters: {
      type: "object",
      properties: {
        description: {
          type: "string",
          description:
            "A description of what this command will do. If it will cause any side effects, highlight with a warning.",
        },
        command: {
          type: "string",
          description:
            "The shell command to run (e.g. ls -l). Use an appropriate command based on the shell and OS.",
        },
      },
      required: ["description", "command"],
    },
  },
],
```

Figure 4.5 Message Specification Code

This code defines an asynchronous function `whatCommandToRun` that interacts with the OpenAI API to determine the appropriate command to execute based on the input messages. Here's a breakdown of what it does:

- Imports:** It imports the `OpenAI` module for interfacing with the OpenAI API, the `process` module for environment information, and the `cliLLMModel` from the `utils.js` file.
- Function Definition:** The `whatCommandToRun` function takes an array of messages as input, which are parameters for generating completions in the OpenAI chat model.
- OpenAI Interaction:** It creates a new instance of the `OpenAI` class. It calls the `openai.chat.completions.create` method to generate completions based on the specified parameters. The `model` parameter (`cliLLMModel`) specifies the language model to use for completions. The `messages` parameter includes system information (OS, Arch, shell, etc.) and user-provided messages to generate completions. The `temperature` parameter controls the randomness of the completions (set to 0 for deterministic completions). The `functions` array defines available functions that can be suggested in completions. In this case, it includes a `shell` function.
- Function Call:** The `function_call` parameter specifies which function to call (`shell` function in this case).



```
import chalk from "chalk";

export const version = "0.0.8-alpha.0";

export const about = "A copilot for your terminal."

${chalk.bold("USAGE")}
  wz <request>

${chalk.bold("OPTIONS")}
  -h, --help      Show this help message
  -v, --version   Show version

${chalk.bold("EXAMPLES")}
  $ wz list all files in current directory
  $ wz open example.com in chrome
  $ wz curl https://example.com and save to file
;
```

**Figure 4.6 About Us Code**

This code defines two constants `version` and `about`, utilizing the `chalk` library for colorful formatting in the CLI. Here's what each part does:

**Import Chalk:**

It imports the chalk library for styling and coloring terminal output.

**Version Constant:**

The version constant holds a string value representing the version of the application ("0.0.8-alpha.0" in this case).

**About Constant:**

The about constant contains a multiline string describing the application and its usage.

It uses chalk.bold to format specific parts of the text in bold for emphasis.

It provides information about usage, options (help and version), and examples using the application (wz in this context).

**Usage of Chalk:**

Chalk's formatting methods (chalk.bold) are used to enhance the appearance of the text in the about constant.

These formatting options help in presenting information clearly and making certain parts of the text stand out (such as command examples or important messages).

Overall, this code snippet is part of a CLI application and serves to define constants for version information (version) and an about/help message (about) with styled formatting using the chalk library for better visual presentation in the terminal.

A screenshot of a code editor window with a dark background and light-colored text. The code is written in TypeScript and shows the import of a shell module and the definition of a commands object. The code is as follows:

```
import { shell } from "./shell.js";

export const commands: Record<string, (args: any) => unknown | Promise<unknown>> = {
  shell,
};
```

**Figure 4.7 Accessing Shell Code**

---

Department of Computer Science Engineering SSGMCE, Shegaon

Page 24



Figure 4.8 Executing Shell Command Code

This code snippet defines a function `shell` that uses the `Enquirer` library to prompt the user for confirmation before executing a shell command. Here's a breakdown of how it works:

1. **Importing Dependencies:** - It imports the `Enquirer` library for creating interactive prompts and the `executeShellCommand` function from the `utils.js` file.
2. **ShellOptions Interface:** - It defines an interface `ShellOptions` representing the options required for running a shell command, such as `command` (the actual command to execute) and `description` (a description of what the command does).

3. **\*\*shell Function:\*\*** - The ``shell`` function takes an object of type ``ShellOptions`` as input. - It uses ``Enquirer.prompt`` to create a confirmation prompt (``confirm`` type) asking the user if they want to run the provided command. - The confirmation message includes the provided command and its description. - It waits for the user's confirmation response (``confirm: boolean``).
4. **\*\*User Confirmation Handling:\*\*** - If the user confirms (``response.confirm`` is ``true``), the ``executeShellCommand`` function is called with the provided command. - If the user cancels the operation (``response.confirm`` is ``false``), it displays "Aborting..." and exits the function. This ``shell`` function adds an interactive layer to the CLI application, ensuring that users confirm before potentially executing sensitive or critical shell commands. It enhances usability and prevents accidental command execution.



# **CHAPTER 05**

## **RESULT AND DISCUSSION**

## 5. RESULT AND DISCUSSION

### 5.1 RESULT

#### List All Files :-

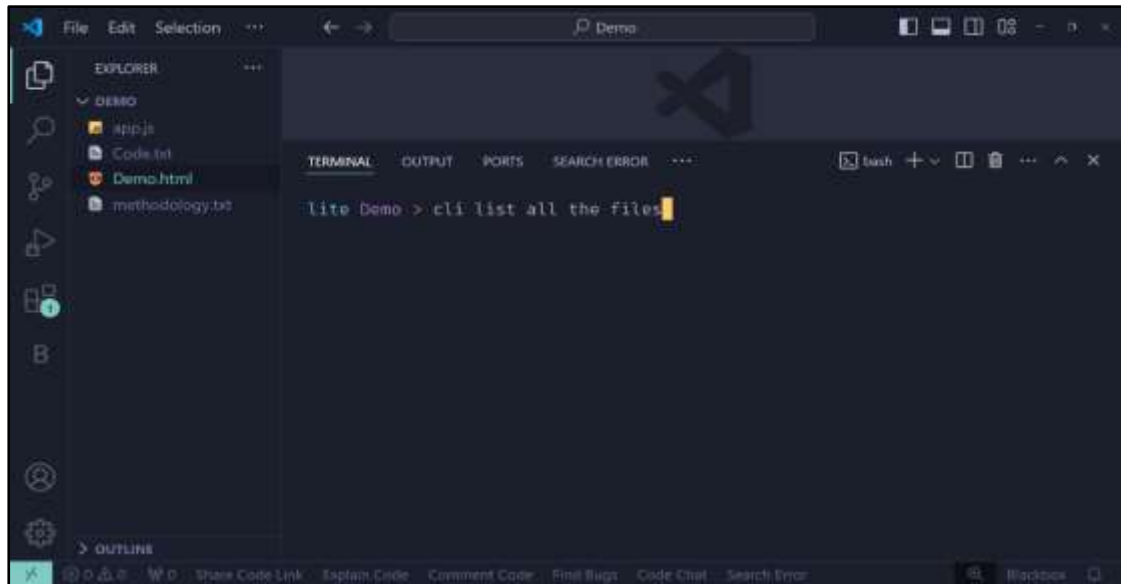


Figure 5.1 Input List all Files Input

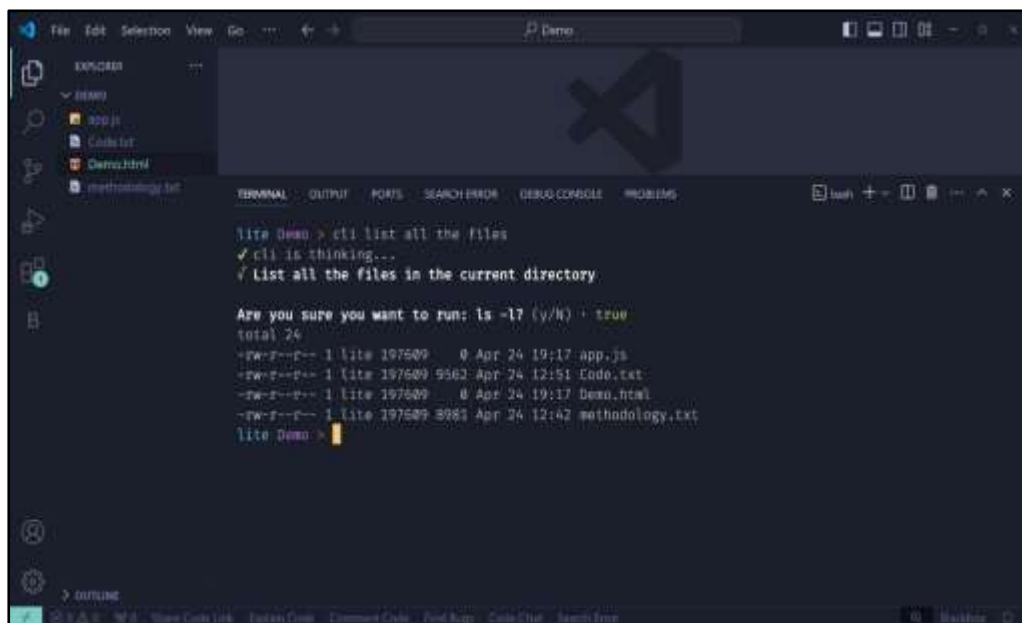


Figure 5.2 Output for List of Files

The image shows the terminal output after running the command "cli list all the files" in an integrated development environment (IDE) or code editor.

The terminal first acknowledges that it is processing the command by displaying "✓ cli is thinking..." and "✓ List all the files in the current directory". It then prompts the user with "Are you sure you want to run: ls -l? (y/n)" and the user has responded with "true", confirming the execution of the command.

The output then displays the list of files in the current directory ("Demo"), along with their permissions, ownership, size, and modification dates. The files listed are:

1. app.js (0 bytes, modified on Apr 24 19:17)
2. Code.txt (9562 bytes, modified on Apr 24 12:51)
3. Demo.html (0 bytes, modified on Apr 24 19:17)
4. methodology.txt (8981 bytes, modified on Apr 24 12:42)

The file listing is presented in a long format (indicated by the "-l" option in the "ls" command), which provides detailed information about each file, such as permissions, owner, group, size, and timestamps.

After displaying the file list, the terminal cursor is positioned on a new line, ready for the next command input.

## Create java File :-



Figure 5.3 Input for Creating Java File

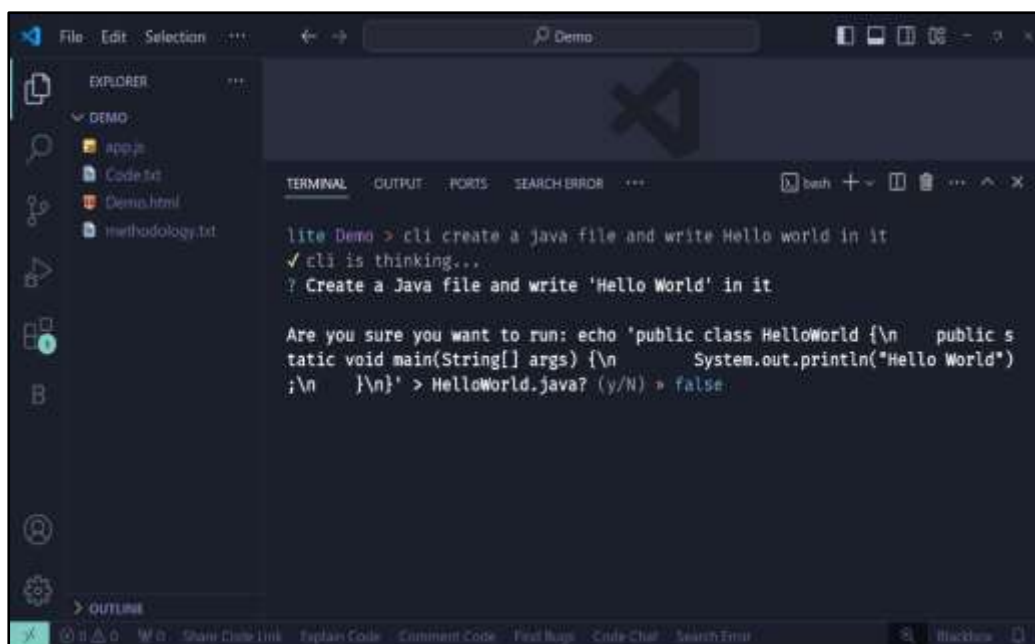


Figure 5.4 Output for Creating Java File

The image shows the continuation of the previous command "cli create a Java file and write Hello world in it" in the integrated development environment (IDE) or code editor.

After acknowledging the command and confirming that it is processing the request ("✓ cli is thinking..." and "? Create a Java file and write 'Hello World' in it"), the IDE is proposing to create a Java file named "HelloWorld.java" with the following contents:

```
java
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello World");
    }
}
```

This is a basic Java program that defines a class named "HelloWorld" with a main method. When executed, this program will print the string "Hello World" to the console or output.

The IDE is prompting the user with the message "Are you sure you want to run: echo 'public class HelloWorld {\n public static void main(String[] args) {\n System.out.println("Hello World")\n }\n}' > HelloWorld.java? (y/n)". This indicates that the IDE is seeking confirmation from the user to create the "HelloWorld.java" file with the proposed Java code.

The user has responded with "false", which means they have declined or rejected the creation of the file with the provided code.

So, at this point, the IDE has not yet created the Java file as per the original command, and it is awaiting further input or instructions from the user.

## Find Pattern in Files :-

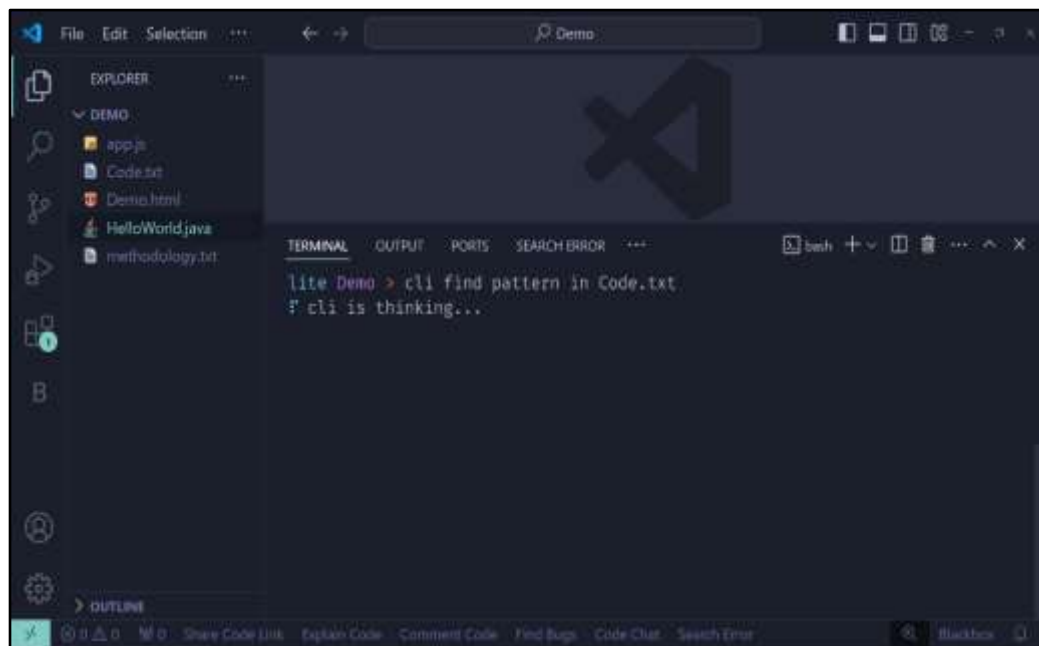


Figure 5.5 Input for Finding Pattern In File

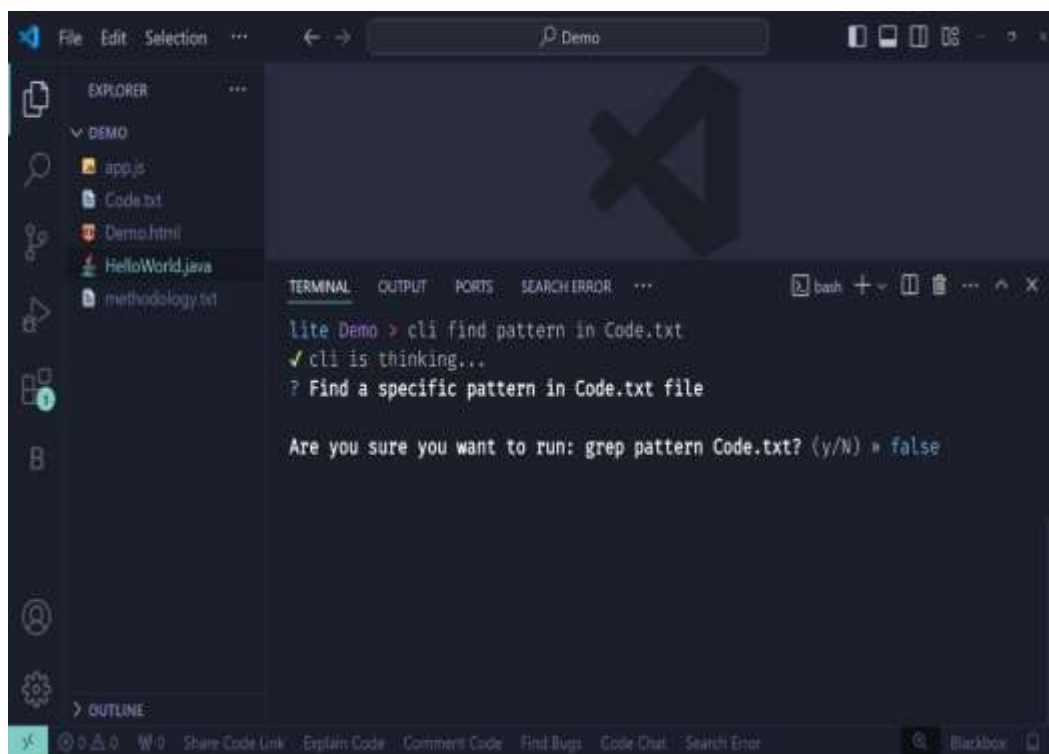


Figure 5.6 Output for Finding Pattern in File

The scenario you provided involves a user initiating a command-line interface (CLI) and requesting to find a specific pattern in a file named "Code.txt." In response, the CLI suggests running the command "grep pattern Code.txt" to search for the pattern.

However, the CLI presents a confirmation prompt asking, "Are you sure you want to run: grep pattern Code.txt? (y/N)" Here, the user's input, "false," indicates that they do not want to proceed with running the suggested command.

To explain in detail, "grep" is a command-line utility for searching plain-text data sets for lines that match a regular expression. In this context, it's being used to search for the specified pattern in the "Code.txt" file. The pattern to be searched for is denoted by "pattern" in the command.

The prompt "Are you sure you want to run..." is a safety measure to prevent unintended or potentially harmful commands from executing without the user's explicit consent. The user's response of "false" indicates that they do not want to proceed with running the suggested command, perhaps because they realized it may not yield the desired results or because they want to refine the search criteria before executing the command.

Overall, this interaction demonstrates the importance of cautious decision-making and user confirmation when executing commands in a command-line interface, especially when dealing with potentially sensitive operations like file manipulation or searching.

## Create HTML File :-

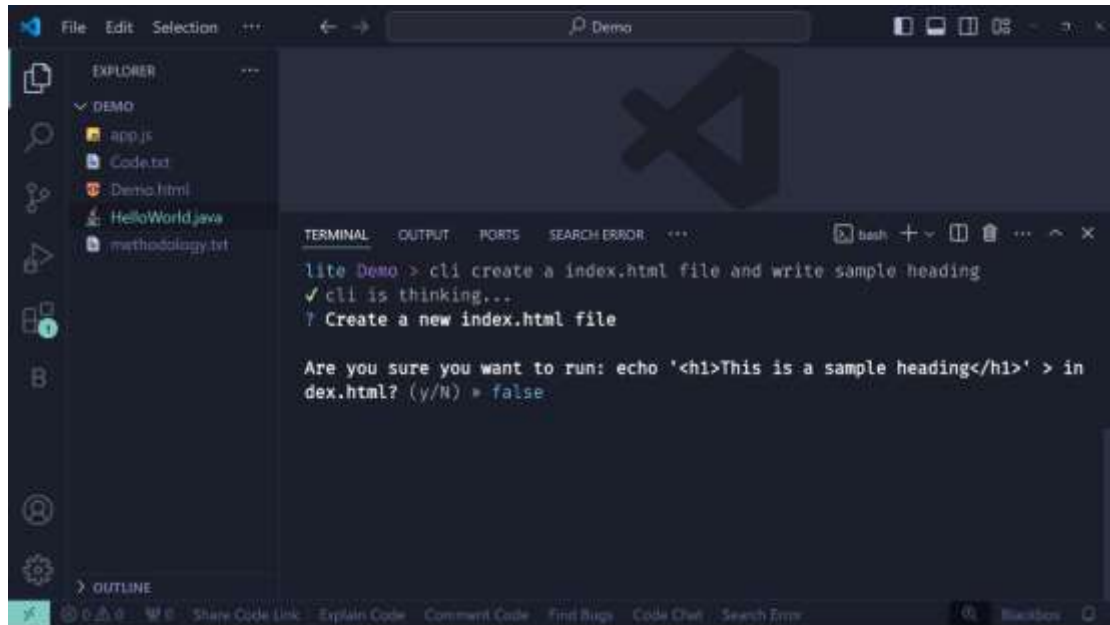


Figure 5.7 Creating HTML File

In this scenario, the user is interacting with a command-line interface (CLI) and requesting to create a new HTML file named "index.html" and write a sample heading within it.

The CLI suggests a command to accomplish this task: "echo '<h1>This is a sample heading</h1>' > index.html". This command uses the "echo" command to output the specified HTML content ("<h1>This is a sample heading</h1>") and redirects (">") that output to a file named "index.html", effectively creating the file and writing the HTML content into it.

However, before proceeding, the CLI presents a confirmation prompt: "Are you sure you want to run: echo '<h1>This is a sample heading</h1>' > index.html? (y/N)". Here, the user's input of "false" indicates that they do not want to proceed with executing the suggested command.



To explain in detail, the suggested command utilizes the "echo" command, a common command in Unix-like operating systems that prints the specified text to the standard output. In this case, it prints the HTML content ("`<h1>This is a sample heading</h1>`") to the standard output. The ">" symbol redirects this output to a file named "index.html", creating the file if it does not exist or overwriting it if it does.

The prompt asking for confirmation is a safety measure to prevent unintended actions. In this context, it ensures that the user confirms their intent to create the file and write the specified content into it before executing the command. The user's response of "false" indicates that they have decided not to proceed with executing the command, possibly due to realizing an error in the command or deciding to handle file creation differently.

Overall, this interaction illustrates the cautious approach taken by the CLI to ensure user consent and prevent accidental or undesired actions.

## Downloading Packages :-

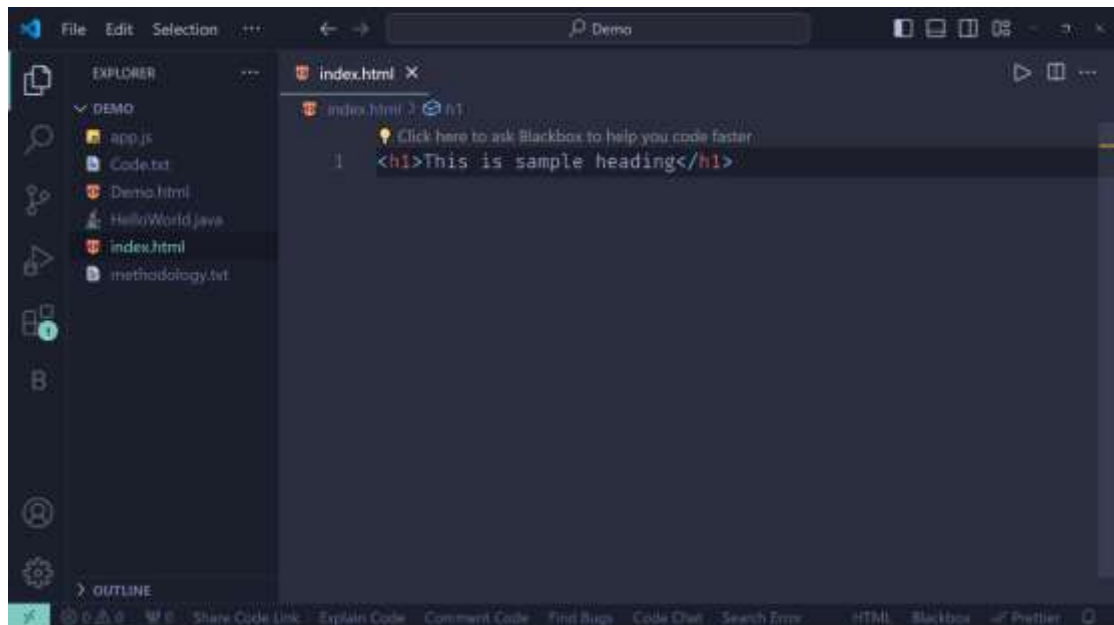


Figure 5.8 Input for Downloading Packages

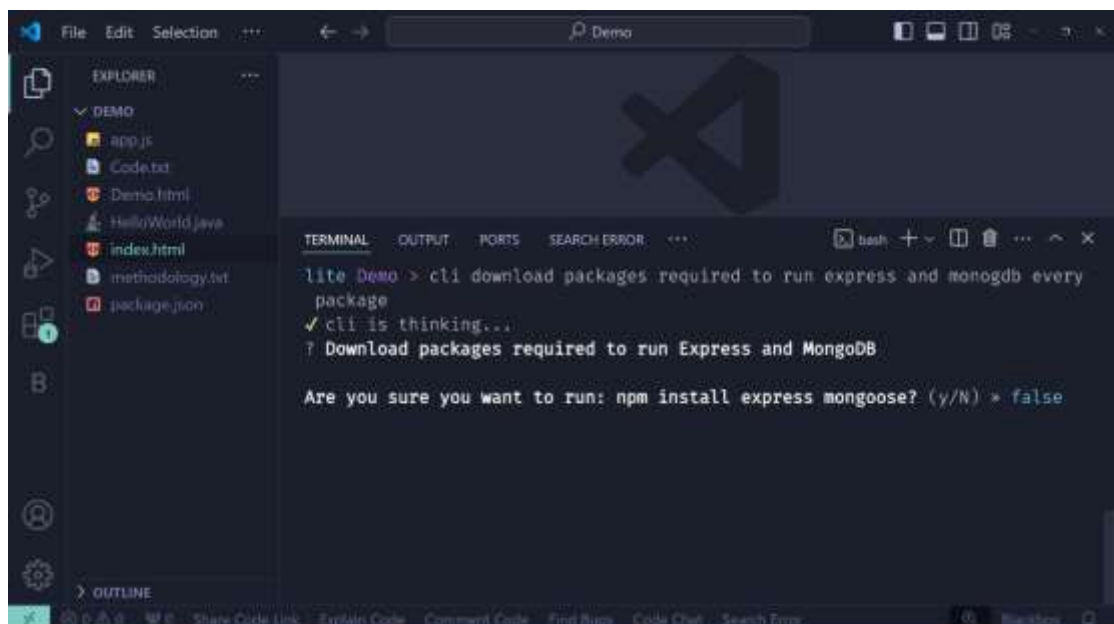


Figure 5.9 Output for Downloading Packages

In this scenario, the user is using a command-line interface (CLI) to request the download of packages required to run Express (a web application framework for Node.js) and MongoDB (a NoSQL database). The CLI suggests a command to accomplish this task: "npm install express mongoose".

However, before proceeding, the CLI presents a confirmation prompt: "Are you sure you want to run: npm install express mongoose? (y/N)". Here, the user's input of "false" indicates that they do not want to proceed with executing the suggested command.

To explain in detail, "npm install" is a command used with the Node Package Manager (npm) to install packages in a Node.js project. In this case, the command is being used to install two packages: "express" and "mongoose". "Express" is a web application framework for Node.js, while "mongoose" is an Object Data Modeling (ODM) library for MongoDB and Node.js, designed to work with MongoDB to simplify interactions with the database.

The prompt asking for confirmation is a precautionary measure to ensure that the user confirms their intent to install the specified packages before proceeding. This is important as package installation can modify the project's dependencies and may have implications for the project's functionality.

The user's response of "false" indicates that they have decided not to proceed with executing the command, possibly because they realized they don't need those packages, or they want to review the command before executing it.

Overall, this interaction demonstrates the CLI's cautious approach to user actions, ensuring that potentially impactful commands are executed only with explicit user consent.

# **CHAPTER 06**

## **CONCLUSION**

## **6.CONCLUSION**

In conclusion, the integration of generative Artificial Intelligence (AI) into Command Line Interface (CLI) command generation marks a significant advancement in computing. This fusion of technologies offers unparalleled potential to simplify and streamline command-line interactions, making computing more accessible and efficient for users across diverse domains. By leveraging machine learning algorithms, AI-driven CLI command generation automates the process of command formulation, reducing the cognitive burden on users and empowering them to focus on their tasks rather than syntax intricacies.

However, while the benefits are evident, the implementation of AI in CLI environments also presents challenges that must be addressed. Usability, accuracy, and security are paramount concerns, requiring robust validation mechanisms and safeguards to ensure the reliability and safety of generated commands. Additionally, efforts to enhance the interpretability and explainability of AI-generated outputs are crucial for fostering user trust and understanding.

Looking ahead, the convergence of CLI and generative AI holds immense promise for reshaping human-computer interaction. As technology continues to evolve, AI-driven CLI tools have the potential to democratize access to computing resources, accelerate innovation, and facilitate seamless interactions between users and systems. By embracing this transformative technology and addressing its challenges, we can unlock new possibilities and propel computing into a more accessible, efficient, and user-friendly future.

# **CHAPTER 07**

## **FUTURE SCOPE**

## 7.FUTURE SCOPE

**Enhanced Usability and Accessibility:** As generative AI continues to evolve, there is a significant opportunity to further improve the usability and accessibility of CLI environments. Future research could focus on developing AI-driven CLI interfaces that adapt to users' preferences and learning styles, making command-line interactions more intuitive and user-friendly for individuals of all skill levels.

**Advanced Natural Language Understanding:** Advancements in natural language understanding (NLU) techniques can enhance the capabilities of generative AI models to accurately interpret user input and generate relevant CLI commands. Future studies could explore the integration of state-of-the-art NLU algorithms with generative AI to improve the accuracy and responsiveness of AI-driven CLI command generation systems.

**Domain-Specific Command Generation:** Tailoring generative AI models to specific domains or industries could unlock new opportunities for CLI command generation. Future research could focus on training specialized AI models for domains such as software development, system administration, data analysis, and DevOps, enabling more contextually relevant and accurate command generation in these areas.

## **REFERENCES**



## REFERENCES

- [1] Partha Pratim Ray "ChatGPT: A comprehensive review on background, applications, key challenges, bias, ethics, limitations and future scope" Published by Elsevier B.V. on behalf of KeAi Communications Co., Ltd.7 April 2023
- [2] Dinesh Kalla "Study and Analysis of Chat GPT and its Impact on Different Fields of Study"International Journal of Innovative Science and Research Technology. 26 March 2023.
- [3] Devadas Menon , K Shilpa "Chatting with ChatGPT: Analyzing the factors influencing users intention to Use the Open AI's ChatGPT using the UTAUT model " Published by Elsevier Ltd. 12 October 2023.
- [4] Tom Fellmann , Manolya Kavakli "A Command Line Interface Versus A Graphical User Interface In Coding Vr Systems" ResearchGate 2015.
- [5] Volker Bilgram , Elix Laarmann "Accelerating Innovation With Generative AI: AI-Augmented Digital Prototyping and Innovation Methods" IEEE Engineering Management review, vol. 51, no. 2, second quarter , June 2023
- [6] V. Gaur and N. Saunshi, "Symbolic math reasoning with language models," in *Proc. IEEE MIT Undergraduate Res. Technol. Conf. (URTC)*, Sep. 2022, pp. 1–5.
- [7] K. Chen, T. Zhao, M. Yang, L. Liu, A. Tamura, R. Wang, M. Utiyama, and E. Sumita, "A neural approach to source dependence based context model for statistical machine translation," *IEEE/ACM Trans. Audio, Speech, Language Process.*, vol. 26, no. 2, pp. 266–280, Feb. 2018.
- [8] A. Mishra, A. Anand, and P. Guha, "Dual attention and question categorization-based visual question answering," *IEEE Trans. Artif. Intell.*, vol. 4, no. 1, pp. 81–91, Feb. 2023.

- [9] M. Yang, C. Li, Y. Shen, Q. Wu, Z. Zhao, and X. Chen, “Hierarchical human-like deep neural networks for abstractive text summarization,” *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 32, no. 6, pp. 2744–2757, Jun. 2021.
- [10] S. Kusal, S. Patil, J. Choudrie, K. Kotecha, S. Mishra, and A. Abraham, “AI-based conversational agents: A scoping review from technologies to future directions,” *IEEE Access*, vol. 10, pp. 92337–92356, 2022.
- [11] A. Sordoni, M. Galley, M. Auli, C. Brockett, Y. Ji, M. Mitchell, J.-Y. Nie, J. Gao, and B. Dolan, “A neural network approach to context-sensitive generation of conversational responses,” 2015, *arXiv:1506.06714*.
- [12] G. Bansal, V. Chamola, P. Narang, S. Kumar, and S. Raman, “Deep3DSCan: Deep residual network and morphological descriptor based framework for lung cancer classification and 3D segmentation,” *IET Image Process.*, vol. 14, no. 7, pp. 1240–1247, May 2020.

# **DISSEMINATION OF WORK**

## PUBLICATION DETAILS

PAPER TITLE	CONFERENCE NAME	CONFERENCE DURATION	ISSN NUMBER
Enhancing Command Line Interface (CLI) Usability through Generative AI: Current Trends and Future Direction	International Journal Of Advanced Research in Computer and Communication Engineering	4 April 2024	2278-1021









## PROJECT GROUP MEMBERS

Name:- Atharva Tattu  
Address:- Digras, Yavatmal  
Email id:- [atharva4tattu2002@gmail.com](mailto:atharva4tattu2002@gmail.com)  
Mobile No:- 7517469932



Name:- Prajwal Chitode  
Address:- Shegaon  
Email Id:- [prajwalchitode2002@gmail.com](mailto:prajwalchitode2002@gmail.com)  
Mobile No:- 9370409780



Name:- Rushikesh Dhawne  
Address:- Yavatmal  
Email Id:- [rushikeshdhawane64691@gmail.com](mailto:rushikeshdhawane64691@gmail.com)  
Mobile No:- 7666254614



Name:- Vedant Chaudhari  
Address:- Amravati  
Email Id:- [ch.vedant0745@gmail.com](mailto:ch.vedant0745@gmail.com)  
Mobile No:- 7620391783



# **PLAGIARISM REPORT**



# PLAGIARISM REPORT

final pdf.pdf

## ORIGINALITY REPORT

13%

SIMILARITY INDEX

10%

INTERNET SOURCES

8%

PUBLICATIONS

8%

STUDENT PAPERS

## PRIMARY SOURCES

1	Submitted to NIT Imphal Student Paper	3%
2	www.coursehero.com Internet Source	1%
3	Submitted to University of Warwick Student Paper	1%
4	vdocuments.mx Internet Source	1%
5	www.ijert.org Internet Source	1%
6	researcher.manipal.edu Internet Source	<1%
7	Volker Bilgram, Felix Laarmann. "Accelerating Innovation With Generative AI: AI-Augmented Digital Prototyping and Innovation Methods", IEEE Engineering Management Review, 2023 Publication	<1%