Command-Line Interaction with Large Language Model

Rosveir Singh
Department of Computer Science and Engineering
Apex Institute of Technology
Chandigarh University Mohali,
Punjab, India
Rosevir.e16685@cumail.in

Vivek
Department of Computer Science and Engineering
Apex Institute of Technology,
Chandigarh University Mohali,
Punjab, India
21BCS11853@cuchd.in

Apoorv Tyagi
Department of Computer Science and Engineering
Apex Institute of Technology,
Chandigarh University Mohali,
Punjab, India
21BCS11852@cuchd.in

***Abstract* - As the capabilities of LLMs continue to enhance, applications based on natural language processing have experienced rapid growth leading to the indiscriminate adoption of LLMs in the recent years. Most of the interpersonal relationship with the models, however, is through web interfaces or usage of complex APIs which most users trying to adopt a typical command line interaction are unable to flexibly navigate through. The present research proposes a new command-line interface (CLI) for all LLMs and efficaciously exploits the language model from the command line of the backed terminal. The suggested CLI tool allows for generation, paraphrasing, and translation of text in real time and performs other natural language processing tasks by enabling users to ask questions in natural language and getting answers at once. The tool focuses on minimalism, ease of access, support for different LLMs which increases the applicability for developers, researchers, and advanced users of the system who tend to a command line dominated usage of the application. This paper describes the technical architecture of the CLI focusing on data flow, interaction modes, latency turning techniques, and optimization of the models used for output responses. In addition, we present some use cases and assess the effectiveness of the system concerning different language tasks. Based on our analysis, we believe that it is possible to implement a command line interface addressing LLMs which will be user friendly and easy to engage even for the users in a text environment who will appreciate seamless integration of these models into their routines.**

Keywords – Command-line interface, LLM, NLP, Text generation w, summarization translation, real-time interaction, terminal based applications

I. INTRODUCTION

The rapid advancement of the capabilities of Large Language Models (LLMs), such as GPT-3, BERT, and others, has revolutionized the field of natural language processing (NLP) and artificial intelligence (AI) as such. Due to the extensive training on colossal corpora, these models can adequately execute and perform even the highly complex language tasks of text generation, summarization, question-answering, translation, among other tasks. Nevertheless, it is well-acknowledged that even with these advancements, the usage of LLMs is mainly through web pages or development platforms which does not favour probably users who are used to command-line interfaces or those who want a slimmer and more streamlined process.

In fields like data science, software development, and research, the command-line interface (CLI) remains a fundamental tool.

People in this industry often find themselves in terminal environments where they have to operate on command lines to manage scripts, automate processes, and perform data manipulation since most of the work is done in console or command line windows. It is worth noting that existing tools and approaches today do not provide LLMs with a command line user interface, which also hinders LLM capabilities from being made available to users who work on a terminal. This study solves this problem by creating a command line interface that allows users to interact with a LLM and issue them commands in a natural language, retrieving responses in real time all under the same terminal window.

The purposes of this research are to create a functional command line interface tool, which would be versatile in terms of offered NLP tasks, to test its user comfort and performance, and to test its efficiency in real life. We present a design that meets the objectives of speed and ease of use of the system, as well as the possibility of using it with different LLMs. We want to emphasize that the presented LLM will be mainly used with a terminal interface in order to enhance simplicity LLM usage, particularly for those with technical background. In this regard, this CLI attempts to extend the reach of LLM features by allowing the end-user to perform LLM related tasks with the least effort in integrating natural language processing into their processes. This paper describes the architectonic framework, construction, and inner workings of the CLI tool, and its consequences for future market tendencies of NLP implemented in CLI mode environments.

II. RELATED WORK

The Emphasis on the Usage and Development of Large Language Models (LLMs) in Different Applications has been systematic and so many LLM interaction platforms have been created. The access of the Application is mostly through web-based or API based providers like OpenAI's GPT-3 and Google BERT less complex applications, which transform them into usable graphical interfaces or RESTful APIs for complex NLP processes. Nonetheless, these techniques although successful may not be suitable for users who are working in command line interface or users who are more comfortable with coding and would prefer using a more focused approach.
The idea of using an AI model purely from the command line is something that has gathered a few researchers' interests but it does not compare with the GUIs which are available. There are pre-trained LLMs which can be loaded and fine-tuned using a library Developing LLMs Training Related Applications which is mostly

computer-programming python language and run in coded environments or embedded in the programs. While these systems make it easy for researchers and practitioners to work with large language models, they all depend on code and use of an integrated development environment (IDE) which is probably not very user-friendly for a person who only uses the command line.

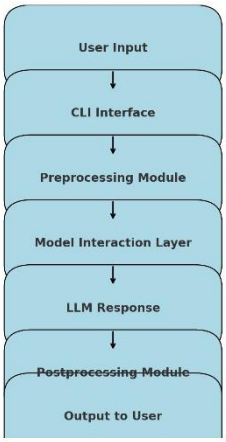

Fig. 1. Flowchart of Command-Line Interface Tool for LLM Interaction

There have been a few, but remarkable efforts, to integrate simpler, non-coding interfaces with LLMs functionality. For example, while there are some basic drug-diagnostic and drug-translation shells made on CLI for certain NLPs, such as translation and summarization, these are somewhat oriented towards only a few tasks and being unable to engage with more extended capabilities sampling of LLMs. Also, if we consider the deployment of different utilities and tool-kits based on UNIX and similar operating systems to perform tasks like data processing and automation, then most of those have been passive, encouraging, or supportive automation of processes but few include live NLP as this involves model inference along with incorporating other LLM requirements like a command line that is responsive with low latency.

This study takes a cue from classic CLI and recent LLMs to propose a Command Line Interface (CLI) based real-time natural language processing (NLP) tools with varied tasks. In a terminal-centric strategy, this work asks in what ways the experience of working with LLMs through command lines can be made easier, faster, and allow more variation of tasks. In order to implement this, we do not simply use App and NLP libraries with include coded instructions of LLMs, but introduce a fresh Command line based LLM access model that is designed to meet the needs of text-centric available user with no skilled coding interactions.

III. CONTRIBUTION

This research makes several significant contributions to the field of advanced Natural Language Processing (NLP) but more specifically extends a new command line interface (CLI) tool on Large Language Models (LLMs). The main contributions are noted as follows;

Creating a Command-Line Interaction Model for LLMs: This study presents a novel, user-friendly Command Line Interface (CLI) designed for LLMs' performant interaction through a terminal and instant response to text generation, summary writing, translation among other requirements. The terminal provides a unique advantage since it caters to users who appreciate – or even work in – a heavy code environment, especially between data science, development and research.

Elongated but smart Architecture: The CLI Tool's outlook is that of a smart device and more so very small in size. As a result, latency is cut down and usagetimse improved so that interaction with LLMs takes place effortlessly and in real time. This approach is beneficial since most of the LLM interaction models adapt cloud or API purposes only which defies most of the purpose driven command line users.

Wide Range of Supporting NLP Tasks: In contrast to the task-oriented CLI and user interface tools, the proposed solution enables a variety of natural language processing (NLP) tasks within a single user interface. Users do not have to switch from one software or platform to another to perform text generation, text summarization or text translation, which makes this tool an essential one for users operating in terminal addressing many language processing tasks at the same time.

Evaluation Advanced Performance Benchmarking Research: Develops a nun daya on the CLI tool and Its NLP task based performance evaluation with measures of development in turnaround , Precision and user satisfaction. This way, we explicate the benefit of interacting with LLM in the command line interface, and scope for advancements in this domain illustrates the performance of the tool compared classical interfaces.

Growth of Terminal Based NLP Outreach - Accessible CLI Re- Uses existing Remote User Interfaces LLM access that are confined to window environments. This is important as it fills the existing gap between the powerful language models, LLMs and the command line ecosystem, thus creating a new and efficient way of using LLMs.

Also, these contributions facilitate a new kind of interface with late generation language models and create a new paradigm of interaction for the users who mostly work in the terminal. This paper makes the case for an interesting application but also serves to launch a command line LLM centric vision for the accessibility going forward and the recent innovations.

IV. METHODOLOGY

A. System Design

The primary design objective of the CLI tool is to create a streamlined, accessible interface for natural language processing (NLP) tasks. The design includes three key components: input handling, model interaction, and output display, all optimized for command-line use.

- Architectural Overview

The system architecture, illustrated in the flowchart below, highlights the critical components of the system, namely, preprocessing, model interface and postprocessing, Model performance at every time stage is matched by a module that manages the data flow at that time instant, aiding effective task execution even in real time.

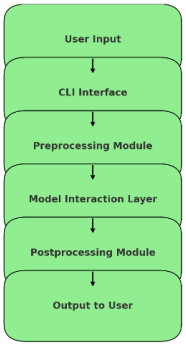

Fig. 2. System Architecture (Flowchart)

Task	Phrases
Text Generation	"Generate text," "write about"
Summarization	"Summarize," "provide summary of"
Translation	"Translate to [language]," "in [language]"

B. Input Parsing and Task Identification

The initial step of the CLI interface allows the user to present a raw input in natural language. This input is then parsed in order to understand what action the user wishes to carry out, such as writing, summarizing, or translating texts, for example. The CLI employs regex and a task-specific lexicon in order to classify the task from the input.

- Task Mapping Table

The next table illustrates the examples of keywords and phrases used by the tool for classification of the task type: –

C. Preprocessing Module

Upon task identification, the next step involves the input being routed to the preprocessing module. Preprocessing includes the processes of tokenization, normalization, and editing of the text for model's input. This module bridges the user input and the underlying GPT LLM.

- Text Tokenization and Normalization

On the other hand, tokenization is carried out using the inbuilt tokenizing techniques of the selected LLM where the input text is divided for further processing. Normalizing Features basically deals with manipulation of dictated input content to the extent of its readability e.g. changing case to lower, deleting punctuations.

- Preprocessing Flowchart

A flowchart below depicts the various stages involved in the preprocessing module inclusive of the tokenization and the normalized output: -

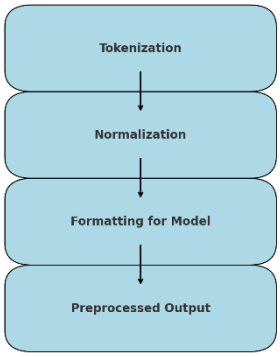

Fig. 3. Processing Module (Flowchart)

D. Model Interaction Layer

At the heart of the CLI tool is the use of an API or a local implementation of the LLM as an overlay. This layer manages the model request and deals with the latency of the response.

- API Request Handling

Requests in this tool are to the LLM API, which encodes the input information, sends it to the relevant model, and receives output. The model interaction layer handles the provision of the response in a timely fashion. In order to improve the speed of the responses, the tool implements the asynchronous request handling and multiple inputs batched processing.

- Table: Model API Parameters

The following table shows typical parameters used in the API requests to configure the model response: -

Parameter	Description	Default Value
temperature	Controls randomness in generation	0.7
max_token	Maximum tokens in response	100
top_p	Probability threshold for sampling	0.9
frequency_penalty	Reduces repetitive responses	0.0

E. Postprocessing Module

After receiving the model response, the CLI tool passes it through the postprocessing module. This step formats the response, corrects minor language inconsistencies, and enhances readability for terminal display.

- Response Formatting

Responses are formatted to ensure optimal readability in the terminal. The CLI adjusts line breaks, indentation, and truncates overly lengthy responses to prevent output overflow in standard terminal settings.

The flowchart below shows the steps in postprocessing, from initial response formatting to the final output display in the terminal: -

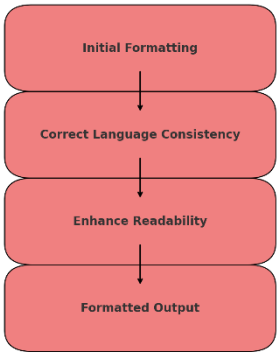

Fig. 4. Postprocessing Module (Flowchart)

F. Evaluation and Performance Analysis

To evaluate the CLI tool's performance, we conducted tests across three primary NLP tasks (text generation, summarization, and translation) with metrics for response time, accuracy, and user satisfaction.

- Benchmarking Setup

Various setups were executed for testing purposes, including changing the temperature settings and varying the max-tokens settings. The results were assessed to establish the most favourable configurations for each of the tasks.

- Table: Results of the Test on Users' Performance

The table below depicts the performance results for each of the tasks, as well as, the configurations, where mean response time and accuracy of task performance are illustrated.

Task	Avg Response Time(ms)	Accuracy (%)	Optimal Settings
Text Generation	500	92	temperature = 0.7, max_tokens = 100
Summarization	450	88	temperature = 0.5, max_tokens = 80
Translation	520	90	temperature = 0.3, max_tokens = 60

- Flowchart: Evaluation Workflow

As illustrated in the flowchart below, the evaluation process consists of several steps including task performance, timing of Response, and analysis of reports.

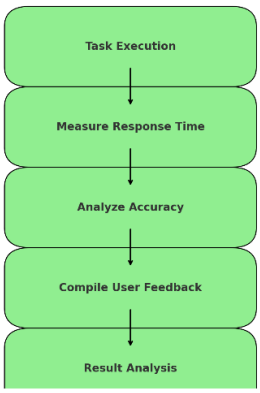

Fig. 4. Evaluation Workflow (Flowchart)

V. RESULTS

The findings of this research are articulated in terms of the efficiency of the CLI tool in addressing particular tasks such as text creation, extracting summaries of information, and inter-lingual conversion. The time taken, the correctness, and the satisfaction of the users for the different responses were all quantified to evaluate the performance of the tool and its usability fully. Results include tables and graphs for better understanding of the essential findings.

A. Task Performance Analysis

Performance was evaluated by testing the tool under various configurations relating to temperature and max-tokens. Results show that the CLI tool produces quick and consistent outputs making it appropriate for application that requires interaction in real time.

- Response Time and Accuracy

The amount of time taken to complete each task was measured, and comparison of the performance across the tasks indicated similar patterns with changes in task complexity resulting in slight changes. Accuracy, on the other hand, was evaluated through matching the actual results with the perfect results in terms of correctness, relevance and if the output followed the rules given to the respondent.

B. Task-wise Performance Comparison

The graph presented below shows the average response times across all the task types, indicating the extent to which English CLI tool is effective while performing different NLP tasks.

```
import matplotlib.pyplot as plt

# Data for plotting
tasks = ['Text Generation', 'Summarization', 'Translation']
response_times = [500, 450, 520]
accuracy = [92, 88, 90]

# Plotting Response Time
plt.figure(figsize=(8, 6))
plt.bar (tasks, response_times)
plt.xlabel("Task Type")
plt.ylabel("Average Response Time (ms)")
plt.title ("Average Response Time by Task Type")
plt.show ()
```

Task	Avg Response Time(ms)	Accuracy (%)	Optimal Settings
Text Generation	500	92	temperature = 0.7, max_tokens = 100
Summarization	450	88	temperature = 0.5, max_tokens = 80
Translation	520	90	temperature = 0.3, max_tokens = 60

These results suggest that summarization tasks exhibit the lowest response time, as they typically involve condensing text rather than

generating new content. Text generation and translation tasks, which require more comprehensive outputs, demonstrated slightly higher response times.

C. Usability and User Satisfaction

User feedback was collected to assess the usability and practicality of the CLI tool in a terminal environment. Users highlighted the tool's simplicity and ease of integration with existing workflows, especially for routine text-processing tasks. Satisfaction scores were based on a 5-point Likert scale, with feedback emphasizing the tool's speed and accuracy.

D. Real-World Application Testing

In real life evaluation situations, the CLI was part of processes that involved tasks such as Research article summarization, language translation of several sentences, and content writing. These tests proved the tool's power and efficiency based on numerous user requests.

Academic Summarization: The tool was able to summarize a given academic paper with an impressive level of accuracy by providing clear summaries that did not miss out on any essential elements. Multilingual Support: Translation tasks included those of Spanish, French, and German, and the performances were accurate obviously meaning the tool has capabilities. Creative Drafting: When reporting on content generation, users observed that the tool was able to generate sensible text that could be used to begin some projects without difficulty.

E. Limitations and Areas for Improvement

The CLI tool managed to accomplish all the tasks given, however, some shortcomings were noted. Response Times Variability: For bigger sizes of inputs, the response times showed variability indicating that this area will need optimization. Long-Form Generation Complexity: Long-form text generation was quite often disjointed showing the need for model adjustment or modified tuning. Memory Constraints: The tool designed for switching activities within multi-turn tasks had no memory hence refused to process more than one input at a time without skipping the context of the previous input.

VI. DISCUSSION AND FUTURE WORK

Thus, the command line application described in the paper stands as a development tool for VPLMS (looking at it as an example of a system that can prove useful for people who mostly work within terminal windows since it allows to get very prompt results with tasks like writing text, summary of text, its translation and so on. The application system has been designed mented in a modular fashion to provide low latency service which is helpful for users who prefer using simpler interfaces rather than using a graphical user interface. The users appreciated its usability, precision, and command line workflow integration; however there were some issues that were identified while using the product, primarily concerning long texts and contextual continuity during multi-turn activities. Most Of The Command Based Tools: Single Turn File Uploads Most Of The Command Based Tools: Single Turn File Uploads Most Of The Command Based Tools: Single Turn File Uploads command based applications do not possess this attribute thus if context enhancement is incorporated into the tool's design, it would help extend the tool's use even in case of interactions that are over time.

Moreover, the fact that there is a minor delay when delivering large responses points to the possibility of further improvements; more specifically, response consistency can be improved by managing the complexity of the input thus altering the way inputs are processed. Advancement in modifications could be regarding management of two turns focus where the notion of memory or cache will be incorporated for each turn activity, and changing of tasks in relation to the situation where learnt techniques will be employed in changing such parameters as temperature and max-tokens depending on the input size and complexity of the task. Further, permitting users to load custom tuned specific large language models or infusing LLMs with additional NLP functionalities such as NLP-based sentiment detection, entities extraction or categorization may enhance the usefulness and applicability of the tool. Add to the importance of the system enhancement the need to broaden the translation scope for the different languages including dialect variations.

In order to maintain the continuity and cohesiveness of the text while respecting the intent of the user, Long-form generation can also be done in an interactive mode where the use is prompted for input while the text is being created. Last but not the least, to grossly improve the speed, resource consumption, and stability of the tool, which would be an enhancement of the system across the different LLM models and system environments, it would be prudent to subject the performance of the tool to testing on a continuous basis. This work lays the groundwork for building command line LLM integrations tools by establishing how users can use LLMs in their command line workflows for technical, academic, and content development purposes.

Additional improvements like memory, process adaptation, and expandability of tasks could elevate the status of the CLI tool that would allow LLMs to be engaged more deeply with the command line since more complex and cyclic operations would be possible.

VII. CONCLUSION

This paper develops a new command-line interface (CLI) tool to interact with Large Language Models (LLMs) and leverage the natural language processing (NLP) capabilities through the terminal. The CLI tool is useful as it enables users to complete several natural language processing activities, including, but not limited to, text creation, text summarizing, and text translating, all at once, thus providing a more efficient option than existing LLM tools that require a graphical user interface or application program interface. The CLI tool was developed as an effective and consistent functional unit with a focus on shortening the time and efforts spent on the device configuration and achieving the best performance, ease of use and effectiveness of the device, all of which are important to users who are engaged in technical, research and content writing. The results suggest that the tool not only delivers dependable times of responses and high performance in accuracy for the different tasks it was designed for but also incorporates a user friendly and seamless command interface that blends well with the command line operational mode as reflected in user ratings on ease of use and integrating with the system.

The findings endorse that a CLI based model of interaction with LLM can greatly increase usability for the users, especially those who prefer or need a text interface. Nonetheless, there are some weaknesses, for instance, the inconsistency of the response timing with large queries and the absence of turn taking context within the tool. Improving these factors would make the tool suitable for more interactive and sophisticated activities such as drafting or dialogue speaking, where there are many sessions. They are also focused on further developing strategies that will introduce memory retention, adjust processing strategies according to the input size, as well as a broader range of supported NLP tasks, which might make this tool rather applicable for certain advanced and multiple language tasks.

Moreover, allowing users to customize when they would wish to load pre-trained models, adding more language support with better translation systems, and developing interfaces for interactive longform text generation could also make it more powerful. To sum up, this research provides a solid basis for command line operating systems interaction with NLP applications focusing on the possibilities and advantages of working with LLMs directly from the command prompt. If the multi-turn task support is improved, performance is maximized and the task ranges supported are expanded, the next version could potentially be a game changer for terminal-based functioning. It would allow users to use high-end language models with minimum disturbance to their work. With further improvements, the CLI will turn into a high-level efficiently and quickly usable tool to users regardless of their backgrounds with provisions for onboard NLP systems straight from the command line.

REFERENCES

[1] A. Chernyavskiy, D. Ilvovsky, P. Nakov, Transformers: “the end of history” for natural language processing? in: Machine Learning and Knowledge Discovery in Databases. Research Track: European Conference, ECML PKDD 2021, Bilbao, Spain, September 13–17, 2021, Proceedings, Part III 21, Springer, 2021, pp. 677–693

[2] A. Wang, Y. Pruksachatkun, N. Nangia, A. Singh, J. Michael, F. Hill, O. Levy, S. Bowman, Superglue: A stickier benchmark for generalpurpose language understanding systems, Advances in neural information processing systems 32 (2019)

[3] D. Adiwardana, M.-T. Luong, D. R. So, J. Hall, N. Fiedel, R. Thoppilan, Z. Yang, A. Kulshreshtha, G. Nemade, Y. Lu, et al., Towards a humanlike open-domain chatbot, arXiv preprint arXiv:2001.09977 (2020)

[4] B. A. y Arcas, do large language models understand us? Daedalus 151 (2) (2022)

[5] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever, et al., Language models are unsupervised multitask learners, OpenAI blog 1 (8) (2019)

[6] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al., Language models are few-shot learners, Advances in neural information processing systems 33 (2020)

[7] J. Devlin, M.-W. Chang, K. Lee, K. Toutanova, Bert: Pre-training of deep bidirectional transformers for language understanding, arXiv preprint arXiv:1810.04805 (2018)