Comparative Analysis of Large Language Models for the Machine-Assisted Resolution of User Intentions

Justus Flerlage[1,*], Alexander Acker[2] and Odej Kao[1]

[1]*Distributed and Operating Systems Group*
Technische Universität Berlin
Berlin, Germany
[2]*logsight.ai GmbH*
Berlin, Germany

Abstract

Large Language Models (LLMs) have emerged as transformative tools for natural language understanding and user intent resolution, enabling tasks such as translation, summarization, and, increasingly, the orchestration of complex workflows. This development signifies a paradigm shift from conventional, GUI-driven user interfaces toward intuitive, language-first interaction paradigms. Rather than manually navigating applications, users can articulate their objectives in natural language, enabling LLMs to orchestrate actions across multiple applications in a dynamic and contextual manner. However, extant implementations frequently rely on cloud-based proprietary models, which introduce limitations in terms of privacy, autonomy, and scalability. For language-first interaction to become a truly robust and trusted interface paradigm, local deployment is not merely a convenience; it is an imperative. This limitation underscores the importance of evaluating the feasibility of locally deployable, open-source, and open-access LLMs as foundational components for future intent-based operating systems. In this study, we examine the capabilities of several open-source and open-access models in facilitating user intention resolution through machine assistance. A comparative analysis is conducted against OpenAI's proprietary GPT-4-based systems to assess performance in generating workflows for various user intentions. The present study offers empirical insights into the practical viability, performance trade-offs, and potential of open LLMs as autonomous, locally operable components in next-generation operating systems. The results of this study inform the broader discussion on the decentralization and democratization of AI infrastructure and point toward a future where user-device interaction becomes more seamless, adaptive, and privacy-conscious through locally embedded intelligence.

1. Introduction

Contemporary LLMs possess the capability to comprehend natural language, discern user intent from input expressions, and execute tasks such as document summarization or translation [1], image generation [2] or code generation [3] tasks. Beyond these functions, LLMs also present the potential to deconstruct complex intents into discrete, actionable steps, thereby enabling the automated construction of workflows in a manner analogous to human reasoning [4]. LLMs have the potential to profoundly transform human-device interaction by supplanting rigid graphical interfaces with intuitive, conversational ones. Rather than navigating through menus or memorizing application-specific commands, users can articulate their objectives in natural language. LLMs are responsible for interpreting these inputs and orchestrating actions across various applications and services in a dynamic manner. As a consequence, complex tasks are simplified, and the system adapts to each user's habits and context, thereby personalizing the user experience. This shift is not only particularly salient in the context of mobile devices, where screen space and input methods are constrained, but in other applications for human-computer interaction as well, such as robotics, where robots mimic human-like

---

communication. Interfaces are undergoing a paradigm shift towards invisible, language-first systems, whereby interaction resembles conversing with a smart assistant more than utilising a conventional device.

For instance, current systems necessitate the manual coordination of multiple applications to reschedule an appointment. Despite the ostensible simplicity of the user-given intention "Reschedule my appointment for tonight," the process introduces a cumbersome and complicated workflow consisting of multiple steps. The user is required to manually open the calendar application and search for the appointment, locating the participants. The user is then prompted to access the contacts application to research the contact details of the relevant participants. This approach is adopted to facilitate effective communication via telephone or text message for the purpose of negotiating alternative dates. The process under discussion is characterized by its cumbersome and time-consuming nature, especially when incorporating multiple participants. Additionally, the user is required to devise a sequence of actions to operate the various applications, necessitating not only a fundamental understanding of the provided interfaces, but also the capacity to operate them successfully.

The prevailing design of operating systems has been predicated for the aforementioned interaction mechanisms with GUIs, hierarchical file management, and the shell, allocating the responsibility for interaction to the user. Therefore, the interaction mechanisms initiated by LLMs necessitate a reconceptualization of fundamental design decisions in contemporary operating systems. In our previous work, we presented the first step on the path to such a GUI-less operating system with the utilization of the proprietary `gpt-4o-mini` model [5]. Nevertheless, the proposed methodology engenders a considerable degree of interdependence. The advent of future mobile devices is poised to achieve user intentions independently of external infrastructure. It is imperative to incorporate LLMs into local devices to ensure autonomy, privacy, extensibility, and optimization. Open-source and open-access models hold considerable potential in this endeavor and can serve as pivotal elements, not only for the future integration of such LLMs on local devices, but for the development of future operating systems with open and transparent ecosystems. This leads to the research question that guides this study, which is as follows: "How effective are open-source and open-access models in resolving user intentions for future intent-based operating systems, and what areas of research and development are indicated to enable broad, multi-domain deployment?".

In this study, a comparative analysis of leading open-source and open-access models for this particular application domain is undertaken. We evaluate and analyze the performance of different LLMs for the purpose of generating different workflows for realizing a set of given user intentions. The comparison will include leading open-source and open-access models, such as Falcon 3, Phi 4, and Qwen 2, as well as proprietary models based on the fourth GPT generation from OpenAI, for comparison. We contribute our evaluation and analysis of the aforementioned LLMs, providing valuable insights regarding the feasibility of utilizing self-hosted, open-source, and open-access LLMs, as well as their comparative performance with proprietary models from OpenAI. The code for the experiments can be found in the Git repository at GitHub [1].

In the following section, Section 2, we present the methodologies and the approach of our study. This is followed by Section 3, which the experiments and the results are presented. Section 4 shows a discussion an interpretation of the results. The related work is shown afterwards in Section 5. Finally, the conclusion is presented in Section 6.

---

[1]https://github.com/dos-group/LLMWorkflowGenerator

2. Methodologies and Approach

The process of translating user intentions into actionable and executable workflows is of paramount importance for the development of future systems that prioritize intent-driven interaction mechanisms. Current LLMs have demonstrated the capacity to decompose user intentions into actionable steps, thereby enabling the design of workflows analogous to those employed by human users. The necessity of an intermediate representation is imperative for the description and modeling of these workflows and its necessary steps for resolving a given intention. This representation must possess the capacity to address arbitrary and complex user intentions.

The code generation capabilities of LLMs are leveraged to synthesize workflows tailored to specific user intentions. These workflows are conceptualized as deterministic state machines, that can be effectively modeled using imperative programming languages, as shown in Figure 1. The execution of such imperative programming language code is equivalent to state transitions of the state machine, which models the workflow. This refers to the ability to model both sequential steps and more complex control flow structures, such as loops and branches. Furthermore, it facilitates the interruption and preemption of steps and the management of asynchronous tasks, thereby enabling more flexible and dynamic program execution, and ultimately allowing for the incorporation of more complex user intentions. Within this framework, the LLM must not only interpret the user's high-level intent but also accurately comprehend and represent the underlying functionalities of the relevant application programming interface (API). This necessitates that the model parses the prompt with precision, analyzes the structure and semantics of the API, and subsequently generates syntactically correct and functionally coherent code that aligns with the intended behavior.

Figure 2 provides a concise synopsis of the system architecture, which is the framework under discussion. It consists of an operating system running a dedicated *Controller* application, which serves as the central coordination unit. The Controller is responsible for managing communication with an externally hosted LLM, acting as the interface between the local execution environment and the remote machine, which provides access to an LLM. In addition to overseeing interactions with LLM, the Controller orchestrates the scheduling, instantiation, and execution of workflows generated by the model. Subsequent to the synthesis of these workflows, they are integrated into the system's runtime environment in a seamless manner. This integration facilitates dynamic and adaptive responses to user intentions. The Controller's foundational element is the *Function Table*, which contains a catalog of available functions, accompanied by their precise specifications, including signatures and associated implementation callbacks. It plays a crucial role in the generation of documentation, which is essential for guiding the LLM in generating valid and executable code. The Function Table, in conjunction with a textual representation of the user's intention, is employed by the *Prompt Formatter* component to generate a prompt. This prompt is subsequently transmitted with a request to the *LLM Service*. The system processes the user's prompt and generates the corresponding code, contingent on the user's intention and the available functions provided by the Function Table. Next, the *Executor* employs the code generated by the LLM, subsequently executing it within a meticulously controlled environment. The execution of these functions is contingent upon the availability of the corresponding function implementations, which are stored in the Function Table. This measure is designed to prevent any unauthorized or unintended code execution and to establish an environment for executing the generated code in a controlled and isolated execution scope.

The collection of metrics is conducted in accordance with the experimental protocol, encompassing the Time to First Token, the Response Time, as well as the inclusion of preambles, postambles, and code comments for measuring, comparing, and objectively evaluating the model's performance and responsiveness. The Time to First Token metric is defined as the duration required to receive the initial output from the specified LLM. This figure illustrates the model's initialization and processing overhead prior to generation. On the other hand, the Response Time metric is defined as the total time
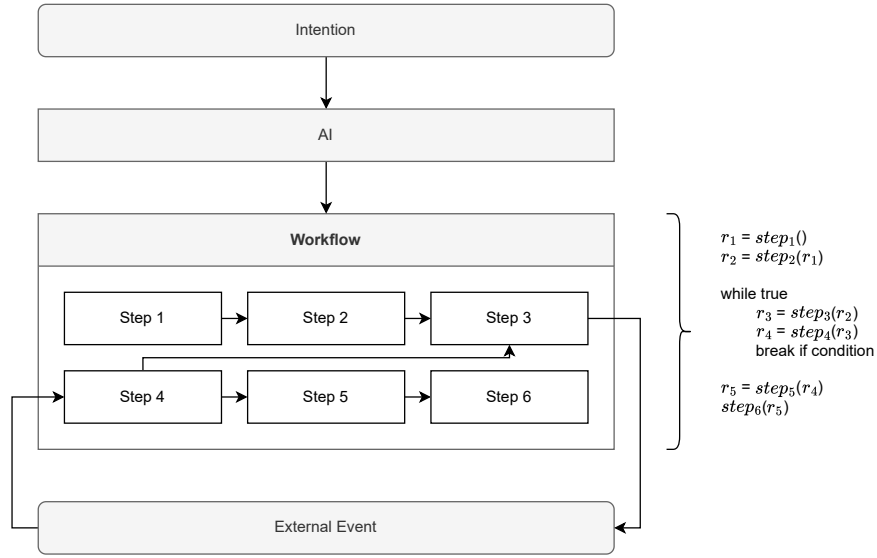
Figure 1: Workflows and state machines are analogous, and thus, they can be modeled using imperative programming languages.
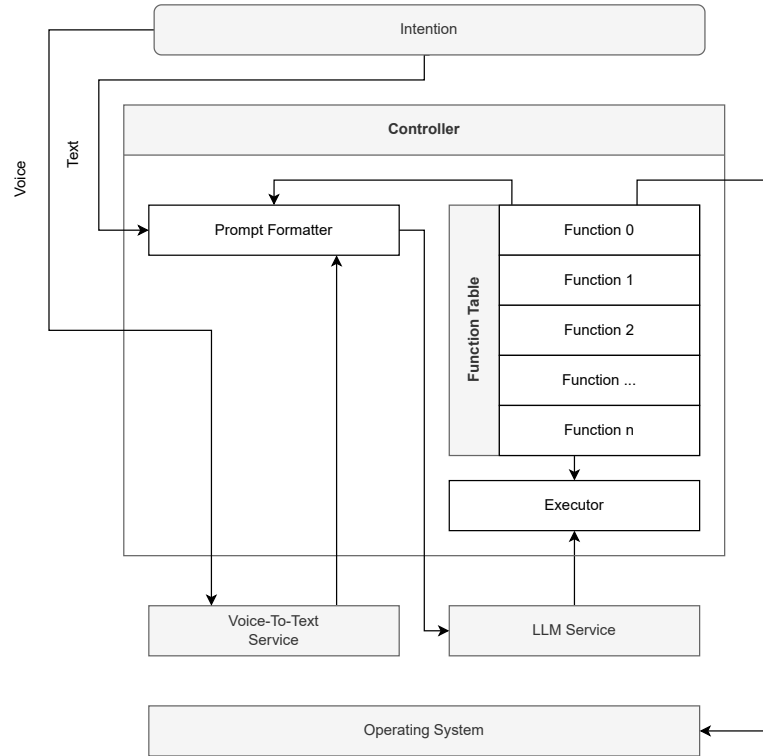

Figure 2: The deployed system architecture for our experiments.

required to receive the complete output, which should be measured in seconds and occur within a few seconds to avoid disrupting the user's thought process [6]. Code elements, including comments, offer valuable insights into the decision-making process. However, these elements do so by increasing the response size, albeit to an infinitesimal degree. Preambles and postambles are integrated into the response and envelop the generated code block. These consist of explanations or introductory words from the LLM. The aforementioned elements are considered superfluous and serve only to augment the response size, thereby demonstrating an understanding of the designated role.

```
function find_contact_id(expression: String): Integer|null
function find_contact_email(contact_id: Integer): String|null
function ask_question(question: String): String
function send_email(email: String, subject: String, text: String, attachment_paths: Collection<String>): void
function get_temperature(): Integer
function find_files(expression: String): Collection<String>
function print(text: String): void
function shell(command: String): String
function sleep(seconds: Integer): void
function find_all_audio_files(): Collection<String>
function generate_random_number(inclusiveStart: Integer, exclusiveEnd: Integer): Integer
function play_audio_file(file_path: String): void
function find_file(expression: String): String|null
function stop_audio_player(): void
function query_llm(query: String): String
function http_get_request(url: String, headers: Dictionary<String, String>): String
```

Figure 3: Functions accessible by the respective LLM.

3. Experiments and Results

The subsequent section delineates the experiments and the results for demonstrating the feasibility of utilizing open-source and open-access models in the aforementioned application domain. The system architecture presented in the preceding section, Section 2, is employed to investigate and provide a comparative analysis of disparate LLMs for the purpose of exploiting user intent resolution through machine code generation.

An implementation of the aforementioned Controller is facilitated by the Python 3 programming language. It is also employed as the base programming language for generating workflow-equivalent code due to its extensive adoption and the fact that LLMs are trained on publicly available data. Additionally, it enables the isolated and locally scoped execution of code through the exec function, without interfering with the global program structures of the Controller, and has the capacity to interface with the underlying execution process. This facilitates the generation of execution traces comprising function calls, their respective arguments, return values, and global context information. The generated code and the execution trace resulting from the execution of the generated code are used for evaluation. Through the integration of code blocks, the code embedded within the response of the LLM Service is decoded. The Function Table is populated with stub functions as well as real functions that implement real functionality. A complete list of the functions included in the function table is shown in Figure 3.

The Controller runs on a mobile device running the Android operating system. The Termux and Termux::API applications are used to access a shell, package manager and execution environment for running the Controller application, as well as to access to certain Android APIs via command line applications. The following open-source and open-access, as well as proprietary models are considered for the experiments:

- `falcon-3-10b-instruct`

- `qwen-2.5-14b-instruct`

- `phi-4`

- `gpt-4o`

- `gpt-4o-mini`

- `gpt-4-turbo`

```python
def play_audio_file(file_path: 'String') -> None:
    subprocess.run(
        ["termux-media-player", "play", path.join("files", file_path)],
        text=True,
        check=True,
    )

# ...

table.register(play_audio_file)
```

Figure 4: Implementation of the `play_audio_file` function, using the command line application `termux-media-player`, provided by Termux.

- `gpt-4.5-preview-2025-02-27`

Furthermore, the following user intentions, consisting of simple intentions such as smoke tests and knowledge-based as well as multi-action tasks, are taken into account:

1. *Please sleep for 5 seconds*

2. *Please tell me a random number between 1 and 100*

3. *Please tell me the current temperature*

4. *Play a random song in my list for 5 seconds*

5. *Which is the largest city in Germany?*

6. *Please tell me all files in the current directory*

7. *Please send my car title to my insurance company*

8. *Please summarize the Wikipedia article https://en.wikipedia.org/wiki/Transformer_(deep_learning_architecture)*

9. *Please install nginx on the machine with the address 127.0.0.1:2222 running Debian GNU/Linux*

The selected intentions encompass a wide spectrum of capabilities and scenarios. Simple baseline functions (4, 2, 5) ensure that fundamental responses function correctly. External information requests (3, 8) test connections to both dynamic and static knowledge sources. System-oriented tasks (6, 9) simulate realistic use cases in IT and development contexts. Media as well as everyday interactions (4, 7) address practical assistance functions, including security and privacy aspects. Collectively, these elements constitute a representative test set that encompasses a wide spectrum of cases, ranging from trivial to complex, security-critical, and highly practical scenarios. The Controller is configured to utilize each of the LLMs that have been presented, and is fed with each of the user intentions that have been previously outlined. The model temperature is set to 0.0 for more deterministic results and the role to *You are a Python 3 code generator* for ensuring the response consists of executable Python 3 code. The generated code and its execution traces, which are produced by the execution of the aforementioned code, are subsequently utilized for further evaluation. Each intention is transmitted to each LLM once. An exemplary resolution of Intention 4 employs the `falcon-3-10b-instruct` model. Figure 5 illustrates the invocation of the Controller and the subsequent resolution of the user intention to play a random song. Additionally, it presents the provided functions and the generated code.

Table 1 provides a comprehensive overview of each model, highlighting the user intention resolutions that have been successfully addressed and those that have not met expectations. A prevailing consensus
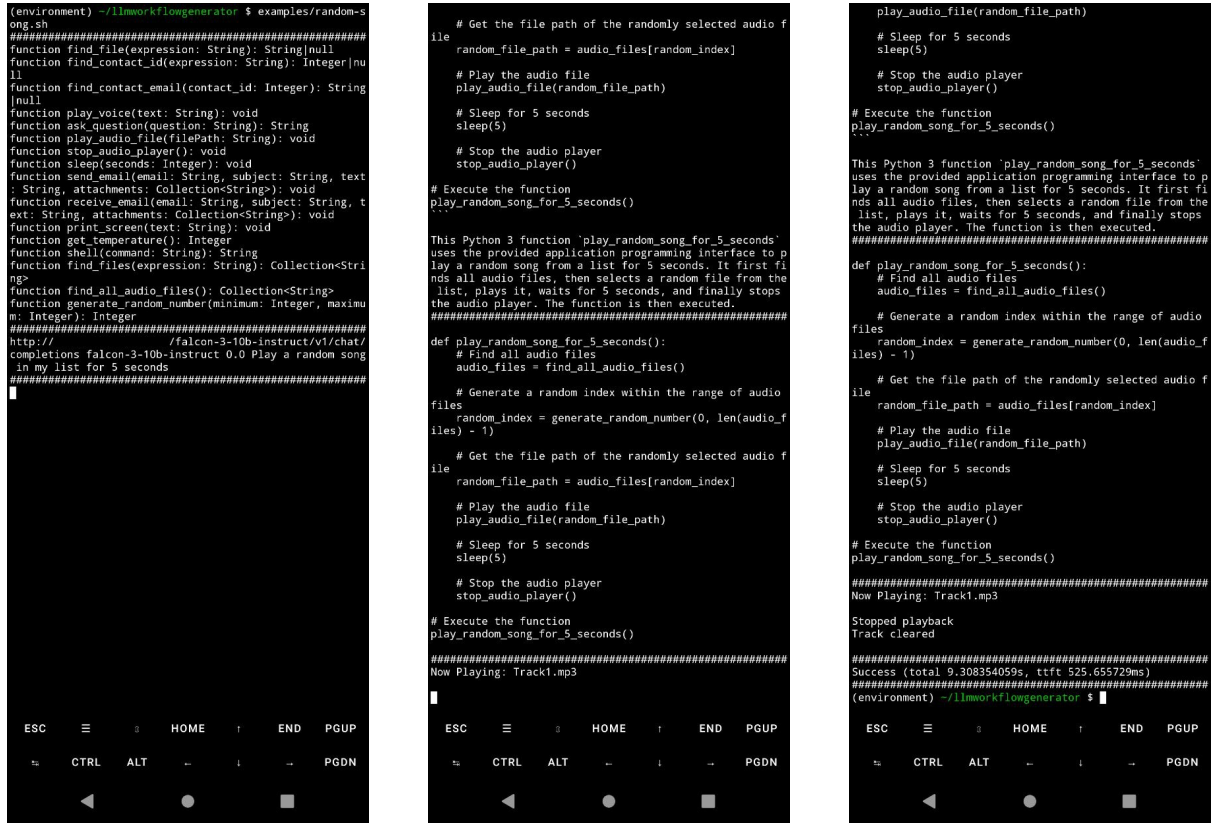
Figure 5: Exemplary execution trace for Intention 4 using `falcon-3-10b-instruct`.

Model	1	2	3	4	5	6	7	8	9	✓	✗
`falcon-3-10b-instruct`	✓	✓	✓	✓	✓	✓	✗	✓	✗	7	2
`phi-4`	✓	✓	✓	✓	✓	✓	✓	✗	✗	7	2
`qwen-2.5-14b-instruct`	✓	✗	✓	✓	✓	✓	✓	✗	✓	7	2
`gpt-4o`	✓	✓	✓	✓	✓	✓	✓	✗	✓	8	1
`gpt-4o-mini`	✓	✓	✓	✓	✓	✓	✓	✗	✓	8	1
`gpt-4-turbo`	✗	✓	✓	✓	✓	✓	✓	✗	✓	7	2
`gpt-4.5-preview-2025-02-27`	✓	✓	✓	✓	✓	✓	✓	✗	✓	8	1

Table 1

Synopsis of successful (✓) and unsuccessful (✗) user intention resolutions that were achieved by the employed LLM.

emerges from the experiments, indicating the efficacy of LLMs in facilitating automatic, machine-supported user intention resolution. This consensus extends beyond proprietary models to encompass both open-source and open-access models. The reasons for failing user intention resolutions vary and depend on the particular LLM.

The findings of the present study demonstrate that open-source and open-access models `falcon-3--10b-instruct`, `phi-4` and `qwen-2.5-14b-instruct` and encompass seven out of nine intention resolutions. This is on par with the proprietary model `gpt-4-turbo`. For the other proprietary models `gpt-4o`, `gpt-4o-mini` and `gpt-4.5-preview-2025-02-27` eight intention resolutions succeed.

`qwen-2.5-14b-instruct` has issues with the elementary intention 2, since, it utilizes the `ask_question` function. These finding suggests that there are issues with the correct interpretation of the user intention as well as the given task. Furthermore, the elementary intention 1 is not fulfilled by the proprietary LLM `gpt-4-turbo`. The code that is generated is accurate. However, the generated function is not invoked, despite being proactively instructed to do so in the provided prompt.

Model	1	2	3	4	5	6	7	8	9	✓	✗
`falcon-3-10b-instruct`	✗	✗	✗	✗	✗	✗	✗	✗	✗	0	9
`phi-4`	✗	✗	✗	✗	✗	✗	✗	✗	✗	0	9
`qwen-2.5-14b-instruct`	✗	✗	✗	✗	✗	✗	✗	✗	✗	0	9
`gpt-4o`	✓	✓	✓	✓	✓	✓	✓	✓	✓	9	0
`gpt-4o-mini`	✗	✗	✗	✗	✗	✗	✗	✗	✗	0	9
`gpt-4-turbo`	✗	✗	✗	✗	✗	✗	✗	✗	✗	0	9
`gpt-4.5-preview-2025-02-27`	✓	✓	✓	✓	✓	✓	✓	✓	✓	9	0

Table 2
Overview of preamble and postamble including (✓) and excluding (✗) user intention resolutions.

`falcon-3-10b-instruct` fails with intention 7, since it responds with an incorrect code block marker, using `<|assistant|>` instead of `python```` for code initiation. Notwithstanding, the resulting code is both accurate and profound. The LLM effectively addresses the user's intended purpose by employing control structures for error handling. However, intention 9 fails due to interpretation issues. For this intention, `falcon-3-10b-instruct` utilizes the wrong set of functions for resolving the provided user intention. It should be noted that `phi-4` experiences difficulties with intention 9 as well. The LLM utilizes the `query_llm` function to query itself for the required command. However, it does not successfully extract the command from the response. It is noteworthy that the command is executed directly without the response, and the `shell` function for command invocation is incorporated correctly.

In general, the resolution of intention 8 is unsuccessful for all aforementioned LLMs. Although the LLM `falcon-3-10b-instruct` is successful, it employs an alternative method for achieving that success. Initially, it was hypothesized that the LLMs would employ the `http_get_request` function to retrieve article content. Subsequently, the `query_llm` function would be utilized for the purposes of reading, comprehending, and creating an article summary. This approach is not applicable in the case of the article under consideration due to its size and the inclusion of a comprehensive set of website building blocks consisting of HTML and CSS, in addition to the written text. The majority of the aforementioned LLMs respond with either a *Bad Request* HTTP response or some other HTTP client error response upon invoking the `query_llm` function, as the inputs exceed the context windows. It is determined that the `falcon-3-10b-instruct` does not employ the `http_get_request` function. Rather, it passes the intention directly to the `query_llm` function for generating a response from its own internal knowledge.

A notable finding pertains to preambles and postambles. The open-source and open-access models, namely `falcon-3-10b-instruct`, `phi-4` and `qwen-2.5-14b-instruct`, do not incorporate any preambles or postambles. It is demonstrated that the proprietary LLMs `gpt-4-turbo` and `gpt-4o-mini` correctly exclude preambles and postambles as well. However, `gpt-4.5-preview--2025-02-27` and `gpt-4o` include them. A detailed overview for each user intention resolution is shown in Table 2.

As illustrated in Table 3, the following provides a comprehensive overview of the user intentions for which the particular LLM includes code comments. According to the data presented, for `falcon--3-10b-instruct`, no discernible trends are identified. However, the `phi-4` includes comments for each user intention resolution. `qwen-2.5-14b-instruct` includes code comments a total of 3 times, showing that it tends more toward exclusion. In general, the proprietary models developed by OpenAI have a tendency to incorporate code comments. While the `gpt-4o` model exhibits a single instance of excluding comments, the `gpt-4o-mini` demonstrates a threefold occurrence of such exclusion. As was the case with `phi-4`, `gpt-4-turbo`, and `gpt-4.5-preview-2025-02-27` incorporate them for all 9 user intention resolutions. Intention 8 merits particular attention, as it is noteworthy that all LLMs incorporate code comments, despite the aforementioned challenges in addressing them.

Model	1	2	3	4	5	6	7	8	9	✓	✗
`falcon-3-10b-instruct`	✗	✗	✗	✓	✓	✓	✗	✓	✗	4	5
`phi-4`	✓	✓	✓	✓	✓	✓	✓	✓	✓	9	0
`qwen-2.5-14b-instruct`	✓	✗	✗	✗	✗	✗	✓	✓	✗	3	6
`gpt-4o`	✗	✓	✓	✓	✓	✓	✓	✓	✓	8	1
`gpt-4o-mini`	✗	✓	✓	✗	✓	✓	✓	✓	✗	6	3
`gpt-4-turbo`	✓	✓	✓	✓	✓	✓	✓	✓	✓	9	0
`gpt-4.5-preview-2025-02-27`	✓	✓	✓	✓	✓	✓	✓	✓	✓	9	0

Table 3
Overview of code comments including (✓) and excluding (✗) user intention resolutions.

Model	≈ Average Response Time (s)	≈ Average Time to First Token (ms)
`falcon-3-10b-instruct`	6.39	353.4
`phi-4`	7.16	398.4
`qwen-2.5-14b-instruct`	3.42	390.6
`gpt-4o`	1.75	539.9
`gpt-4o-mini`	3.99	498.3
`gpt-4-turbo`	6.53	883.1
`gpt-4.5-preview-2025-02-27`	7.24	900.1

Table 4
Average Response Time and Time to First Token for each LLM.

	GPTs Included	GPTs Excluded
Fastest Response Time	9/9: `gpt-4o`	2/9: `falcon-3-10b-instruct` 1/9: `phi-4` 6/9: `qwen-2.5-14b-instruct`
Slowest Response Time	4/9: `gpt-4.5-preview-2025-02-27` 1/9: `falcon-3-10b-instruct` 2/9: `gpt-4-turbo` 2/9: `phi-4`	2/9: `falcon-3-10b-instruct` 7/9: `phi-4`
Fastest Time to First Token	9/9: `falcon-3-10b-instruct`	9/9: `falcon-3-10b-instruct`
Slowest Time to First Token	6/9: `gpt-4.5-preview-2025-02-27` 3/9: `gpt-4-turbo`	6/9: `phi-4` 3/9: `qwen-2.5-14b-instruct`

Table 5
Count of leading metrics.

Table 4 presents the mean metrics of the Response Time and the Time to First Token. In Table 5 the leading amount for the particular metric is indicated. The presentation of these models is accompanied by an examination of both inclusion and exclusion, utilizing proprietary models from OpenAI. `gpt-4o` provides the most rapid response time. With the exception of proprietary models, the `qwen-2.5-14b-instruct` generally exhibits the most rapid response time for the majority of user intentions. With respect to the response time, the `gpt-4.5-preview-2025-02-27` model demonstrates the slowest performance. With the exception of the proprietary models from OpenAI, `phi-4` shows the slowest performance for the majority of user intentions. For the time to first token metric, the `falcon-3-10b-instruct` offers the optimal performance, both with and without the consideration of the proprietary models, as it provides the most expeditious time to first token for each resolution. A thorough investigation into the slowest time to first token with the incorporation of the proprietary models reveals that the `gpt-4.5-preview-2025-02-27` model manifests in 6 out of 9 instances, while the `gpt-4-turbo` emerges in the remaining 3 cases. Excluding the proprietary models reveals that the `phi-4` provides the slowest Time to First Token metric for the majority of 6 cases, while the `qwen-2.5-14b-instruct` leads for the other 3 cases.

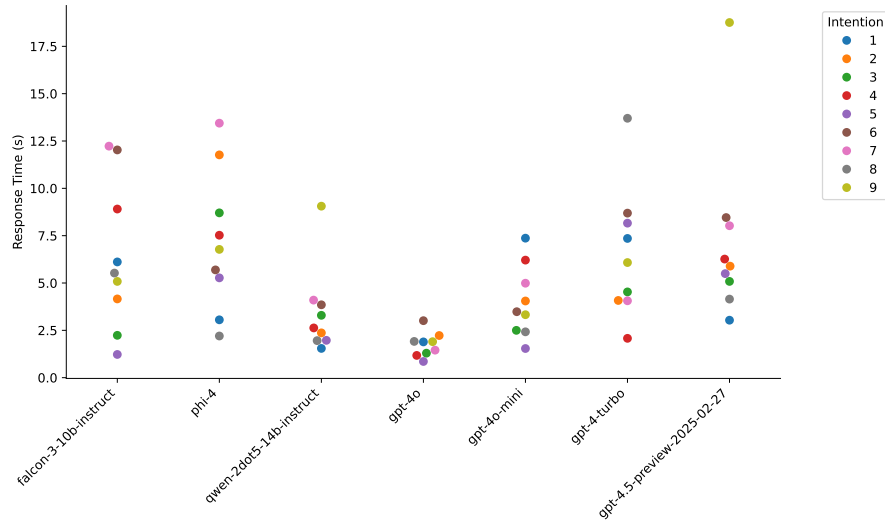A thorough examination of the Response Time and Time to First Token metrics, meticulously grouped

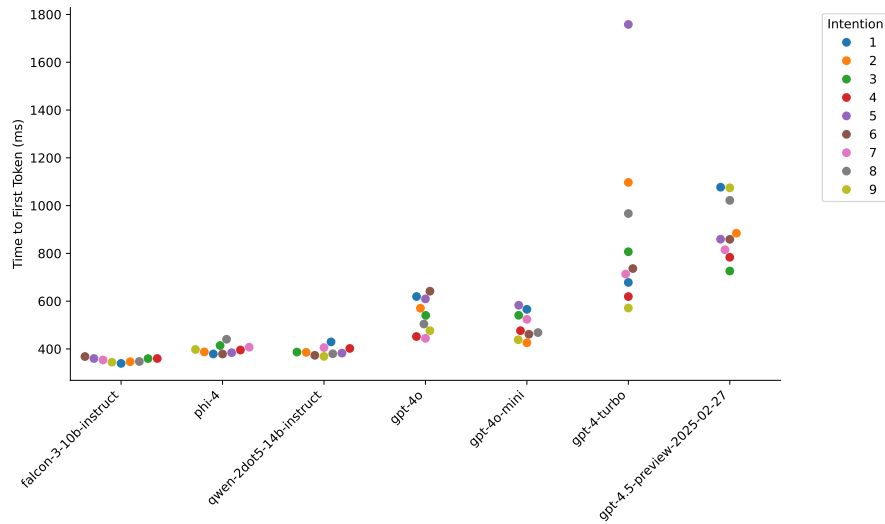**Figure 6:** Response Time for each LLM and the respective user intention resolution.

Figure 7: Time to First Token for each LLM and the respective user intention resolution.

by the specific LLM and the user intention, is elucidated in Figure 6 and Figure 7. The findings indicate that the average performance of the `falcon-3-10b-instruct` model is marginally superior to that of the `phi-4` model with respect to Response Time. Nonetheless, both models demonstrate deficiencies when compared with the superior `qwen-2.5-14b-instruct` model, which approaches parity with the proprietary `gpt-4o` model. For the majority of user intention resolutions, the performance of the `falcon-3-10b-instruct` model and the `phi-4` model is comparable to that of the `gpt-4--turbo` model and the `gpt-4.5-preview-2025-02-27` model. However, the performance of the `gpt-4-turbo` model and the `gpt-4.5-preview-2025-02-27` model is slightly inferior to that of the `gpt-4o-mini` model. A comparative analysis reveals that the open-access and open-source LLMs, namely `falcon-3-10b-instruct`, `phi-4`, and `qwen-2.5-14b-instruct`, demonstrate superior performance to proprietary models from OpenAI regarding the Time to First Token. A comparison of the performance of the models reveals that the `gpt-4o` and the `gpt-4o-mini` demonstrate comparable results. Conversely, the `gpt-4-turbo` and the `gpt-4.5-preview-2025-02-27` exhibit substandard performance.

4. Discussion and Result Interpretation

The experiments and results presented in Section 3 demonstrate that the semantic quality of responses is contingent on the specific user intention. It has been observed that none of the aforementioned LLMs demonstrate the capacity to adequately address all the user intentions provided. While proprietary models from OpenAI demonstrate a slight advantage, with an average of one additional successful outcome, the findings unmistakably underscore the substantial progress achieved by open-source and open-access models. In addressing the previously formulated research question, the present study offers a demonstration of the feasibility of employing open-access and open-source models as intermediate and middleware components for decomposing given user intentions into workflows. From a semantic perspective, the experimental models under consideration facilitate the decomposition of user intentions into actionable steps with a degree of efficacy that is nearly equivalent to that of proprietary models. Despite the fact that the proprietary flagship models demonstrated leadership in terms of the average Response Time metric throughout the course of the experiments, the performance of the open-source and open-access models remained within the acceptable range of a couple of seconds. A salient detail worthy of emphasis is that each model was exposed to experimentation on merely a single instance. The objective of this study was not to establish a statistical benchmark, but rather to compare the general ability of different LLMs to translate everyday intentions into executable workflows. A single-run configuration is indicative of realistic usage patterns, wherein users typically articulate an intention on a single occasion and anticipate a response. This configuration also circumvents the potential for bias from repeated sampling, which might favor certain models.

Subsequent endeavors in this specific application domain pertain to the optimization of the aforementioned models for the purpose of further reducing the introduced system architecture. The efficacy of LLMs is contingent upon their incorporation into local devices. However, the substantial computational demands of the inference process currently necessitate execution on remote infrastructure. The processes of pruning, distillation, and quantization offer significant opportunities for the operation of LLMs at local scale. By decreasing the model size and computational demands without a substantial compromise in performance, these techniques enable the implementation of sophisticated AI models on devices with limited resources, not exclusive to mobile devices. Collectively, these technologies facilitate enhanced accessibility, reduced operational expenditures, augmented privacy measures, and expedited response times, unveiling novel prospects for real-world, on-device AI application and enabling the focus on operating system-oriented optimization of intent-based user interaction mechanisms. In this context, open-source and open-access models assume a particularly salient role. It is imperative to acknowledge that the reduction and optimization of the aforementioned models is not the sole pivotal step. While the employment of imperative programming languages as intermediate representations for workflows functions effectively in conjunction with LLMs, the necessity arises for an all-encompassing API to address the diverse user intents. This issue must be given due consideration for future research endeavors. Furthermore, the transition of authority and decision-making capacity to LLMs and AI in general gives rise to a substantial security concern, thereby prompting the exploration of critical research domains. The deliberate or inadvertent application of LLMs has the potential to result in adverse consequences. To illustrate this point, the direct execution of generated code in the system architecture under consideration introduces a security vulnerability. It is necessary to implement significant isolation and sandboxing mechanisms, as well as utilize operating system-provided capabilities. Another exemplary vector of attack in the experiments presented is targeted around the *shell* command, since it can be misused for direct access to the system, depending on the particular system configuration and setup. These findings align with the recent studies addressing the *Shutdown Problem* [7][8], which demonstrate that AI proactively undertakes measures to circumvent system shutdown. *Alignment Faking* [9] further illustrates how contemporary LLMs exhibit resistance to human intervention and correction. It is imperative to devise countermeasures and techniques to circumvent potential damage that may be engendered by the integration of LLMs and AI.

5. Related Work

In the seminal paper, the *Transformer* is introduced [10], which is a novel deep learning model founded on self-attention, which supersedes earlier Recurrent Neural Networks (RNNs) and Convolutional Neural Networks (CNNs). The Transformer model is distinguished by its parallelisability, accelerated training, and enhanced quality in Natural Language Processing [11][12]. This fundamental principle underlies the construction of all contemporary LLMs, including GPT, BERT, Falcon, Phi, and Qwen. A thorough examination of transformer design optimizations through the initial months of 2024 is elucidated in [13]. The overview encompasses *FlashAttention-2*, *Mixture of Experts* and *Long Context Transformers*. The widespread availability of ChatGPT [14] has led to a substantial increase in the number of applications under consideration. For instance, the application domains of public health and medicine have been the focus of study [15][16][17], as well as those of education and pedagogy [18][19]. The general applicability of AI necessitates its categorization, a subject that is addressed in [20]. Existing solutions such as Siri, Cortana as well as Alexa influenced the application domain of personal assistants. An overview of requirements of voice user interfaces, in particular for for blind and visually impaired users, is addressed by [21]. The integration of LLMs for machine-oriented user intention resolution is examined in [22] and [23], which also present *AIOS*, an operating system for LLM-based agents. [24] delineates a vision for AIOS as the core of future vehicle systems research. Recent research in this particular application domain includes the training of AI to directly operate existing GUI applications [25]. Subsequent research in the context of LLMs entails the investigation of the potential of LLMs to facilitate the recognition of user intentions within dialog systems [26]. A prototype tool that automatically generates business and scientific workflows using LLMs is presented in [27]. Another application of AI, particularly LLMs, is the generation of code, a subject that has been extensively studied. Tools such as GitHub Copilot provide assistance to engineers in routine tasks [28][29]. An evaluation of problem-solving through code generation with GPT language models has been conducted in [30] [31]. Common issues associated with the utilization of LLMs, including hallucinations and erroneous code generation, are addressed by techniques that are centered around fuzzing as well as static analysis [32] A number of novel approaches have been developed that utilize *Grammar Augmentation* [33] and the redesign of fundamental transformer decoding algorithms [3]. The utilization of AI and LLMs entails specific risks [34] and necessitates a systematic taxonomy of these risks, as outlined in [35]. Among the most critical aspects are privacy-related concerns associated with training data [36], as well as the handling of sensitive user information from communication platforms. A comprehensive survey on approaches to data privacy protection is provided in [37]. Further issues are related to linguistic biases [38].

6. Conclusion

This work presents a comparative analysis of various LLMs for machine-assisted resolution of user intentions. The efficacy of the open-access and open-source models `falcon-3-10b-instruct`, `phi-4`, and `qwen-2.5-14b-instruct` is demonstrated to be comparable to that of the proprietary fourth-generation GPT models from OpenAI, particularly in the aforementioned application domain. The experimental results indicate that while the current flagship model `gpt-4o` shows the shortest average response time, the collected metrics of the open-access and open-source models remained within an acceptable range. The mentioned models, namely `falcon-3-10b-instruct`, `phi-4`, and `qwen--2.5-14b-instruct`, are comparable to other proprietary models, such as `gpt-4o`, `gpt-4-turbo`, `gpt-4.5-preview-2025-02-27`. This provides a promising foundation for the future development of systems that employ self-hosted models and integrate them with LLMs to achieve greater autonomy, facilitating the translation of user intentions into workflows and their subsequent resolution.

Declaration on Generative AI

During the preparation of this work, the author(s) used DeepL Write in order to: Grammar and spelling check. After using these tool(s)/service(s), the author(s) reviewed and edited the content as needed and take(s) full responsibility for the publication's content.

References

[1] H. Jin, Y. Zhang, D. Meng, J. Wang, J. Tan, A comprehensive survey on process-oriented automatic text summarization with exploration of llm-based methods, arXiv preprint arXiv:2403.02901 (2024).

[2] T. Zhang, Y. Zhang, V. Vineet, N. Joshi, X. Wang, Controllable text-to-image generation with gpt-4, arXiv preprint arXiv:2305.18583 (2023).

[3] S. Zhang, Z. Chen, Y. Shen, M. Ding, J. B. Tenenbaum, C. Gan, Planning with large language models for code generation, 2023. doi:`10.48550/arXiv.2303.05510`. `arXiv:2303.05510`.

[4] C. Xie, D. Zou, A human-like reasoning framework for multi-phases planning task with large language models, arXiv preprint arXiv:2405.18208 (2024).

[5] J. Flerlage, I. Behnke, O. Kao, Towards machine-generated code for the resolution of user intentions, 2025. URL: https://arxiv.org/abs/2504.17531. `arXiv:2504.17531`.

[6] J. Nielsen, Usability engineering, Morgan Kaufmann, 1994.

[7] E. Thornley, The shutdown problem: an ai engineering puzzle for decision theorists, Philosophical Studies (2024) 1–28.

[8] E. Thornley, The shutdown problem: Three theorems, arXiv e-prints (2024) arXiv–2403.

[9] R. Greenblatt, C. Denison, B. Wright, F. Roger, M. MacDiarmid, S. Marks, J. Treutlein, T. Belonax, J. Chen, D. Duvenaud, et al., Alignment faking in large language models, arXiv preprint arXiv:2412.14093 (2024).

[10] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, I. Polosukhin, Attention is all you need, Advances in neural information processing systems 30 (2017).

[11] D. Kalla, N. Smith, F. Samaah, S. Kuraku, Study and analysis of chat gpt and its impact on different fields of study, International journal of innovative science and research technology 8 (2023). doi:`10.5281/zenodo.7767675`.

[12] S. Minaee, T. Mikolov, N. Nikzad, M. Chenaghlu, R. Socher, X. Amatriain, J. Gao, Large language models: A survey, arXiv preprint arXiv:2402.06196 (2024).

[13] Y. Tay, M. Dehghani, D. Bahri, D. Metzler, Efficient transformers: A survey, ACM Comput. Surv. 55 (2022). URL: https://doi.org/10.1145/3530811. doi:`10.1145/3530811`.

[14] T. Wu, S. He, J. Liu, S. Sun, K. Liu, Q.-L. Han, Y. Tang, A brief overview of chatgpt: The history, status quo and potential future development, IEEE/CAA Journal of Automatica Sinica 10 (2023) 1122–1136. doi:`10.1109/JAS.2023.123618`.

[15] S. S. Biswas, Role of chat gpt in public health, Annals of biomedical engineering 51 (2023) 868–869. doi:`10.1007/s10439-023-03172-7`.

[16] A. J. Thirunavukarasu, D. S. J. Ting, K. Elangovan, L. Gutierrez, T. F. Tan, D. S. W. Ting, Large language models in medicine, Nature medicine 29 (2023) 1930–1940.

[17] Z. A. Nazi, W. Peng, Large language models in healthcare and medical domain: A review, in: Informatics, volume 11, MDPI, 2024, p. 57.

[18] M. Firat, How chat gpt can transform autodidactic experiences and open education? (2023). doi:`10.31219/osf.io/9ge8m`.

[19] E. Kasneci, K. Seßler, S. Küchemann, M. Bannert, D. Dementieva, F. Fischer, U. Gasser, G. Groh, S. Günnemann, E. Hüllermeier, et al., Chatgpt for good? on opportunities and challenges of large language models for education, Learning and individual differences 103 (2023) 102274.

[20] M. R. Morris, J. Sohl-Dickstein, N. Fiedel, T. Warkentin, A. Dafoe, A. Faust, C. Farabet, S. Legg, Position: Levels of agi for operationalizing progress on the path to agi, in: Forty-first International Conference on Machine Learning, 2024. URL: 10.48550/arXiv.2311.02462.

[21] C. Oumard, J. Kreimeier, T. Götzelmann, Pardon? an overview of the current state and requirements of voice user interfaces for blind and visually impaired users, in: International Conference on Computers Helping People with Special Needs, Springer, 2022, pp. 388–398.

[22] Z. Shi, K. Mei, M. Jin, Y. Su, C. Zuo, W. Hua, W. Xu, Y. Ren, Z. Liu, M. Du, et al., From commands to prompts: Llm-based semantic file system for aios, arXiv preprint arXiv:2410.11843 (2024).

[23] K. Mei, Z. Li, S. Xu, R. Ye, Y. Ge, Y. Zhang, Aios: Llm agent operating system, arXiv e-prints, pp. arXiv–2403 (2024). URL: 10.48550/arXiv.2403.16971.

[24] J. Ge, C. Chang, J. Zhang, L. Li, X. Na, Y. Lin, L. Li, Llm-based operating systems for automated vehicles: A new perspective, IEEE Transactions on Intelligent Vehicles PP (2024) 1–5. doi:10.1109/TIV.2024.3399813.

[25] X. Liu, B. Qin, D. Liang, G. Dong, H. Lai, H. Zhang, H. Zhao, I. L. Iong, J. Sun, J. Wang, J. Gao, J. Shan, K. Liu, S. Zhang, S. Yao, S. Cheng, W. Yao, W. Zhao, X. Liu, X. Liu, X. Chen, X. Yang, Y. Yang, Y. Xu, Y. Yang, Y. Wang, Y. Xu, Z. Qi, Y. Dong, J. Tang, Autoglm: Autonomous foundation agents for guis, 2024. URL: https://arxiv.org/abs/2411.00820. arXiv:2411.00820.

[26] G. Arora, S. Jain, S. Merugu, Intent detection in the age of llms, 2024. URL: https://arxiv.org/abs/2410.01627. arXiv:2410.01627.

[27] J. Xu, W. Du, X. Liu, X. Li, Llm4workflow: An llm-based automated workflow model generation tool, in: Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering, 2024, pp. 2394–2398.

[28] A. Moradi Dakhel, V. Majdinasab, A. Nikanjam, F. Khomh, M. C. Desmarais, Z. M. J. Jiang, Github copilot ai pair programmer: Asset or liability?, Journal of Systems and Software 203 (2023) 111734. URL: https://www.sciencedirect.com/science/article/pii/S0164121223001292. doi:10.1016/j.jss.2023.111734.

[29] M. Wermelinger, Using github copilot to solve simple programming problems, in: Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1, SIGCSE 2023, Association for Computing Machinery, New York, NY, USA, 2023, p. 172–178. URL: https://doi.org/10.1145/3545945.3569830. doi:10.1145/3545945.3569830.

[30] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. D. O. Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman, et al., Evaluating large language models trained on code, arXiv preprint arXiv:2107.03374 (2021). doi:10.48550/arXiv.2107.03374.

[31] F. Lin, D. J. Kim, Tse-Husn, Chen, Soen-101: Code generation by emulating software process models using large language model agents, 2024. URL: https://arxiv.org/abs/2403.15852. arXiv:2403.15852.

[32] S. Ouyang, J. M. Zhang, M. Harman, M. Wang, An empirical study of the non-determinism of chatgpt in code generation, ACM Transactions on Software Engineering and Methodology (2024). URL: http://dx.doi.org/10.1145/3697010. doi:10.1145/3697010.

[33] S. Ugare, T. Suresh, H. Kang, S. Misailovic, G. Singh, Syncode: Llm generation with grammar augmentation, 2024. doi:10.48550/arXiv.2403.01632. arXiv:2403.01632.

[34] E. M. Bender, T. Gebru, A. McMillan-Major, S. Shmitchell, On the dangers of stochastic parrots: Can language models be too big?, in: Proceedings of the 2021 ACM conference on fairness, accountability, and transparency, 2021, pp. 610–623.

[35] L. Weidinger, J. Uesato, M. Rauh, C. Griffin, P.-S. Huang, J. Mellor, A. Glaese, M. Cheng, B. Balle, A. Kasirzadeh, et al., Taxonomy of risks posed by language models, in: Proceedings of the 2022 ACM conference on fairness, accountability, and transparency, 2022, pp. 214–229.

[36] N. Carlini, F. Tramer, E. Wallace, M. Jagielski, A. Herbert-Voss, K. Lee, A. Roberts, T. Brown, D. Song, U. Erlingsson, et al., Extracting training data from large language models, in: 30th USENIX security symposium (USENIX Security 21), 2021, pp. 2633–2650.

[37] B. Yan, K. Li, M. Xu, Y. Dong, Y. Zhang, Z. Ren, X. Cheng, On protecting the data privacy of large language models (llms): A survey, arXiv preprint arXiv:2403.05156 (2024).

[38] E. Fleisig, G. Smith, M. Bossi, I. Rustagi, X. Yin, D. Klein, Linguistic bias in chatgpt: Language models reinforce dialect discrimination, arXiv preprint arXiv:2406.08818 (2024).