GPTVoiceTasker: Advancing Multi-step Mobile Task Efficiency Through Dynamic Interface Exploration and Learning

Minh Duc Vu
dustin.vu@csiro.au
CSIRO's Data61
Melbourne, Australia

Han Wang*
han.wang@monash.edu
Monash University
Melbourne, Australia

Zhuang Li
zhuang.li@monash.edu
Monash University
Melbourne, Australia

Jieshan Chen
jieshan.chen@data61.csiro.au
CSIRO's Data61
Sydney, Australia

Shengdong Zhao
City University of Hong Kong
Hong Kong, China
shezhao@cityu.edu.hk

Zhenchang Xing
CSIRO's Data61 & Australian National University
Canberra, Australia
zhenchang.xing@data61.csiro.au

Chunyang Chen†
Technical University of Munich & Monash University
Heilbronn, Germany
chun-yang.chen@tum.de

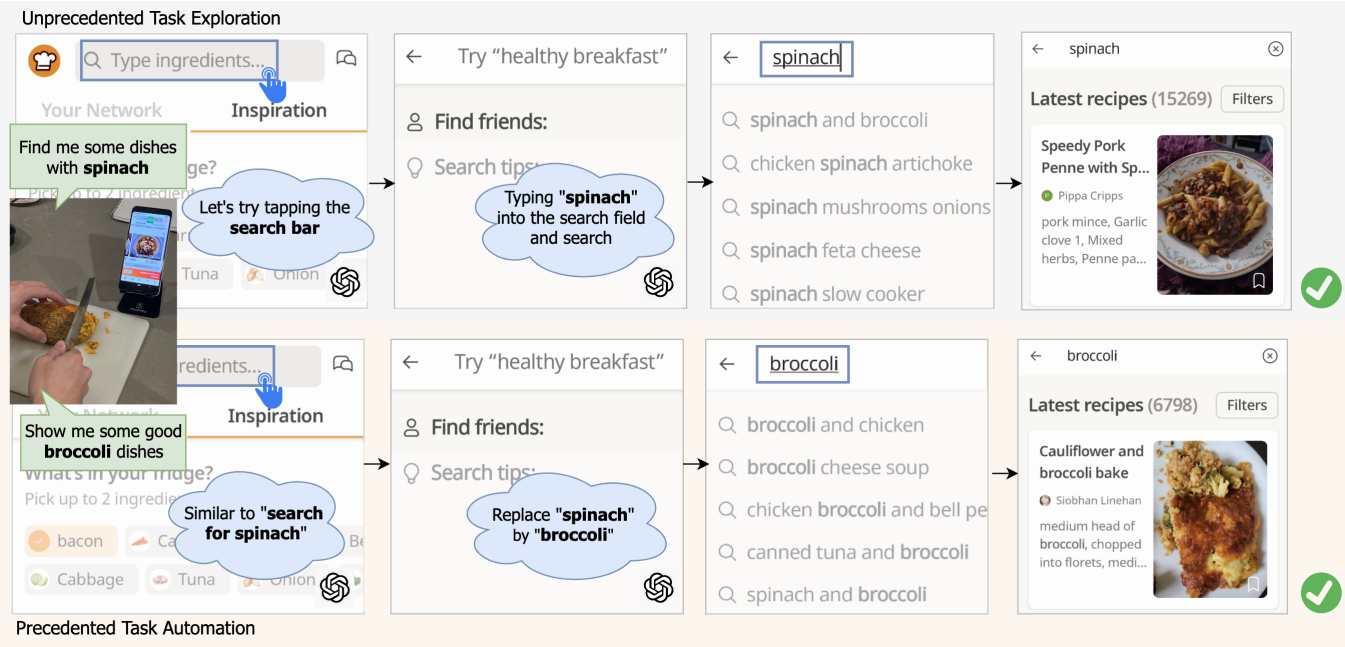

Figure 1: GptVoiceTasker provides an intuitive method for automating complex commands on smartphones during physically demanding activities, such as cooking. It automatically explores step-by-step interactions to complete unprecedented tasks and uses the saved information to accelerate the automation process for tasks that have been previously encountered.

*Minh Duc Vu and Han Wang contributed equally.
†Chunyang Chen is the corresponding author.

UIST '24, October 13–16, 2024, Pittsburgh, PA, USA

ABSTRACT

Virtual assistants have the potential to play an important role in helping users achieves different tasks. However, these systems face

ACM ISBN 979-8-4007-0628-8/24/10
https://doi.org/10.1145/3654777.3676356

challenges in their real-world usability, characterized by inefficiency and struggles in grasping user intentions. Leveraging recent advances in Large Language Models (LLMs), we introduce GptVoiceTasker, a virtual assistant poised to enhance user experiences and task efficiency on mobile devices. GptVoiceTasker excels at intelligently deciphering user commands and executing relevant device interactions to streamline task completion. For unprecedented tasks, GptVoiceTasker utilises the contextual information and on-screen content to continuously explore and execute the tasks. In addition, the system continually learns from historical user commands to automate subsequent task invocations, further enhancing execution efficiency. From our experiments, GptVoiceTasker achieved 84.5% accuracy in parsing human commands into executable actions and 85.7% accuracy in automating multi-step tasks. In our user study, GptVoiceTasker boosted task efficiency in real-world scenarios by 34.85%, accompanied by positive participant feedback. We made GptVoiceTasker open-source, inviting further research into LLMs utilization for diverse tasks through prompt engineering and leveraging user usage data to improve efficiency.

CCS CONCEPTS

• **Human-centered computing → Interaction techniques**; **Smartphones**; **Natural language interfaces**; *Sound-based input / output*.

ACM Reference Format:
Minh Duc Vu, Han Wang, Zhuang Li, Jieshan Chen, Shengdong Zhao, Zhenchang Xing, and Chunyang Chen. 2024. GPTVoiceTasker: Advancing Multi-step Mobile Task Efficiency Through Dynamic Interface Exploration and Learning. In *The 37th Annual ACM Symposium on User Interface Software and Technology (UIST '24), October 13–16, 2024, Pittsburgh, PA, USA.* ACM, New York, NY, USA, 17 pages. https://doi.org/10.1145/3654777.3676356

1 INTRODUCTION

The advancements in voice control technology have sparked a new wave of innovation, driving the exploration of its potential in transforming smartphone interactions [31, 32]. With the integration of voice control, users can effortlessly navigate through various applications, compose messages, and even initiate tasks like checking the weather or playing a video on YouTube [67]. This natural mode of interaction not only saves time but also promotes a hands-free experience, allowing individuals to engage with their smartphones in situations where manual input operations are impractical or inconvenient [44, 80] (refer to Figure. 2 illustrating a user engaged in gym activities). Moreover, the success stories of widely recognized voice assistants like Google Voice Assistant [4] and Siri [2] have further propelled the adoption of voice-based interactions, inspiring researchers and developers to delve deeper into its capabilities and refine its usability for an even broader range of users.

Developing autonomous voice-controlled assistants involves addressing various challenges that impact the usability of these systems [50]. Current industrial products (such as Siri and Google Assistant) do not provide a universal approach as they require app developers to explicitly define a narrow set of voice-supported actions within the code. For instance, while those voice assistants fully support searching for videos on YouTube, other YouTube features such as accessing video history is unavailable. This results in a disjointed and often unfinished experience for users as they cannot rely on voice interactions to fully control their smartphone. In addition, these assistants often struggle to comprehend commands when users' inputs are unclear or do not align with the predefined patterns [23, 24]. This lack of understanding further impacts the overall utility of the voice-controlled systems. Furthermore, app developers face the challenge of anticipating and programming an extensive variety of potential intents, a task that is both complex and limiting in the scope of voice assistant capabilities.

Recently, Large Language Models (LLMs) have brought a paradigm shift in natural language processing (NLP), demonstrating remarkable capabilities in tasks like reading comprehension, translation, and text completion [8]. The advent of Few-Shot Learning has further amplified the capabilities of LLMs, enabling them to quickly adapt to new logical reasoning tasks with minimal examples. This versatility and efficiency in managing various conversational interactions without the need for extensive retraining offer a groundbreaking approach, obviating the need for task-specific models and extensive datasets [57]. In the context of mobile assistants/agents, the integration of LLMs to comprehend user commands and interface with mobile UIs has gained traction [13, 53, 68, 72, 77]. Amazon's integration of LLMs into Alexa represents a significant step in enhancing voice assistants [20]. Their primary focus has been on improving Alexa's ability to understand users' needs more accurately and to control other devices more effectively. But it's not clear how Alexa can help with smartphones users and if the assistant is optimized with mobile UIs. Some research in this area concentrates on translating mobile GUIs into text, relying on LLMs to understand the context and predict interactive screen elements [68, 72]. However, this method sometimes struggles with the extraction of irrelevant GUI elements or fails when the command does not directly relate to the current screen. Other UI automation tools for UI testing explored enhancing LLMs with image processing capabilities, such as those found in GPT-4v [77]. While this approach shows a higher success rate, it is hampered by longer processing times and increased costs, which can negatively impact user experience in real-time systems like voice assistants.

This paper introduces GptVoiceTasker, a novel voice assistant that automates multi-step unprecedented tasks by dynamically exploring app interface and accelerate similar tasks through prior usages. Drawing inspiration from the conventional record-and-replay approach [38, 41], GptVoiceTasker is designed to learn GUI transformations as users navigate through apps. Instead of simply transmitting live UI information, GptVoiceTasker captures information of the current UI as the user interacts with the app, storing this data in a backend database as known knowledge to the system. Whenever a user command is issued, GptVoiceTasker cross-references the new executing task with this stored knowledge to make informed decisions. This method allows reproduction of tasks for similar future requests, enhancing task efficiency and accuracy. If a user command references a unprecedented feature, GptVoiceTasker will step-by-step explore and record the navigational path to achieve the feature. Our system achieves this by advanced prompt engineering techniques to ensure a precise understanding of user commands without extensive model training. GptVoiceTasker effectively bridges the divide between natural language commands and interactive mobile tasks, enabling seamless

automation of everyday tasks that include actions like scrolling, tapping, and text input purely via voice commands.

We validated the technical contributions of GptVoiceTasker by evaluating i) the ability to parse user commands into executable actions, ii) the ability to complete multi-step tasks given one command, and iii) the ability to streamline saved tasks. The command parser achieved over 90% accuracy on a human command dataset collected from the user study. GptVoiceTasker also achieved 85.7% success rate in completing human-collected multi-step tasks. Our automated execution achieved 82.7% success rate for direct match tasks and 72.0% success rate for tasks with different parameters. To validate the usability of GptVoiceTasker, we conducted a user evaluation with 18 participants, each completing a set of tasks using GptVoiceTasker and two state-of-the-art baselines. We collected the time taken to complete each task, as well as quantitative and quality feedback from users. The results showed that GptVoiceTasker accelerated the tasks by 34.85% and received positive feedback regarding usability.

To summarize, the contributions of this paper include:

- Development of GptVoiceTasker, a voice assistant that harnesses the capabilities of LLMs to streamline the automation of multi-step tasks by predicting the most optimal step on each individual screen.
- A graph-based local database design that automates the recording and retrieval of personal app usages, enhancing task execution efficiency for virtual assistant interactions.
- Conducting a user evaluation to validate the effectiveness of our approach, along with empirical findings on system limitations and considerations for voice assistant design.
- GptVoiceTasker[1] is open-sourced so that anyone can use and continue to improve the system.

2 BACKGROUND & RELATED WORKS

2.1 Voice Control & Automation on Mobile Devices

Recent advancements in Natural Language Understanding (NLU) have significantly enhanced the development of voice assistants across various platforms, including ubiquitous systems [6, 36] and home appliances [55].

An early milestone in smartphone voice control interfaces was JustSpeak, which harnessed Google's Automatic Speech Recognition (ASR) to record user commands and introduced innovative utterance parsing techniques [80]. Subsequently, the Smart Voice Assistant expanded on JustSpeak's capabilities by enabling users to manage calls and SMS through voice commands [9]. However, these initial approaches, foundational as they were, encountered usability issues stemming from rigid language parsing heuristics and limited use cases, which spurred the need for further development of smartphone virtual assistants.

In recent years, significant advancements in language parsing capabilities have been achieved through deep learning models. SAVANT leveraged Dialogflow as a conversational agent to extract user intent from utterances [3], while DoThisHere employed the pre-built Almond language model to enable voice control for retrieving and setting UI contents in Android [78]. Google released Voice Access [1], aimed to replace manual interactions with voice command, which has over 100 millions downloads on Google Play Store. Moreover, there are Firefox Voice, an open and extensible web-based voice assistant with speech-to-text engine [12], and Talk2Care, which leverages LLMs to facilitate communication between healthcare providers and older adults [79]. Additionally, Just Speak It has focused on minimizing cognitive load during eyes-free text editing with a smart voice assistant [26], while GazePointAR uses context-aware multimodal inputs for pronoun disambiguation in augmented reality [39]. Voicify [67] introduced VoicifyParser, an advanced deep learning approach for parsing user commands into on-screen interactions. However, the interaction paradigm with these existing approaches remains somewhat unnatural, requiring users to issue precise machine-like instructions, such as "*Press save button*". This limitation means that they may struggle to fully comprehend high-level user intentions, such as "*I want to save this note*". We propose GptVoiceTasker to address these challenges and revolutionise the voice-based interactions between human and software systems. Our solution leverages the capabilities of LLMs to map high-level user intentions to executable actions, enabling on-screen interactions through intention-based voice commands. This approach seeks to accommodate the flexibility and natural language of human commands, ushering in a new era of user-friendly assistive tools.

Research has also delved into voice command interfaces for automating smartphone tasks, often categorized as programming-by-demonstration tools [3]. These systems generally utilize a record-and-replay strategy, where the user records a series of actions to complete a task and later triggers that sequence with a voice command. SUGILITE [41], for instance, introduced methods for performing task variations with different parameters from a single recorded instance. Building upon this, AutoVCI [54] automated the generation of verbal commands for activating saved tasks. However, these tools entail usability challenges as they require users to manually record execution paths for each use case. In contrast, GptVoiceTasker innovates by predicting the most suitable action for each UI screen based on user requests without the need for pre-programming. Our database, tailored for streamlining task execution, is automatically constructed in real-time as users interact with their mobile apps.

2.2 Large Language Models for Enhanced Human-AI Collaboration

The advent of generative AI has given rise to innovative LLMs, such as GPT-4 [52], DALL-E [59] and Llama [66]. These LLMs have revolutionized the landscape of AI development by enabling developers to achieve complex tasks through few-shot prompting, eliminating the need for extensive custom model training. Their remarkable versatility has spurred active research in both IT and non-IT domains, spanning areas like software testing [27, 48], high-performance computing [15], finance [73], and health science [28]. LLMs have particularly excelled in enhancing the intuitiveness of existing methods, as seen in software testing, where they generate authentic text inputs based on the current UI page information,

[1]https://github.com/vuminhduc796/GPTVoiceTasker

replacing the conventional random text input approach [48]. This demonstrates the transformative potential of LLMs in advancing research and innovation across a multitude of domains.

The capabilities of LLMs have sparked a surge in their application within assistive technology, revolutionizing the translation of user commands into executable tasks across diverse systems. Recent research in this domain has witnessed the transformation of human natural language commands into various types of tasks, including visualization tasks [69], operating system tasks [47], and robotic tasks [43, 64]. LLMs have enabled these systems to tackle more intricate commands beyond the scope of existing heuristic approaches. They also exhibit a remarkable ability to comprehend variations of commands that share similar intentions but are expressed differently. This pioneering framework, with the support of LLMs, paves the way for a novel (semi)automated task execution paradigm, erasing the boundaries between traditional command patterns and intuitive command modalities.

Notable advancements include the World of Bits platform, which enables web-based agent training through interactions with real-world websites using low-level actions. WoB utilizes reinforcement learning and behavioral cloning to demonstrate these techniques' potential in web-based tasks, ensuring reproducibility through cached HTTP traffic [62]. Additionally, Mind2Web pushes the boundaries of generalist web agents by leveraging LLMs to handle complex, open-ended tasks across a wide range of real-world websites, showcasing the capacity of LLMs to generalize across diverse web environments [18]. Generative agents further highlight the application of LLMs in simulating human behavior in interactive settings, providing a framework for more dynamic and lifelike simulations [56]. Lastly, the design framework involving cells, generators, and lenses aims to optimize object-oriented interactions with LLMs, enhancing usability and functionality in various applications [33].

Within the domain of mobile assistants, LLMs have become a transformative force, overtaking traditional machine learning models as demonstrated in previous works [54, 67]. This shift has simplified the translation of natural voice commands into actions on mobile UIs. Wang et al. [68] used LLMs for conversation-like interactions with mobile UIs, showcasing a superior understanding of on-screen elements compared to earlier machine learning methods [42]. However, their approach only focuses on single-screen support and interactions, which is inadequate for completing multi-step tasks. AutoDroid [72] and AutoTask [53] have employed LLMs, incorporating a degree of application knowledge (e.g., recalling previous actions or repeating similar commands) to execute multi-step tasks through a single command. that can complete multi-steps tasks under one command. Yet, these methods have tended to concentrate on discrete tasks without fully addressing the continuity between tasks within the same application. `GptVoiceTasker` advances this field by collecting sophisticated domain knowledge and employing advanced prompt techniques, aimed at improving precision and establishing a more advanced smartphone virtual assistant. This enhancement allows users to execute both familiar and novel tasks more effectively on their devices.

3 THE GPTVOICETASKER SYSTEM

We introduce `GptVoiceTasker`, a virtual assistant that empowers users to efficiently perform multi-step tasks on their smartphones using voice commands. Upon receiving a user command, `GptVoiceTasker` first attempts to streamline the task using the collected in-app navigation database (Section 3.2), to improve execution efficiency and reliability. If a task is unprecedented and not in our saved records, `GptVoiceTasker` will perform a series of step-by-step predictions of the on-screen navigation sequence to complete the task (Section 3.1). Simultaneously, the system expands the database with new in-app navigation knowledge for subsequent autonomous task execution.

3.1 Unprecedented Task Exploration

Upon receiving an unprecedented task from the user, `GptVoiceTasker` will progressively predict and automatically execute each step until the task is accomplished. For each step, we collect contextual data from the mobile UI, task execution context, and current application information, which is combined with system-level information. `GptVoiceTasker` constructs all relevant data into prompts in a specific format and feeds them to the LLMs to determine the appropriate action on the user's smartphone. Upon receiving the response from the LLMs, `GptVoiceTasker` executes the action on the smartphone accordingly.

3.1.1 Data Collection Module. As LLMs execute logical reasoning based on textual inputs, known as prompts, a detailed and comprehensive prompt aids LLMs in understanding the task at hand and generating appropriate responses. Therefore, our primary focus is to incorporate sufficient information into our prompts to ensure accurate decisions from LLMs. This critical information, which we define as knowledge, is categorized into User Interface (UI) knowledge, Task knowledge, Application knowledge, and System knowledge. We extract this information through static analysis of the smartphone and its applications.

UI knowledge. Information about the on-screen UI elements is a major component in our prompts, as it allows LLMs to comprehend the content currently displayed. Our primary emphasis is on representing smartphone GUIs in a textual format that can be interpreted by LLMs through text-based input. While recent research has proposed converting the UI elements list to HTML format [27, 68] to reduce the prompt length, this approach becomes less relevant as LLMs now have relaxed restrictions on the number of tokens in a prompt. Therefore, we propose a more comprehensive view of the hierarchical structure of smartphone GUIs to improve the decision accuracy of LLMs. We represent each screen as a tree of nodes, with non-leaf nodes representing UI containers and leaf nodes representing visible UI elements. For each UI element, we collect the element type, text label, and append it with a unique ID. For some element types, such as buttons or text fields, the label can be extracted directly from the screen. However, for certain graphical UI elements like icons or image buttons, such information is not readily available. A potential solution is to apply deep learning models to predict the potential label of icons [14]. Nevertheless, this approach can cause excessive overhead on app pages that include multiple images and icons, significantly impacting the responsiveness of real-time assistants. Alternatively, we propose a lightweight

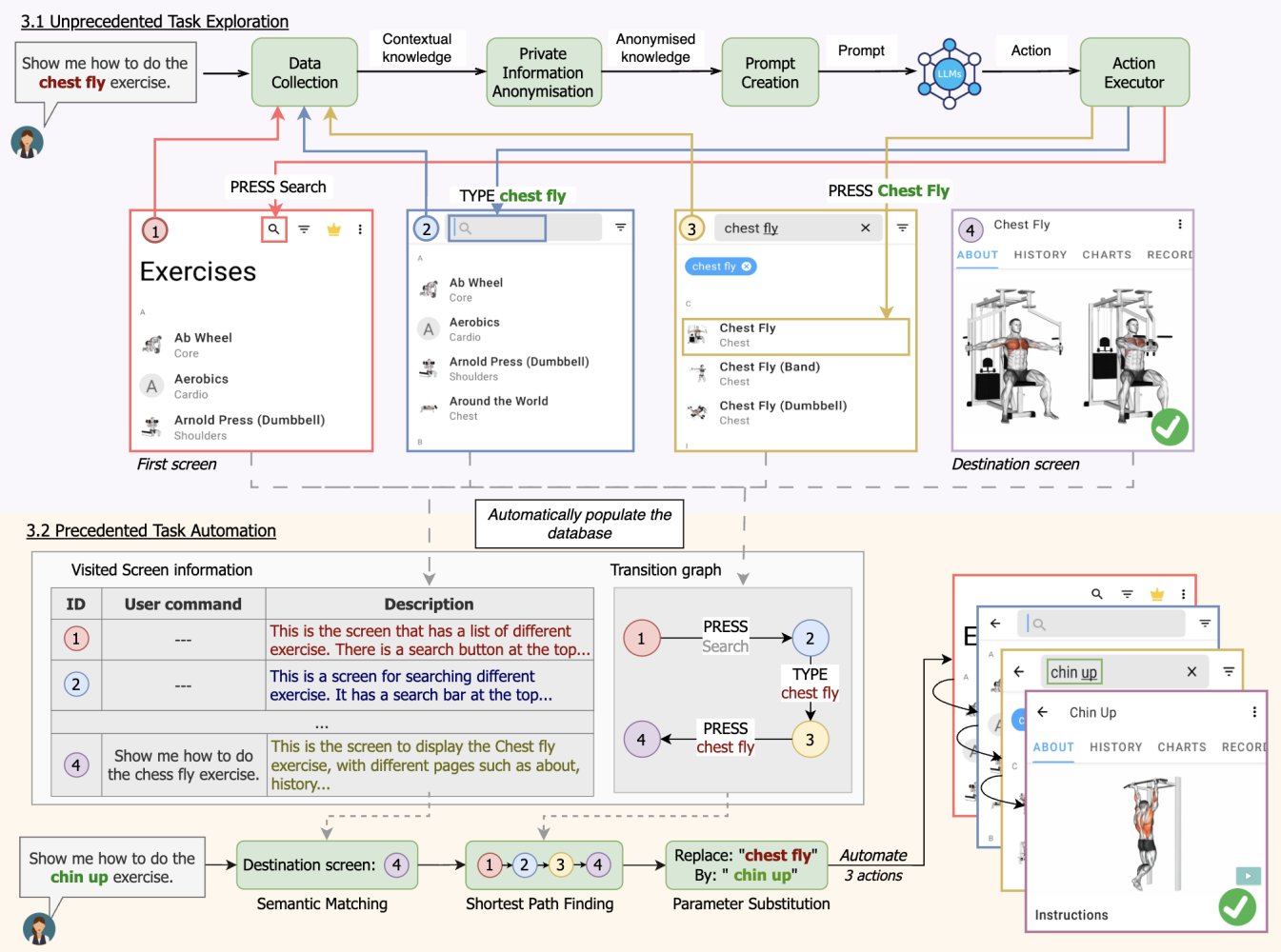

Figure 2: An example use case in Home Workout application when the user needs to interact with the smartphone hands-free due to physical busyness. When performing an unprecedented tasks (Section 3.1), `GptVoiceTasker` repeatedly predicts on-screen actions with current UI information and executes the response to achieve user tasks. The interactions collected during this process is then saved to streamline the execution of subsequent similar tasks (Section 3.2).

approach to collect alternative captions and resource names of these elements as labels, as they often include informative descriptions. For example, the search icon in screen 1 of Figure. 2 has the resource name *"ic_search"*, defined by app developers, indicating that this button is used for the search functionality. Furthermore, we collect the precise element location on the screen to cater to commands that refer to UI elements by their locations, such as *"Press the icon at the top-right corner"*. We also retrieve the list of allowed actions for each element, which can include CLICKABLE, TEXT_EDITABLE, SCROLLABLE, etc. This knowledge acts as a guardrail to ensure that LLMs do not return unsupported actions, such as pressing a disabled button. The runtime UI elements may contain UI noise, which is a prevalent issue linked to the real-time gathering of UI elements [40]. This problem arises when the collected UI information does not align with its visual representation, which affects the semantic understanding of LLMs on current UI elements, resulting in incorrect interactions. To address this, we implement heuristics to mitigate potential inaccuracies and ensure the reliability of collected UI information. First, we utilise the collected coordination of each UI element to eliminate out-of-bound or empty elements. We also eliminate views are fully overlapped by other views, which does are invisible and not interactible. In addition, we remove those views that do not contain any interpretable information, such as empty view containers.

Task knowledge. As complex tasks on smartphones involve multiple steps, treating each step as a separate action may lead to execution inaccuracies due to the misalignment of sequential actions. Additionally, the system may reattempt a single action multiple times, potentially causing an endless execution loop. To address this, we maintain information about the currently executing task and include it in our prompts. The task knowledge in each prompt specifies the user's request, the previously executed actions,

and the visited pages. Such knowledge is incorporated into our prompts as natural language in a predefined template (e.g., *"Started from <page 1>, we <action 1> <target 1> to get to <page 2>. After that, we <action 2> <target 2> to get to <page 3>."*), helping LLMs to comprehend the previous actions and better align the subsequent actions accordingly.

Application knowledge. Our experiment shows that LLMs can incorporate the knowledge about smartphone apps to output better results. This is particularly helpful for popular apps as LLMs are more familiar with these apps in their training data. In addition, these information helps validating each step in multi-step executions. For example, if the action caused unintended navigation to another app, LLMs can utilise the current app name to know this changes and output appropriate action to navigate back to the requested app. Therefore, we provide the app name, package name and list of activities as the application knowledge in our prompt.

System knowledge. To enhance the contextual understanding and reasoning capabilities of the LLMs, we not only fetch on-screen data but also retrieve relevant system-related information. We collect resolution information, which identifies the current screen orientation, and include it in our prompts. In addition, the screen resolution helps in the pre-processing of collected UI elements, such as validating the position of UI elements for out-of-bounds checking.

3.1.2 Private Information Anonymisation. Smartphones often contain and display personal data, including phone numbers, addresses, and payment details. Anonymizing this data is crucial when inputting UI screen content into LLMs to prevent exposing sensitive user information. To this end, we have integrated a lightweight Named Entity Recognition (NER) to efficiently searches and categorizes textual information within the UI elements. Upon detecting private information, we substitute it with standardized tags, such as <address> or <phone number>, before the data is processed by the LLMs. This approach not only preserves user privacy but also ensures that LLMs understand the UI semantics accurately to generate appropriate responses.

3.1.3 Prompt Creation. We include the user command with collected data in the previous step in our prompt to predict the most suitable action to perform on user smartphone. As the naive approach to prompt LLMs for various tasks may yield suboptimal results due to low accuracy and randomness in responses [17], we adopt of different prompt engineering techniques. These techniques involve crafting prompts according to specific rules and components to elicit optimal responses from LLMs [46]. Following Least-to-most Prompting strategy [81], `GptVoiceTasker` implements a two-step prompting approach to determine the action. First, we map the user task to a specific action (e.g., tapping an element, entering text, scrolling). Subsequently, based on the determined action, subsequent prompts are sent to identify the target UI element for executing that action (as illustrated as an example Figure. 3). This approach allows breaking down a complex task into smaller steps, enabling the LLMs to improve accuracy.

To empower LLMs to facilitate rapid comprehension of specific rules and guidelines, we employed the Few-shot Prompting approach [10], where we provide additional examplars in our prompt. Each examplar contains the sample prompt as the input with its corresponding response as the expected output. In addition, we integrate "Chain of Thought" [71] into our few-shot exemplars to help LLMs simulate human-like reasoning and provide logical output. This includes a sentence that explains the logic behind its output, which guides the LLM to apply a similar thought process in handling tasks. We dynamically calculate the number of examplars in a prompt based on the estimation of tokens used to include the knowledge. This approach maximizes the number of exemplars, thereby enhancing the accuracy of the response, while ensuring we do not exceed the token limit. In few-shot exemplar 1, as in Figure. 3, we provided a chain of thought that explains why Library button should be pressed to find the video history in YouTube.

3.1.4 Action Executor. `GptVoiceTasker` extracts the action and target from LLMs response to perform the interactions on user's device, such as tapping, scrolling, or entering text on certain UI element. Prior mobile automation approaches [54] encounter challenges in handling runtime UI changes and app updates that might alter UI representations and operation sequences. To improve from previous approaches, `GptVoiceTasker` dynamically propose the action based on the current UI, ensuring reliable navigation through dynamic UI changes and updates. Additionally, `GptVoiceTasker` provides audio feedback to users, confirming that the system is automatically proceeding to the next command. This real-time feedback ensures a smooth and intuitive user experience with `GptVoiceTasker`'s interaction capabilities.

To address challenges in real-time system automation, including failure detection in automated steps [74], our implementation integrates a screen transition detector. This detector employs Hamming distance [35] to measure differences between screens, ensuring screen content changes due to the action. Moreover, to validate the success of actions, we implement additional heuristics. For example, in scenarios involving ENTER_TEXT, we verify the presence of the entered text in the target text box. If an action proves inexecutable (i.e., not inducing appropriate changes to the UI), we repeat the step with supplementary information about the failed interaction attempts, thereby excluding these actions from the selection. Another challenge in mobile app automation is the dependency of UI screens on asynchronous internet content. Providing LLMs with incomplete or loading UI instead of fully rendered screen content may reduce the precision in identifying appropriate actions. In response, `GptVoiceTasker` delays UI collection until the screen is fully loaded. This is achieved by detecting screen-loading widgets in Android, leveraging information such as widget names, types, and shapes to identify progress bars and loading indicators. Additionally, we enhance our approach by collecting data on network transmissions to identify ongoing content download tasks, inspired by the methodology presented in [45], which leveraged network analysis for detecting ads. These methods enhance the reliability of `GptVoiceTasker` in effectively navigating through the app's interface.

3.2 Precedented Task Automation

In mobile apps, UI elements on a specific app page and in-app navigation paths are predefined by developers during app development. Therefore, the series of user interactions on the screen to complete a task in one app remains consistent over different instances. Based

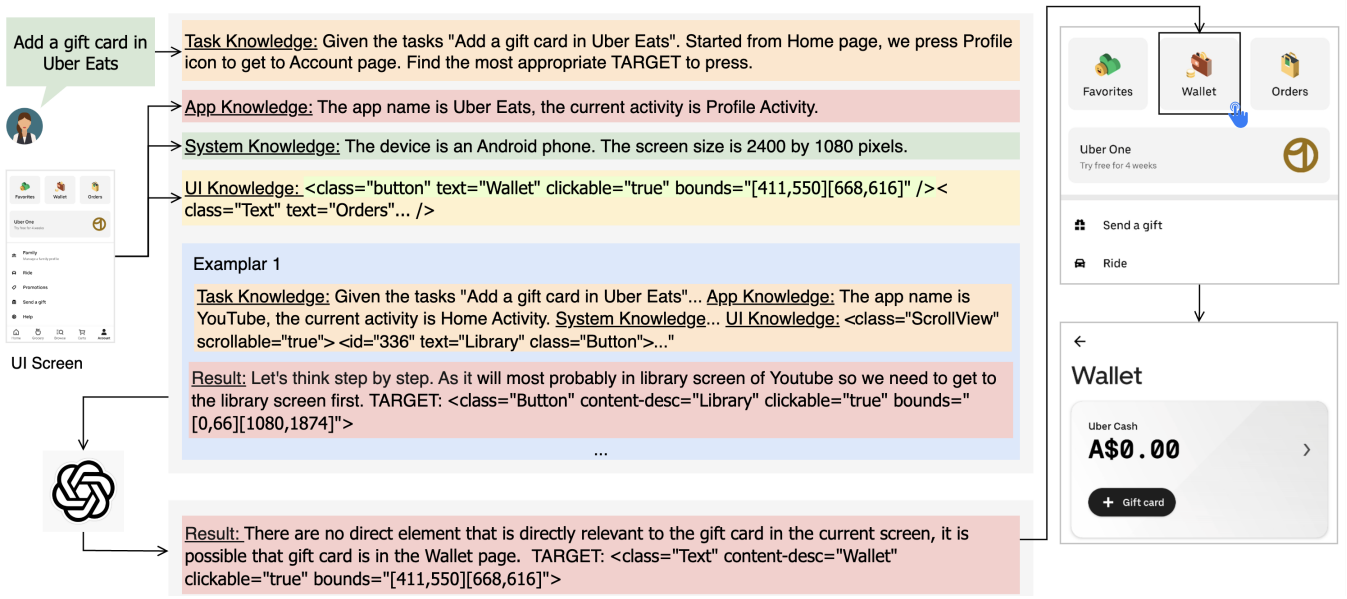

Figure 3: An example of our prompt and response format to determine the most relevant target to press.

on this, GptVoiceTasker automatically creates a saved path for each user request, enabling GptVoiceTasker to replicate interactions when receiving the same or similar commands from users. Unlike previous approaches [41, 54], which depend on manual task creation for automation support, GptVoiceTasker automatically records in-app navigation through on-screen interactions as described in Section 3.1. This automated process not only expands the coverage of automated tasks but also eliminates the need for manual efforts in pre-defining shortcut tasks. In this section, we outline our method (as shown in Figure. 4) to streamline subsequent similar tasks from users, combining both LLMs-based and heuristic-based modules. We introduce our database design in Section 3.2.1. Initially, we identify the current UI screen displayed on the user's device and the destination screen (Section 3.2.2). We then find the most viable path from the current screen to the destination screen (Section 3.2.3). Finally, we incorporate human interactive feedback to validate and fine-tune the execution for future use (Section 3.2.4).

3.2.1 Transition Graph. In alignment with previous approaches for automating tasks in mobile apps [16], GptVoiceTasker utilizes a directed graph for each app to enable seamless transitions between different app pages. Each node in the database corresponds to a UI page in the app, containing a unique ID and a screen description, as detailed in Section 3.2.2. Additionally, for each node, we record the list of previous user requests that concluded on this page, as these indicate the functions served by the page. The directed edges between nodes signify possible navigation paths from one UI page to another. These edges hold details about the required actions and target UI elements for page transitions. This data enables GptVoiceTasker to perform specific actions on the identified targets, thus replicating user actions to facilitate navigation between screens. This graph expands automatically to include new UI pages and associated page transitions as users interact with their smartphones.

3.2.2 Screen Description & Command Pattern Matching. Previous task automation approaches typically start from an application's launcher page [3, 54], a method that falls short in real-world scenarios, especially when users request voice assistants to complete ongoing tasks. To address this, GptVoiceTasker enables task automation from any page of the app, aiming to fulfill any new or ongoing tasks from users. We first identify the node that represent the current app page in our graph database. However, identifying a page by performing string matching on the UI content presents substantial practical challenges. This difficulty arises as a specific application page can display dynamic content. For example, consider the search result page in the Uber Eats app, where distinct restaurants are displayed for *"spaghetti"* and *"sushi"* search terms. Despite featuring different UI content as they display different restaurants, these pages share the same layout and are identified as the same node in our graph database. To compare between app pages, GptVoiceTasker leverages the capabilities of LLMs to semantically summarize the UI content into a semi-structured description in natural language. The description includes a short paragraph describing overall functionality that the screen serves based on the UI elements, current activity name and app name. In addition, GptVoiceTasker appends the list of interactive elements, including clickable, scrollable, and text-editable elements to the description. To locate the current UI page among previously visited pages, GptVoiceTasker uses LLMs for semantic matching between the screen's description and existing screen descriptions in the database. As LLMs are proficient in logical tasks on natural language, the response allows us to identify the node in the graph database.

After identifying the current screen, we perform semantic matching to find the destination screen. We define the destination screen

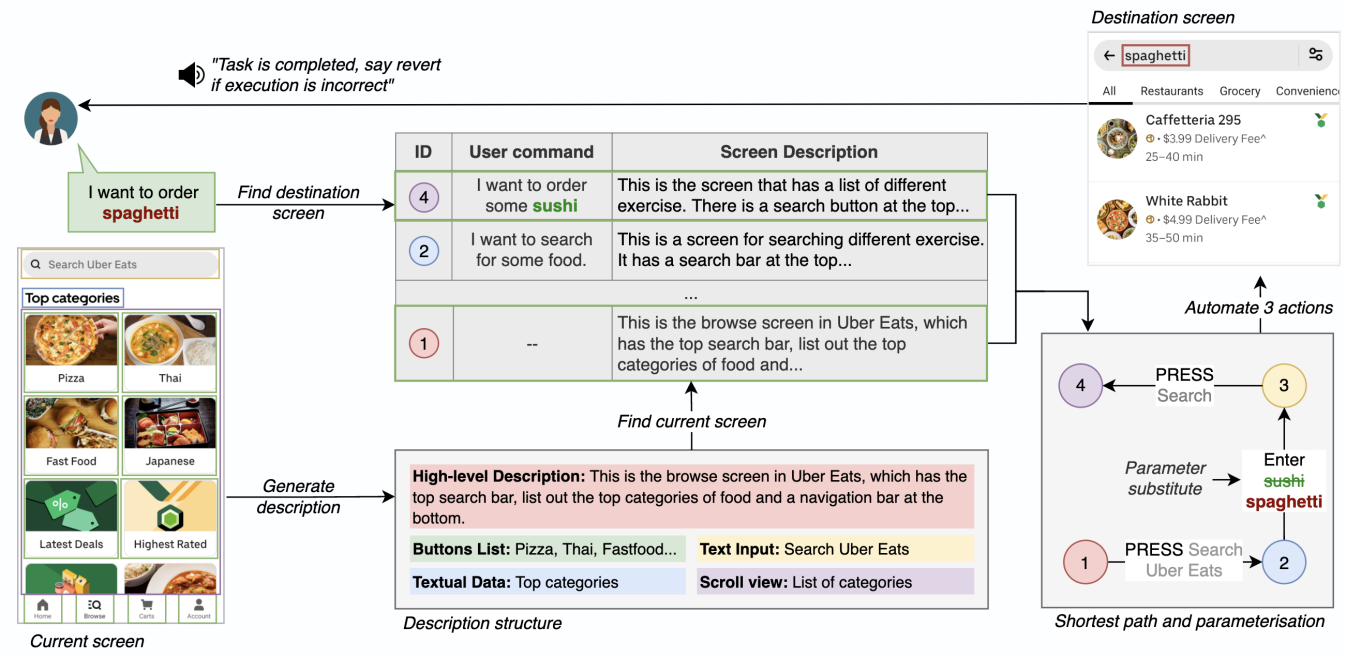

Figure 4: An example use case in Uber Eats to how `GptVoiceTasker` use the historical tasks to execute user new command. The system first locate the current screen and destination screen from the collected graph. After that, it identifies and execute the action sequence to traverse to the destination screen. Finally, we utilise feedback from users to improve subsequent execution.

as the most relevant screen that can serve user requests. For example, the destination screen for user command *"I want to order spaghetti"* is the search result page for the keyword *"spaghetti"*, where users can view the list of spaghetti restaurants. We utilise the saved commands and screen description for each app page to prompt LLMs to determine the most relevant page from the app. We build a prompt that includes user requests and the list of saved screens in the app to rank the relevancy of each app screen with the request that user made.

3.2.3 Path Finding & Execution. After identifying the current and destination pages within the saved graph database, `GptVoiceTasker` uses the Shortest Path algorithm [29] to determine the sequence of actions required to navigate from the start to the destination node. `GptVoiceTasker` extracts this sequence from the edges connecting the start node to the destination node and executes each action in sequence using the Action Executor described in Section 3.1.4.

Given the dynamic nature of smartphone GUIs, where the relative coordinates of buttons may vary due to scrollable screens or unexpected pop-ups and ads, execution validation is crucial. To ensure automation follows the expected sequence, `GptVoiceTasker` performs validation after each action. We generate the description for the current page (as explained in Section 3.2.2) and compare it with the description in the anticipated node. If the two descriptions do not semantically match, `GptVoiceTasker` reverts the preceding action and seeks an alternative path to the destination page. If no other path is found, `GptVoiceTasker` predicts and executes subsequent steps using the On-screen Interaction Module to explore a new path. This iterative process adapts to changes in the smartphone GUI, dynamically executing actions based on real-time screen information.

Additionally, `GptVoiceTasker` leverages command parameterization techniques [37, 54], allowing saved paths to be reused for similar tasks with different parameters. Using the LLMs' enriched vocabulary and robust natural language understanding, `GptVoiceTasker` prompts the LLMs to function as an advanced Named Entity Recognition system. This system identifies and replaces substitutable words with new keywords in the action sequence. For instance, in the Uber Eats app scenario (Figure. 4), a saved command for *I want to order some sushi"* can be adapted by replacing *sushi"* with *spaghetti"*, thus modifying the command from ENTER *sushi"* to ENTER *"spaghetti"*. `GptVoiceTasker` then executes this adapted sequence to complete the task.

3.2.4 Human Feedback Loop. One common issue with UI automation tools is dealing with changes in UI elements and in-app navigation due to new version updates by app developers. Additionally, pop-up ads may dynamically appear during the automation process. These challenges make predefined actions, such as clicking on an anticipated screen element becomes infeasible. To enhance the reliability of the saved paths, `GptVoiceTasker` incorporates a human-in-the-loop approach, modifying execution paths based on human feedback. After completing an autonomous task, `GptVoiceTasker` uses user feedback to validate whether the task was successfully executed. If users are unsatisfied with the automation, they can provide feedback in natural language, specifying where the issue occurs. This valuable information is saved and included in our

prompts, resulting in improved accuracy for future executions. Additionally, we provide an interface for users to manually amend saved commands, allowing them to customize or shorten their commands to trigger certain automation according to their preferences. Through user feedback and iterative learning, `GptVoiceTasker` facilitates human-AI collaboration in the automation process, enhancing LLMs decision-making and personalizing user experiences.

3.3 Implementation

We developed `GptVoiceTasker` as an Android application, leveraging the Accessibility Service provided by the Android OS and programming it in Java [22]. Specifically, `GptVoiceTasker` is designed to subscribe to the *typeWindowContentChanged* accessibility events [22], enabling it to detect and respond to changes in the UI on the screen. We engineered a dynamic pipeline that systematically extracts UI elements and organizes them into a hierarchical structure. This is achieved by processing AccessibilityNodeInfo objects [21], which are Android's data representations of UI elements accessible through the Android Accessibility Service. Additionally, `GptVoiceTasker` gathers application-level information using the Android PackageManager class.

For the integration of a Large Language Model, we selected GPT-4, the most advanced model developed by OpenAI at the time of this research [51]. For personalized services, such as screen descriptions and transition graph data, we store this information locally on the device for efficient future access. We made `GptVoiceTasker` and the prompt templates[2] publicly available at the GitHub repository[3] for further research in this field.

4 TECHNICAL EVALUATION

To evaluate the effectiveness and reliability of the proposed system, we conducted three experiments on our command interpreting module, unprecedented tasks exploration and usage-based execution. Specifically, we first assess the system's ability to comprehend user commands and perform on-screen interactions, comparing its performance to other state-of-the-art approaches. In addition, we experiment the ability of `GptVoiceTasker` to explore the execution path for unseen multi-step tasks. Lastly, we investigate the system's capability to execute multi-step tasks based on the saved user usages.

4.1 On-screen Interaction Evaluation

4.1.1 Experiment Setup & Metric. **Datasets:** We collect a specialised test set to evaluate our system's capabilities in understanding natural language commands and mapping them to appropriate actions and target UI elements. This dataset comprises 278 natural language user commands to interact with on-screen Android UI elements.

Although prior research [11, 67] has produced a similar test set, it is not directly adaptable to our context for two critical reasons. First, some instances in the test set are artificially synthesized based on predetermined heuristic rules. The resulting natural language commands are linguistically biased toward simpler linguistic patterns and do not align with the complex linguistic variants inherent in real-world human spoken utterances. Second, some test examples in the existing dataset have become obsolete or are no longer replicable due to updates in the corresponding applications.

To construct a test set that more closely aligns with real-world user interactions, we adopted a data-driven approach. We engaged 31 participants (17 females, 14 males), with 4 individuals having never utilized voice assistants before, 4 using them 3-4 times a week, 6 using them daily, and 17 using them less than 3-4 times a week. All participants are work professionals and university students who use smartphones daily. We provided these participants with screenshots alongside a specific task to accomplish. We then recorded the verbal commands they issued to their mobile device to complete the given task. After the collection, we annotated the commands to specify the intended action and target UI elements within the Android system; here, the term *action* refers to executable functions, while *target* denotes specific UI components or elements on the current screen. As a result, we collected 278 natural user commands for the dataset.

Metrics: Similar to Vu et al. [67], we adopt three evaluation metrics, namely *Exact Match Accuracy* (EM), *Target F1* and *Action F1*. EM calculates the percentage of instances in the test set where the predicted sequence exactly matches its corresponding ground-truth sequence. The measures *Target F1* and *Action F1* quantify the average micro F1 score for the target (i.e., the UI components to be interacted with) and the action (i.e., the actions to be performed on the UI components), respectively. The F1 score for each instance is computed using the formula:

$$\text{F1} = \frac{2 \times |\text{pred} \cap \text{gold}|}{|\text{pred}| + |\text{gold}|}$$

where |pred| represents the size of the set of predicted targets or actions, and |gold| denotes the size of the set of ground-truth targets or actions. The average micro F1 score is calculated across all instances for either targets or actions.

Baselines: We consider five baselines for converting natural language into semantic meaning representations, which comprise actions and targets. These baselines are **vanilla Seq2Seq** [5], **BERT-LSTM** [75], **Voicify Parser** [67], the **`GptVoiceTasker` w Base-Prompt** model, and **Wang et al.**'s work [68]. In the original work by Voicify, all three baselines employ deep learning models trained on datasets synthesized using the Overnight method [70]. This method generates training sets based on predefined lists of actions and targets that are designed for evaluation scenarios in Voicify. To ensure a fair comparison, we modified these lists to include the actions and targets present in our test dataset. We then re-synthesize the training set, which includes 1,384 instances, using the Overnight method, adhering to the implementation outlined in Voicify's work. The `GptVoiceTasker` with base prompt is used as an ablation study to the current prompt design of `GptVoiceTasker`. The base prompt contains only a minimal explanation of the task and the UI knowledge (XML file) of the current mobile screen. Wang et al.'s work [68] was the first to incorporate LLMs for interacting with mobile interfaces. We incorporated the 2-shots LLM prompts they demonstrated in the paper for mapping instruction to UI actions. Additionally, we have enhanced their model by integrating the more advanced GPT-4, which also inline with the one we used in `GptVoiceTasker`.

[2] https://github.com/vuminhduc796/GPTVoiceTasker/blob/main/prompts.txt
[3] https://github.com/vuminhduc796/GPTVoiceTasker

Table 1: The experiment results of different baselines compared with GptVoiceTasker in three metrics.

Models	EM Accuracy (%)	Action F1 (%)	Target F1 (%)
Seq2Seq	25.2	47.6	35.6
BERT-LSTM	41.4	59.7	57.3
VoicifyParser	47.5	64.0	58.8
GptVoiceTasker *w BasePrompt*	58.4	71.6	62.4
Wang et al. [68]	79.9	85.4	83.4
GptVoiceTasker	**84.7**	**91.7**	**84.7**

4.1.2 Evaluation Result. Table 1 presents the results of our technical experiments. Overall, GptVoiceTasker outperforms all baseline models across all metrics, achieving an **84.7%** EM accuracy, a **91.7%** Action F1 score, and a **84.7%** Target F1 score. Among the baselines without the LLMs, the Voicify Parser performs the best, aligning with the results reported in its original paper [67]. However, its performance suffers when faced with linguistic variations in our new test set. For instance, while the command "*back*" is correctly interpreted as "*(PRESS , back)*", the phrase "*return to last page*", which represents the same command, is incorrectly parsed as "*(SWIPE , DOWN)*". Both BERT-LSTM and Seq2Seq models encounter similar issues, largely because they share architectural and training similarities with the Voicify Parser, yet perform even worse due to Voicify Parser being specifically optimized for task completion on Android systems. Despite the lack of prompt design, GptVoiceTasker with a base prompt achieved better performance than the other three DL-based baselines. However, when compared with GptVoiceTasker, the results indicate that incorporating a multi-level knowledge-based prompt design, along with few-shot learning and the Chain of Thought technique within GptVoiceTasker, can enhance the accuracy of converting natural language to on-screen actions and the corresponding elements. The method by Wang et al. [68] demonstrates the highest capability among the baselines, courtesy of the LLM's intervention, effectively rectifying the errors previously noted. However, the lack of the chain-of-thought and least-to-most prompt techniques occasionally leads to inaccuracies. This is evident in instances where the system misinterprets the intended direction in commands, such as confusing *DOWN* with *UP*, or when it cannot adequately differentiate between actions like *PRESS ENTER* or *OPEN* when various verbs are employed in the commands.

Benefiting from the integration of LLMs and the prompting techniques, GptVoiceTasker excels at handling linguistic variants, consistently deriving the intended action and target regardless of variations in the input. The Action F1 score for GptVoiceTasker reaches 91.7, indicating its enhanced ability to predict actions across various linguistic patterns. Moreover, we observed that LLMs effectively learns the true associations between actions and targets, thereby excelling at target prediction as well. For instance, *PRESS* is exclusively predicted with UI buttons, *ENTER_TEXT* is linked solely with text input fields, and *OPEN* corresponds to app names. In contrast, the baselines often learns incorrect associations and outputs wrong target predictions. Our experimental results show that GptVoiceTasker is effective in understanding and accurately processing different linguistic variations, demonstrating its adaptability in real-world scenarios.

4.2 Multi-step Execution Evaluation

4.2.1 Experimental Setup & Metrics. This experiment assesses the Unprecedented Task Exploration module's capability to execute unseen tasks. We evaluated the module's performance using the most recent human-collected demonstrations and natural language instructions from the Android-in-the-wild dataset [60]. This dataset provides step-by-step on-screen interactions to complete tasks based on natural language instructions, mirroring real-world commands. From this dataset, we randomly sample the data from multi-step Google Apps subset and manually validate each pair to get 140 test cases, with each test case has from 1 to 15 steps (M=6.705, SD=2.764). We treat the end screen after the final action in the action sequence from the dataset as the ground truth, indicating successful command execution. We did not evaluate the accuracy of each steps to the dataset as one task could be performed by multiple approach. For each test case, we have GptVoiceTasker iteratively explore the path to complete the instruction, comparing the destination screen achieved by GptVoiceTasker with the dataset's ground truth. A test case was deemed successful if GptVoiceTasker reached the same screen as it is in the ground truth with no more than three additional steps than the dataset demonstration. Cases where the step count was exceeded or the next step could not be identified were marked unsuccessful. GptVoiceTasker executed commands solely using its task exploration module, without relying on a database for guidance. Similar to Sec 4.1, we used Wang et al. [68] approach as the baseline, which used a 2-shot prompting technique on the GPT-4 model, making iterative requests after each action response.

4.2.2 Results. GptVoiceTasker achieved an **85.7%** success rate (120 out of 140), outperforming the baseline approach by Wang et al., which achieved a **56.4%** success rate (79 out of 140). GptVoiceTasker succeeded in performing logical reasoning to execute the tasks. It utilizes the current app name and package name to determine if required apps need to be opened, and uses the list of run-time device-available app names to query the most suitable app for the task. Notably, GptVoiceTasker effectively navigated tasks without encountering cyclic navigation issues or repeating actions, such as repeatedly tapping the *Settings* title within the Settings app, which hindered task completion in the baseline. This improvement is attributed to the integration of historical messages and runtime execution error handling module. Our study also found that GptVoiceTasker does not rigidly adhere to the demonstrated paths presented in the Android-in-the-wild dataset for task completion. For example, in managing tasks within the Android Settings App, it occasionally opted to search directly for options rather than scrolling through menus to locate them. Other added constraints in GptVoiceTasker helped to improve the usability of multi-step interactions, such as validating if the current screen is scrollable or a text field is focused before inserting text, which were observed as reasons that caused failures in the baseline.

Further investigation into unsuccessful cases highlighted two main areas for improvement. First, GptVoiceTasker struggled with time-related tasks, such as *"Check the schedule for Friday next week"*, due to the LLM's limited knowledge of the current date and time. As a result, while GptVoiceTasker correctly opened the app, it was unable to select the Friday of next week to view the schedule. This could be mitigated by integrating temporal information into our

prompts. Second, GptVoiceTasker exceeded the number of steps when performing *"Open the settings page in Google Maps"*, where the immediate step required pressing the profile picture, which is not directly relevant to the task command. To address this issue, we could enrich the prompts with additional contextual knowledge, such as the steps involved in accessing settings in similar applications where pressing the profile picture is necessary. Additionally, incorporating a strategy of random exploration steps before repeating a command could help the system discover more direct paths to complete tasks.

4.3 Database Execution Evaluation

Table 2: Saved task execution evaluation result for direct match tasks and parameterised tasks across 5 categories.

Category	Average Number of Automated Steps	Success Rate (%)	
		Direct Match	Parameterised
Message Friends	4.33	93.33	86.67
Listen to Music	5.27	80.00	73.33
Set an Alarm	5.73	73.33	53.33
Check Weather	5.07	80.00	73.33
Get Directions & Map	5.53	86.67	73.33
Average	**5.19**	**82.67**	**72.00**

4.3.1 Experimental Setup & Metric. In this experiment, we assessed GptVoiceTasker's ability to automate tasks using the usage-based execution module. We initially identified the five common smartphone application categories, as shown in previous study [3]. Within each application category, we randomly selected five popular applications from the Google Play Store, with downloads ranging from 1 million to over 1 billion. For each selected app, we identified three features introduced by the developers in their Play Store descriptions. Each feature was then used to create both a direct match task, involving a straightforward match between user commands and corresponding app actions, and a parameterized task, requiring GptVoiceTasker to perform keyword substitutions to complete the task successfully, as shown in Section 3.2.3. For creating the direct match test cases, we paraphrased each saved command using state-of-the-art paraphrasing tool Quillbot[4], as in [63]. In the case of parameterized tasks, we substituted one entity in the paraphrased command with another entity that has similar semantic. For example, consider the saved task "*Get directions to the nearest supermarket*". In this case, the direct matching task would be "*Find the nearest supermarket's location*", while the parameterized task would involve substituting "*restaurant*" for "*supermarket*", resulting in "*Find the nearest restaurant's location*". This process resulted in a total of five app categories, each category contains 15 direct match tasks and 15 parameterised tasks. These tasks involve 4 to 7 steps, with an average of 5.19 steps per task as illustrated in Table 2. All tasks can be automated with one voice command with the saved user app usage patterns. For a detailed list of the apps and features used in the experiment, please refer to our GitHub repository[5].

[4]https://quillbot.com/

[5]https://github.com/vuminhduc796/GPTVoiceTasker/blob/main/Result.xlsx

To populate the transition graph and store screen descriptions, we manually navigated through each screen in every application using GptVoiceTasker. Subsequently, we configured the saved commands to reach the respective screens as the ground truth. We used the success rate as the primary metric, each test case is marked as success if GptVoiceTasker can successfully opened the desired feature using a single command.

4.3.2 Results. Table 2 illustrates the accuracy of our saved task execution modules. Our findings indicate that GptVoiceTasker achieved an impressive level of automation, successfully handling **82.7%** of exact match tasks and **72.0%** of parameterized tasks. Notably, GptVoiceTasker exhibited exceptional performance in tasks related to messaging and directions & maps applications. This success can be attributed to the relatively static nature of these apps, where user interfaces maintain a consistent structure. Our results underscore GptVoiceTasker's proficiency in command analysis, semantic matching to saved tasks, and parameterized phrase substitution within these contexts. However, the accuracy of GptVoiceTasker diminished when confronted with tasks related to setting alarms. To better understand the root causes of this decline in performance, we conducted an error analysis on the failed test cases. Several key issues emerged:

- *Complex Parameterized Tasks*: For parameterized tasks with additional steps, such as setting an alarm for 7:30 instead of 7:00, GptVoiceTasker struggled due to the extra step involved in selecting the minutes, which was on a separate UI element. Further works include making GptVoiceTasker adaptable to these additional steps in the automation process.
- *Pop-ups Ads and Unusual UI Elements*: Certain applications presented pop-ups ads and unusual UI elements in run time that were not encountered during the initial task-saving process. Consequently, GptVoiceTasker faced difficulties in completing these tasks. To improve the robustness of our approach, we recommend exploring the integration of a deep learning model to detect and handle such ad widgets and unusual UI elements, as in [25, 45].

5 USER STUDY

To demonstrate the practical utility of our tool, we conducted a user study to evaluate the holistic performance of the GptVoiceTasker system within real-world scenarios. Our evaluation involved a comparative analysis against two baseline systems: 1) Voice Access [76], the official voice assistant product developed by Google, with over 100 million downloads, and 2) Voicify [67], the state-of-the-art research product endeavor incorporating deep learning models to enhance command comprehension. This study pursued a threefold objective: i) establish a performance benchmark for user interactions utilizing the GptVoiceTasker system as opposed to the aforementioned baseline systems, ii) juxtapose user feedback concerning the cognitive load and overall usability of the GptVoiceTasker system against the baselines and iii) capture qualitative insights from participants, thus enabling the identification of potential avenues for enhancing the GptVoiceTasker system. In order to achieve these objectives, we recorded the task completion times for tasks undertaken using both the GptVoiceTasker system and the baselines. Furthermore, a comprehensive post-experiment interview

was conducted with each participant, facilitating the collection and analysis of both quantitative and qualitative feedback.

5.1 Tasks

We designed 6 experimental tasks, encompassing a broad spectrum of the most common interactions performed on the screen, ranging from tapping and swiping to entering text. Each task was structured to comprise between 6 to 10 sequential steps. The detailed list of these tasks is outlined in Table 3.

5.2 Participants

We recruited 18 participants, consisting of 10 males and 8 females, aged between 18 and 31 years old for our study. The mean age of participants was 25.83 years (SD = 4.26). The group included 8 bachelor students (from IT and Business fields), 4 master students (IT), and 6 PhD candidates. 8 participants are native English speaker while all other participants are proficient in English. All participants possess a commendable level of familiarity with technological devices and actively use smartphones in their daily routines.

We advertised our experiment on LinkedIn to recruit participants from our university. During recruitment, participants provided their gender, age, study level, English proficiency, tech proficiency, and experience with voice assistants. While participants exhibited exposure to virtual assistants like Siri or Google Assistant, none were acquainted with utilizing assistive tools for smartphone control via voice commands. Specifically, none of the participants had prior experience with any of the experimental tools employed in our study. This participant selection was deliberate, as our study sought to gauge the learnability aspect of the experimental tools. Each participant received a USD $30 gift card for the participation.

5.3 Procedure

We conducted face-to-face user evaluations using an Android device as the experimental tool. On this device, we had the graph of each experimental app populated, which include the majority of app pages and navigation within the app. At the start of the sessions, participants were introduced to all experimental tools via demonstrative videos. The preliminary phase involved practicing basic tasks across all tools, enhancing participants' familiarity with step-by-step instructions and informative walk-through videos. We also use the searching for exercise tasks in Figure. 2 as the practice tasks, allowing users to achieve this task using each of the tool.

After that, participants independently executed six distinct tasks with no experimenter intervention. Each tool was employed for the completion of two tasks, and participants remained unaware of which tool was developed by us. To mitigate any potential biases, the order of tasks and the tools used were systematically counterbalanced for each participant [19].

We applied a time cap of 60 seconds per step. We recorded the time taken to fulfil each task, including the cut-off time to perform quantitative analysis. We collected 108 data entries since each of the 18 participants has finished 6 tasks. In the end, using the System Usability Scale (SUS) [7] form with a 5-point Likert scale, we evaluate the usability of `GptVoiceTasker`, compared to Voice Access and Voicify. In addition, we investigated the cognitive load when experimenting with each tool using the NASA-TLX [30] form with a 7-point Likert scale. Lastly, we collected qualitative feedback on which part they liked the most about `GptVoiceTasker` and what might improve the system.

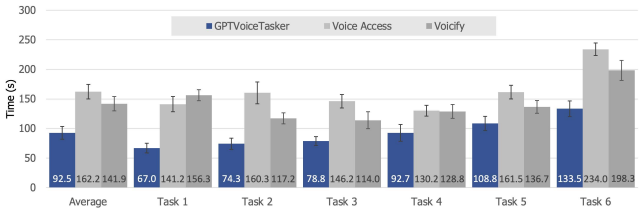

Figure 5: The average time taken to complete each task using `GptVoiceTasker` and the baselines in seconds.

5.4 Result

5.4.1 Overall User Performance. In Figure. 5, we present the average task completion times for each experimental tool. Our tool stands out with an average completion time of **92.5** seconds, surpassing Voice Access (162.2 seconds) and Voicify (141.9 seconds). This improvement in `GptVoiceTasker`'s performance can be attributed to two primary factors. Firstly, we can tell that `GptVoiceTasker` is better at comprehending user intentions and mapping user commands to the correct actions on specific UI elements, regardless of the command format. In contrast, baseline tools often demand specific command formats, introducing errors in various usages. This issue caused extra time costs as participants needed to seek different ways to express their intentions with the baseline tools. For example, participants tried to tap the option button, in the Notes app with Voice Access by multiple attempts such as "*press on the option button*", "*press the three-dot icons*", "*tap icon for options*" before successfully give the right command "*tap option*". Secondly, `GptVoiceTasker` optimizes the performance by automating several steps in one user command, as shown in Table 4. On average, the participants saved 2.2 steps across all six tasks. For instance, in Task 2, `GptVoiceTasker` efficiently automated the process of searching for Love Yourself song (as in Figure. 6(B)), drawing from a previously stored action designed for searching other songs. This eliminated the need for three steps required for in-app navigation. However, some participants did not realize that they could trigger the saved tasks, leading to a missed opportunity for a significant performance boost. In addition, `GptVoiceTasker` relates to network latency when sending and receiving data from the LLMs API endpoint. This issue could be mitigated with a better network connection.

5.4.2 Cognitive Load & Usability Ratings. Figure. 6(A) presents an overview of participant feedback regarding their cognitive load levels for each system, assessed using the NASA-TLX form. We conducted a Friedman test [61] for statistical analysis on the result. Participants reported decreased mental demand, temporal demand, and effort when using `GptVoiceTasker` in comparison to the baseline systems while achieving better performance. This result show an improvement in `GptVoiceTasker`'s ability to reduce the cognitive load required for operation, aligning with our design goal.

Table 3: The list of tasks for user evaluation.

No.	Task	#Steps	App Name	#Downloads
1	Check the weather within a particular city.	6	BOM Weather	1M+
2	Search for a specific song and play it.	6	Apple Music	100M+
3	Create a note and write "Hello world" and delete it.	8	Notes	1M+
4	Check for an unread message, reply with a message and delete the conversation.	8	Messages	1B+
5	Search for a pizza store, and complete the order.	10	Uber Eats	100M+
6	Create a new alarm and save it.	10	Challenges Alarm Clock	1M+

Table 4: Average number of automated steps by all participants in each task.

	Average	Task 1	Task 2	Task 3	Task 4	Task 5	Task 6
#Steps Automated	2.22	2.67	2.00	1.67	2.17	2.50	2.33

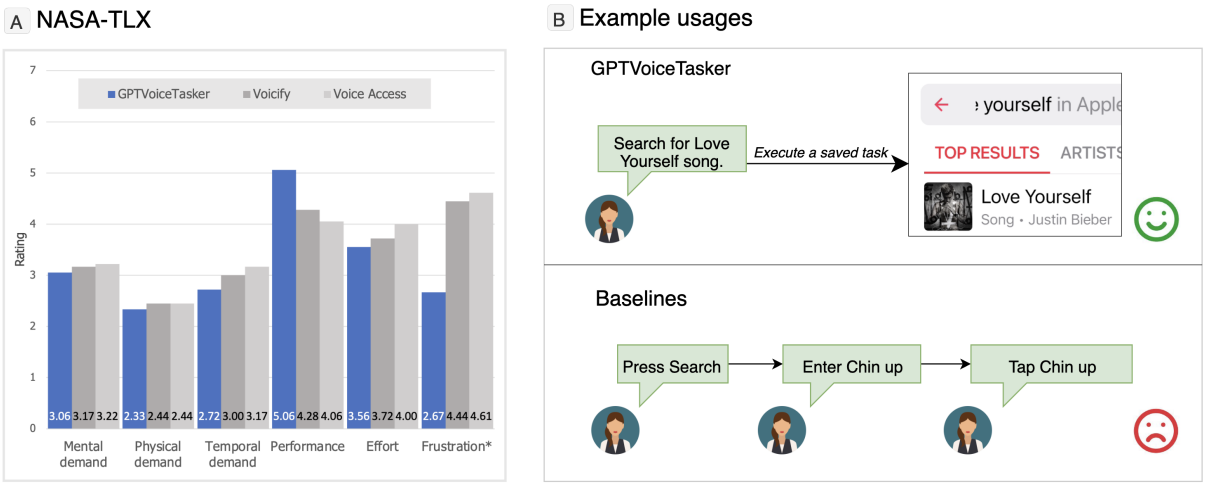

Figure 6: The comparison between `GptVoiceTasker`, Voicify, and Voice Access for A) the average cognitive load when using NASA-TLX form (lower is better) *: $p < 0.01$, **: $p < 0.001$ and B) Task 2 from the user evaluation with `GptVoiceTasker` and other baselines.

To assess `GptVoiceTasker`'s usability in comparison to the baseline systems, we employed Friedman test for statistical analysis on collected System Usability Scale (SUS) scores, as depicted in Figure. 7. The analysis verified the enhanced usability of the voice control system, with `GptVoiceTasker` achieving an average SUS score of 79.861, surpassing Voicify (47.917) and Voice Access (36.528). Participants found `GptVoiceTasker` less complex ($p < 0.001$), less inconsistent ($p < 0.001$) and well-integrated ($p < 0.001$), leading to more frequently use ($p < 0.001$). In addition, participants can learn to use `GptVoiceTasker` quickly ($p < 0.001$) as they do not need to learn a lot ($p < 0.001$). This remarkable outcome can be attributed to `GptVoiceTasker`'s ability to effortlessly comprehend natural human commands, reducing the need for extensive training and practice. The lower likelihood of misinterpreting user commands also contributed to the positive results.

5.4.3 Qualitative Feedback. In this section, we collate qualitative feedback from participants after the experiment. Overall, the participants are satisfied with the tool, as well as providing suggestions for further improvements.

Ability to precisely interpret and execute human command. Participants expressed enthusiasm about the remarkable ability of `GptVoiceTasker` to interpret human commands naturally, enhancing the overall system's intuitiveness. P1 and P12 highlighted that they could issue commands *"in their preferred manner"* and *"converse naturally"* with `GptVoiceTasker`. This addresses cognitive overload concerns, as P4 appreciated the *"stress-free experience"*, and P6 and P7 found `GptVoiceTasker` more *"comfortable to use"*. For instance, when adding a new note, users could simply say *"add a new note"* to prompt `GptVoiceTasker` to press the add button on the screen. Moreover, participants were impressed by our tool's accuracy in handling user input errors. P3 noted their satisfaction with how `GptVoiceTasker` *"can still execute the correct action even when I make mistakes in my commands"*. Both P4 and P17

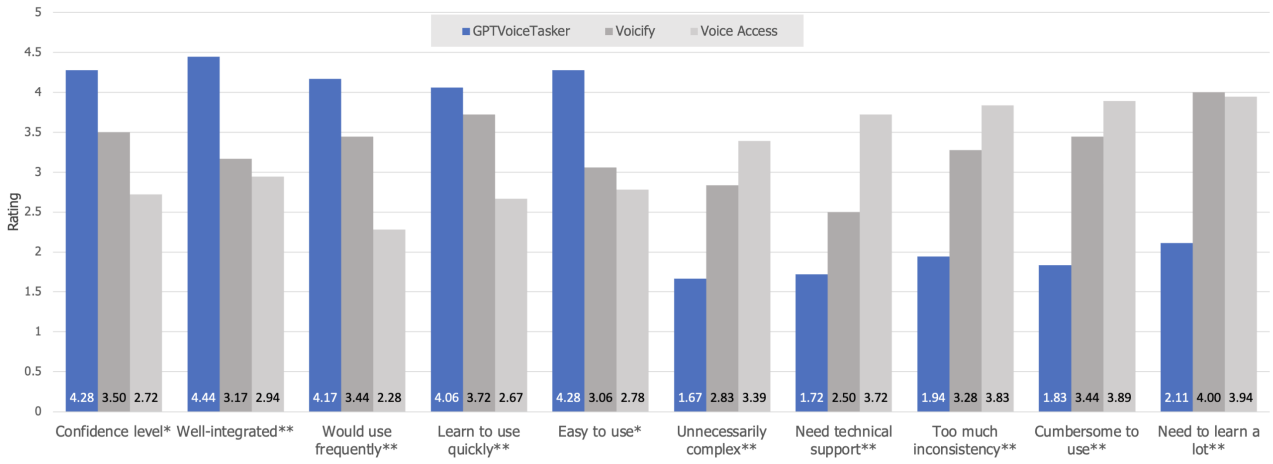

Figure 7: The comparison between GptVoiceTasker, Voicify, and Voice Access for the System Usability Scale (SUS). *: $p < 0.01$, **: $p < 0.001$.

highlighted the tool's usefulness in daily tasks, as it eliminates the need to *"exercise caution and stay alert"* when interacting with GptVoiceTasker. These feedback remarks strongly affirm the practicality of our approach in real-world task scenarios. In contrast, traditional approaches typically demand fixed input formats, making them ill-suited for real-world scenarios where user input can vary significantly.

Automated execution helps accelerate tasks and improve user experiences. Participants offered positive feedback regarding the use of saved task automation, highlighting its significant impact on efficiency and user experiences. P11 mentioned that this feature is *"accelerating the tasks"* while P13 emphasized the potential utility of GptVoiceTasker during physical activities, stating it would be *"really useful when I work out"*. P5 appreciated this feature, describing it as *"perfect for voice-interacting tools"*, as it mitigates the inherent challenges of voice command interactions. Additionally, P18 praised the feature, noting that tasks became *"fairly easy"* with its implementation, indicating significant performance improvements. This, combined with the advanced capability to understand user intentions, enhances the intuitiveness of voice-based interfaces. When using a smartphone, users often have a specific task in mind, such as setting an alarm or checking the news. Unlike other approaches that require users to perform additional steps to translate their intention into executable commands that a voice interface can understand and execute, GptVoiceTasker can directly execute these tasks without causing additional mental stress. However, users also provided valuable suggestions for enhancement. They expressed the desire for GptVoiceTasker to suggest executable saved tasks and display a list of saved tasks. Furthermore, participants suggested improving the introduction of this feature, as P4 noted it was *"not familiar at first"*, and P6 emphasized the need for *"better introduction."* These insights underscore opportunities to refine the feature's usability and user onboarding, ultimately enhancing overall user satisfaction.

Suggestions for enhancing user experience. Participants provided valuable suggestions for improving the intuitiveness of GptVoiceTasker. Regarding UI design, P14 recommended the inclusion of a *"live transcription"* feature to display recognized voice commands. This would help users confirm that their commands were correctly received and make necessary adjustments if needed. Furthermore, P1 and P15 suggested incorporating a *"loading indicator"* to signify ongoing executions, addressing latency issues caused by execution delays. In terms of functionality, P7 proposed displaying a list of available tasks as suggestions, enhancing user interaction. Additionally, P15 discussed the potential for an interface that allows users to modify saved tasks, providing greater customization. Lastly, participants P7 and P12 suggested making the audio feedback from GptVoiceTasker clearer. These suggestions hold significant value for GptVoiceTasker's continuous improvement, aiming to deliver a more seamless user experience.

6 DISCUSSION

We introduced GptVoiceTasker as an autonomous speech-based virtual assistants. In this section, we delve into the implications and limitations of GptVoiceTasker.

Towards the adoption of the voice-centric interface. The advancements in natural language understanding, particularly through LLMs like GPT and Bard [58], are propelling the transition towards voice-centric interfaces. These interfaces expand the capabilities smartphones to devices such as smartwatches, AR-VR headsets, and desktops, thereby becoming more integral to everyday activities. While visual-manual methods such as tapping on smartphones or mouse-clicking on desktops are preferred for their speed and accuracy, GptVoiceTasker leverages the power of LLMs to enhance the intuitiveness of voice interactions. This enables more intelligent mapping of user intentions to visual elements, facilitating the shift to voice-assisted interactions and promoting wider adoption of voice-centric interfaces. Additionally, the features introduced by GptVoiceTasker contribute to the domain of voice-centric research on other devices. For example, breaking down one task into a sequence of actions provides more flexibility than fixed intents (e.g., Firefox Voice [12]). The continuous learning from historical

usage helps these systems understand user commands better and personalize efficient experiences, such as prefilling patient information for Talk2Care [79] and providing personal shortcuts for Firefox Voice [12]. Moreover, the implementation of anonymization techniques addresses privacy concerns, particularly for voice-centric interfaces that access the display content from screens.

Voice-centric interfaces also improve accessibility for users with disabilities [80]. `GptVoiceTasker` helps individuals with motor and visual impairments by substituting touch-based interactions with voice commands. For motor impairments, this enables easier task completion on mobile devices without touching the screen. Additionally, `GptVoiceTasker` provides voice shortcuts for visually impaired users, allowing quicker navigation of familiar screens. For example, users can find specific buttons with commands instead of clicking through each one with Talkback. Natural interaction methods could also benefit individuals who struggle with technology and elderly users.

Despite the promise, challenges such as the effectiveness of voice recognition in diverse environments still persist. Addressing these will be crucial for the broader adoption of voice-centric interfaces, like smart homes and healthcare. This transition, while challenging, opens new avenues for user interaction and emphasizes the need for continued research in the HCI domain.

LLMs for task automation on user visual interfaces. Research has highlighted the capability of LLMs to provide reasoning based on the UI layout, applying to task automation and testing tools [27]. These models show remarkable capabilities in incorporating extensive knowledge concerning prevalent app design principles and recognizing standard mobile interface elements, including the toolbar, navigation drawer, and bottom navigation bar [49] to enhance proficiency in facilitating precise in-app navigation. Our study highlighted the vital role of spatial information and hierarchical UI representations for LLMs in comprehending semantic connections between diverse UI elements, particularly useful for elements lacking textual information like unlabeled icons or images. In our user study, when tasked with deleting a message lacking a visible delete button, LLM intelligently suggested initiating the process by pressing the unlabelled icon button at the top right, typically the location of the option button, and then selecting *"delete"* from the ensuing options list. The core of this research lies in the transformation of visual interfaces into textual descriptions that LLMs can process, a critical step for enabling effective task execution based on user inputs. Future research should address the models' limitations in unconventional UI scenarios and focus on expanding their adaptability across varied interface designs and complex user tasks. As models grow, visual models (e.g., GPT-4v) can process images to further enhance the accuracy of such interactions. However, optimizing the usage of visual models to balance accuracy and efficiency is crucial to compensate for the drop in response time, thereby improving their practical application in real-time scenarios. Such progress in LLM capabilities is pivotal for advancing user interface automation, leading to more user-friendly and efficient digital experiences.

Towards responsible AI in software systems. In recent years, the remarkable advancements in LLMs have enabled the seamless integration of AI into various software and systems. However, this integration raises significant concerns, particularly regarding data privacy and security [65]. The very nature of AI-integrated systems requires access to data, potentially putting sensitive or confidential information at risk. Put in the context of voice assistants on smartphones, users are sceptical as smartphones contain many personal and sensitive data [34]. Tools like `GptVoiceTasker` can read such on-screen data and further process them to LLMs. To mitigate these risks, it is essential to implement several key measures, not only to protect users but also to build trust, thereby fostering greater adoption of AI-based interactive systems. `GptVoiceTasker` represents a pioneering effort in voice-assistive research by applying personal information anonymization to protect user privacy when using LLMs for logical tasks. Additionally, when executing actions on behalf of users, voice assistants must operate responsibly, ensuring that actions do not adversely affect users. This involves seeking explicit user confirmation for decisions, particularly in scenarios where actions may have significant implications, such as replying to important emails or transferring money. Future work in this field should focus on identifying sensitive actions and prompting user confirmation while maintaining a seamless user experience.

Limitations. The current approach poses several limitations. Firstly, the usage-based execution relies prior usage in the particular application, therefore it is inapplicable to unused apps. To address this challenge, our future work aims to develop a more generalized approach to application usage, categorizing apps by their primary functions. For instance, we could devise a standardized set of steps for searching and playing a song that could be applicable across various music applications, thereby simplifying the process for new and unfamiliar apps. Secondly, while our system shows proficiency on Android smartphones, its effectiveness on other Android-based devices remains untested. As previously indicated, there's potential to extend this voice-centric interface to a broader range of gadgets, including smartwatches and AR-VR head-mounted displays. Although the vocal commands might be processed by LLMs across devices, the user interfaces (UIs) of these devices can vary significantly in their logic and layout. For instance, the streamlined interface of a smartwatch might necessitate more concise output due to its smaller screen, while the immersive environment of an AR-VR device could introduce new interaction paradigms. This diversity in UI design and interaction methods across different devices requires more investigations in future works.

7 CONCLUSION

In this paper, we introduce `GptVoiceTasker`, an innovative virtual assistant designed to enhance user interactions and performance on smartphones. `GptVoiceTasker` leveraged advanced prompt engineering techniques to harness the capabilities of LLMs for interpreting user commands and constructing logical reasoning components. `GptVoiceTasker` further streamlined user interactions by automatically storing previous usages to automate subsequent repetitive tasks. Our experiments demonstrated outstanding command interpretation accuracy and the effectiveness of automated execution based on historical usage. In addition, the user evaluation validated `GptVoiceTasker`'s high usability in real-world tasks by improving user performance and reducing mental stress load, aligning with our design objectives. As an open-source project, `GptVoiceTasker`

paves the way for future enhancements in virtual assistant intuitiveness, contributing to the evolution of human-computer interactions. Further research includes applying our versatile database execution approach across diverse platforms and operating systems, as well as exploring innovative prompt engineering techniques to fine-tune LLMs for various reasoning tasks.

REFERENCES

[1] Voice Access. 2022. Troubleshoot Voice Access. https://support.google.com/accessibility/android/answer/6377053?hl=en#:~:text=If%20you%20have%20trouble%20starting,Access%20from%20the%20lock%20screen.

[2] Apple. [n. d.]. Siri. https://www.apple.com/au/siri/

[3] Deniz Arsan, Ali Zaidi, Aravind Sagar, and Ranjitha Kumar. 2021. App-Based Task Shortcuts for Virtual Assistants. In *The 34th Annual ACM Symposium on User Interface Software and Technology*. 1089–1099.

[4] Google Assistant. 2022. Assistant. https://assistant.google.com/intl/en_au/platforms/phones/

[5] Dzmitry Bahdanau, Kyung Hyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. In *3rd International Conference on Learning Representations, ICLR 2015*.

[6] Lijun Bai. 2022. Research on voice control technology for smart home system. In *Proceedings of the Asia Conference on Electrical, Power and Computer Engineering*. 1–7.

[7] Aaron Bangor, Philip T Kortum, and James T Miller. 2008. An empirical evaluation of the system usability scale. *Intl. Journal of Human–Computer Interaction* 24, 6 (2008), 574–594.

[8] Patrick Bareiß, Beatriz Souza, Marcelo d'Amorim, and Michael Pradel. 2022. Code generation tools (almost) for free? a study of few-shot, pre-trained language models on code. *arXiv preprint arXiv:2206.01335* (2022).

[9] Aditi Bhalerao, Samira Bhilare, Anagha Bondade, and Monal Shingade. 2017. Smart Voice Assistant: a universal voice control solution for non-visual access to the Android operating system. *Int. Res. J. Eng. Technol* 4, 2 (2017).

[10] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems* 33 (2020), 1877–1901.

[11] Andrea Burns, Deniz Arsan, Sanjna Agrawal, Ranjitha Kumar, Kate Saenko, and Bryan A. Plummer. 2022. A Dataset for Interactive Vision Language Navigation with Unknown Command Feasibility. In *European Conference on Computer Vision (ECCV)*.

[12] Julia Cambre, Alex C Williams, Afsaneh Razi, Ian Bicking, Abraham Wallin, Janice Tsai, Chinmay Kulkarni, and Jofish Kaye. 2021. Firefox voice: An open and extensible voice assistant built upon the web. In *Proceedings of the 2021 CHI conference on human factors in computing systems*. 1–18.

[13] Samuel Carreira, Tomás Marques, José Ribeiro, and Carlos Grilo. 2023. Revolutionizing Mobile Interaction: Enabling a 3 Billion Parameter GPT LLM on Mobile. *arXiv preprint arXiv:2310.01434* (2023).

[14] Jieshan Chen, Amanda Swearngin, Jason Wu, Titus Barik, Jeffrey Nichols, and Xiaoyi Zhang. 2022. Towards Complete Icon Labeling in Mobile Applications. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*. 1–14.

[15] Le Chen, Pei-Hung Lin, Tristan Vanderbruggen, Chunhua Liao, Murali Emani, and Bronis de Supinski. 2023. LM4HPC: Towards Effective Language Model Application in High-Performance Computing. In *International Workshop on OpenMP*. Springer, 18–33.

[16] Sen Chen, Lingling Fan, Chunyang Chen, Ting Su, Wenhe Li, Yang Liu, and Lihua Xu. 2019. StoryDroid: Automated Generation of Storyboard for Android Apps. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. 596–607. https://doi.org/10.1109/ICSE.2019.00070

[17] Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W Cohen. 2022. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *arXiv preprint arXiv:2211.12588* (2022).

[18] Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Sam Stevens, Boshi Wang, Huan Sun, and Yu Su. 2024. Mind2web: Towards a generalist agent for the web. *Advances in Neural Information Processing Systems* 36 (2024).

[19] Venita DePuy and Vance W Berger. 2014. Counterbalancing. *Wiley StatsRef: Statistics Reference Online* (2014).

[20] Amazon Developer. 2024. Build LLM-powered Alexa experiences. https://developer.amazon.com/en-US/alexa/alexa-ai

[21] Android Developers. 2022. AccessibilityNodeInfo. https://developer.android.com/reference/android/view/accessibility/AccessibilityNodeInfo.AccessibilityAction

[22] Android Developers. 2022. AccessibilityService. https://developer.android.com/guide/topics/ui/accessibility/service

[23] Apple Developers. 2024. Handle SiriKit intents in an Intents extension. https://developer.apple.com/documentation/xcode/configuring-siri-support#Handle-SiriKit-intents-in-an-Intents-extension

[24] Google Developers. 2024. Custom Intents. https://developers.google.com/assistant/app/custom-intents

[25] Feng Dong, Haoyu Wang, Li Li, Yao Guo, Tegawendé F Bissyandé, Tianming Liu, Guoai Xu, and Jacques Klein. 2018. Frauddroid: Automated ad fraud detection for android apps. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 257–268.

[26] Jiayue Fan, Chenning Xu, Chun Yu, and Yuanchun Shi. 2021. Just speak it: Minimize cognitive load for eyes-free text editing with a smart voice assistant. In *The 34th Annual ACM Symposium on User Interface Software and Technology*. 910–921.

[27] Sidong Feng and Chunyang Chen. 2023. Prompting Is All Your Need: Automated Android Bug Replay with Large Language Models. *arXiv preprint arXiv:2306.01987* (2023).

[28] Stephen Gilbert, Hugh Harvey, Tom Melvin, Erik Vollebregt, and Paul Wicks. 2023. Large language model AI chatbots require approval as medical devices. *Nature Medicine* (2023), 1–3.

[29] Bruce Golden. 1976. Shortest-path algorithms: A comparison. *Operations Research* 24, 6 (1976), 1164–1168.

[30] Sandra G Hart. 2006. NASA-task load index (NASA-TLX); 20 years later. In *Proceedings of the human factors and ergonomics society annual meeting*, Vol. 50. Sage publications Sage CA: Los Angeles, CA, 904–908.

[31] Julia Hirschberg and Christopher D. Manning. 2015. Advances in natural language processing. *Science* 349, 6245 (2015), 261–266.

[32] Matthew B Hoy. 2018. Alexa, Siri, Cortana, and more: an introduction to voice assistants. *Medical reference services quarterly* 37, 1 (2018), 81–88.

[33] Tae Soo Kim, Yoonjoo Lee, Minsuk Chang, and Juho Kim. 2023. Cells, generators, and lenses: Design framework for object-oriented interaction with large language models. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*. 1–18.

[34] Spyros Kokolakis. 2017. Privacy attitudes and privacy behaviour: A review of current research on the privacy paradox phenomenon. *Computers & security* 64 (2017), 122–134.

[35] Tsvi Kopelowitz and Ely Porat. 2018. A simple algorithm for approximating the text-to-pattern hamming distance. In *1st Symposium on Simplicity in Algorithms (SOSA 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

[36] Y Bala Krishna and S Nagendram. 2012. Zigbee based voice control system for smart home. *International Journal on Computer Technology and Applications* 3, 1 (2012), 163–168.

[37] Rebecca Krosnick and Steve Oney. 2022. ParamMacros: Creating UI Automation Leveraging End-User Natural Language Parameterization. In *2022 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. 1–10. https://doi.org/10.1109/VL/HCC53370.2022.9833005

[38] Wing Lam, Zhengkai Wu, Dengfeng Li, Wenyu Wang, Haibing Zheng, Hui Luo, Peng Yan, Yuetang Deng, and Tao Xie. 2017. Record and replay for android: Are we there yet in industrial cases?. In *Proceedings of the 2017 11th joint meeting on foundations of software engineering*. 854–859.

[39] Jaewook Lee, Jun Wang, Elizabeth Brown, Liam Chu, Sebastian S Rodriguez, and Jon E Froehlich. 2024. GazePointAR: A Context-Aware Multimodal Voice Assistant for Pronoun Disambiguation in Wearable Augmented Reality. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*. 1–20.

[40] Gang Li, Gilles Baechler, Manuel Tragut, and Yang Li. 2022. Learning to Denoise Raw Mobile UI Layouts for Improving Datasets at Scale. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems* (New Orleans, LA, USA) *(CHI '22)*. Association for Computing Machinery, New York, NY, USA, Article 67, 13 pages. https://doi.org/10.1145/3491102.3502042

[41] Toby Jia-Jun Li, Amos Azaria, and Brad A Myers. 2017. SUGILITE: creating multimodal smartphone automation by demonstration. In *Proceedings of the 2017 CHI conference on human factors in computing systems*. 6038–6049.

[42] Yang Li, Jiacong He, Xin Zhou, Yuan Zhang, and Jason Baldridge. 2020. Mapping natural language instructions to mobile UI action sequences. *arXiv preprint arXiv:2005.03776* (2020).

[43] Jason Xinyu Liu, Ziyi Yang, Ifrah Idrees, Sam Liang, Benjamin Schornstein, Stefanie Tellex, and Ankit Shah. 2023. Lang2LTL: Translating Natural Language Commands to Temporal Robot Task Specification. *arXiv preprint arXiv:2302.11649* (2023).

[44] Kuei-Chun Liu, Ching-Hung Wu, Shau-Yin Tseng, and Yin-Te Tsai. 2015. Voice helper: A mobile assistive system for visually impaired persons. In *2015 IEEE International Conference on Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing*. IEEE, 1400–1405.

[45] Tianming Liu, Haoyu Wang, Li Li, Xiapu Luo, Feng Dong, Yao Guo, Liu Wang, Tegawendé Bissyandé, and Jacques Klein. 2020. Maddroid: Characterizing and detecting devious ad contents for android apps. In *Proceedings of The Web Conference 2020*. 1715–1726.

[46] Vivian Liu and Lydia B Chilton. 2022. Design guidelines for prompt engineering text-to-image generative models. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*. 1–23.

[47] Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, et al. 2023. Agentbench: Evaluating llms as agents. *arXiv preprint arXiv:2308.03688* (2023).

[48] Zhe Liu, Chunyang Chen, Junjie Wang, Xing Che, Yuekai Huang, Jun Hu, and Qing Wang. 2023. Fill in the blank: Context-aware automated text input generation for mobile gui testing. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 1355–1367.

[49] Zhe Liu, Chunyang Chen, Junjie Wang, Mengzhuo Chen, Boyu Wu, Xing Che, Dandan Wang, and Qing Wang. 2023. Chatting with GPT-3 for Zero-Shot Human-Like Mobile Automated GUI Testing. *arXiv preprint arXiv:2305.09434* (2023).

[50] Chelsea Myers, Anushay Furqan, Jessica Nebolsky, Karina Caro, and Jichen Zhu. 2018. Patterns for How Users Overcome Obstacles in Voice User Interfaces *(CHI '18)*. Association for Computing Machinery, New York, NY, USA, 1–7. https://doi.org/10.1145/3173574.3173580

[51] OpenAI. 2023. API Reference - OpenAI API. https://platform.openai.com/docs/api-reference/introduction

[52] OpenAI. 2023. GPT-4 Technical Report. *arXiv preprint arXiv:2303.08774* (2023).

[53] Lihang Pan, Bowen Wang, Chun Yu, Yuxuan Chen, Xiangyu Zhang, and Yuanchun Shi. 2023. AutoTask: Executing Arbitrary Voice Commands by Exploring and Learning from Mobile GUI. *arXiv preprint arXiv:2312.16062* (2023).

[54] Lihang Pan, Chun Yu, JiaHui Li, Tian Huang, Xiaojun Bi, and Yuanchun Shi. 2022. Automatically Generating and Improving Voice Command Interface from Operation Sequences on Smartphones. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems* (New Orleans, LA, USA) *(CHI '22)*. Association for Computing Machinery, New York, NY, USA, Article 208, 21 pages. https://doi.org/10.1145/3491102.3517459

[55] Geonwoo Park and Harksoo Kim. 2018. Low-cost implementation of a named entity recognition system for voice-activated human-appliance interfaces in a smart home. *Sustainability* 10, 2 (2018), 488.

[56] Joon Sung Park, Joseph O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S Bernstein. 2023. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th annual acm symposium on user interface software and technology*. 1–22.

[57] Ethan Perez, Douwe Kiela, and Kyunghyun Cho. 2021. True few-shot learning with language models. *Advances in neural information processing systems* 34 (2021), 11054–11070.

[58] Md Saidur Rahaman, MM Ahsan, Nishath Anjum, Md Mizanur Rahman, and Md Nafizur Rahman. 2023. The AI race is on! Google's Bard and OpenAI's ChatGPT head to head: an opinion article. *Mizanur and Rahman, Md Nafizur, The AI Race is on* (2023).

[59] Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. 2021. Zero-shot text-to-image generation. In *International Conference on Machine Learning*. PMLR, 8821–8831.

[60] Christopher Rawles, Alice Li, Daniel Rodriguez, Oriana Riva, and Timothy Lillicrap. 2024. AndroidInTheWild: A Large-Scale Dataset For Android Device Control. *Advances in Neural Information Processing Systems* 36 (2024).

[61] Michael R Sheldon, Michael J Fillyaw, and W Douglas Thompson. 1996. The use and interpretation of the Friedman test in the analysis of ordinal-scale data in repeated measures designs. *Physiotherapy Research International* 1, 4 (1996), 221–228.

[62] Tianlin Shi, Andrej Karpathy, Linxi Fan, Jonathan Hernandez, and Percy Liang. 2017. World of bits: An open-domain platform for web-based agents. In *International Conference on Machine Learning*. PMLR, 3135–3144.

[63] Fatemeh Shiri, Terry Yue Zhuo, Zhuang Li, Shirui Pan, Weiqing Wang, Reza Haffari, Yuan-Fang Li, and Van Nguyen. 2022. Paraphrasing Techniques for Maritime QA system. In *2022 25th International Conference on Information Fusion (FUSION)*. IEEE, 1–8.

[64] Ishika Singh, Valts Blukis, Arsalan Mousavian, Ankit Goyal, Danfei Xu, Jonathan Tremblay, Dieter Fox, Jesse Thomason, and Animesh Garg. 2023. Progprompt: Generating situated robot task plans using large language models. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 11523–11530.

[65] Albert Yu Sun, Eliott Zemour, Arushi Saxena, Udith Vaidyanathan, Eric Lin, Christian Lau, and Vaikkunth Mugunthan. 2023. Does fine-tuning GPT-3 with the OpenAI API leak personally-identifiable information? *arXiv preprint arXiv:2307.16382* (2023).

[66] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288* (2023).

[67] Minh Duc Vu, Han Wang, Zhuang Li, Gholamreza Haffari, Zhenchang Xing, and Chunyang Chen. 2023. Voicify Your UI: Towards Android App Control with Voice Commands. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* 7, 1, Article 44 (mar 2023), 22 pages. https://doi.org/10.1145/3581998

[68] Bryan Wang, Gang Li, and Yang Li. 2023. Enabling conversational interaction with mobile ui using large language models. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*. 1–17.

[69] Lei Wang, Songheng Zhang, Yun Wang, Ee-Peng Lim, and Yong Wang. 2023. LLM4Vis: Explainable Visualization Recommendation using ChatGPT. *arXiv preprint arXiv:2310.07652* (2023).

[70] Yushi Wang, Jonathan Berant, and Percy Liang. 2015. Building a semantic parser overnight. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. 1332–1342.

[71] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Ed Chi, Quoc Le, and Denny Zhou. 2022. Chain of thought prompting elicits reasoning in large language models. *arXiv preprint arXiv:2201.11903* (2022).

[72] Hao Wen, Yuanchun Li, Guohong Liu, Shanhui Zhao, Tao Yu, Toby Jia-Jun Li, Shiqi Jiang, Yunhao Liu, Yaqin Zhang, and Yunxin Liu. 2023. Empowering LLM to use Smartphone for Intelligent Task Automation. *arXiv preprint arXiv:2308.15272* (2023).

[73] Shijie Wu, Ozan Irsoy, Steven Lu, Vadim Dabravolski, Mark Dredze, Sebastian Gehrmann, Prabhanjan Kambadur, David Rosenberg, and Gideon Mann. 2023. Bloomberggpt: A large language model for finance. *arXiv preprint arXiv:2303.17564* (2023).

[74] Feng Xiao and Long Wang. 2008. Asynchronous Consensus in Continuous-Time Multi-Agent Systems With Switching Topology and Time-Varying Delays. *IEEE Trans. Automat. Control* 53, 8 (2008), 1804–1816. https://doi.org/10.1109/TAC.2008.929381

[75] Silei Xu, Sina Semnani, Giovanni Campagna, and Monica Lam. 2020. AutoQA: From Databases To QA Semantic Parsers With Only Synthetic Training Data. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. 422–434.

[76] Kannon Yamada. 2020. How to control your Android device entirely with your voice. https://www.makeuseof.com/tag/control-android-device-entirely-voice/

[77] An Yan, Zhengyuan Yang, Wanrong Zhu, Kevin Lin, Linjie Li, Jianfeng Wang, Jianwei Yang, Yiwu Zhong, Julian McAuley, Jianfeng Gao, et al. 2023. Gpt-4v in wonderland: Large multimodal models for zero-shot smartphone gui navigation. *arXiv preprint arXiv:2311.07562* (2023).

[78] Jackie Yang, Monica S Lam, and James A Landay. 2020. Dothishere: multimodal interaction to improve cross-application tasks on mobile devices. In *Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology*. 35–44.

[79] Ziqi Yang, Xuhai Xu, Bingsheng Yao, Ethan Rogers, Shao Zhang, Stephen Intille, Nawar Shara, Guodong Gordon Gao, and Dakuo Wang. 2024. Talk2Care: An LLM-based Voice Assistant for Communication between Healthcare Providers and Older Adults. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 8, 2 (2024), 1–35.

[80] Yu Zhong, T. V. Raman, Casey Burkhardt, Fadi Biadsy, and Jeffrey P. Bigham. 2014. JustSpeak. *Proceedings of the 11th Web for All Conference on - W4A 14* (2014).

[81] Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc Le, et al. 2022. Least-to-most prompting enables complex reasoning in large language models. *arXiv preprint arXiv:2205.10625* (2022).