Enabling Conversational Interaction with Mobile UI using Large Language Models

Bryan Wang∗
University of Toronto
Toronto, ON, Canada
bryanw@dgp.toronto.edu

Gang Li
Google Research
Mountain View, CA, USA
leebird@google.com

Yang Li
Google Research
Mountain View, CA, USA
liyang@google.com

ABSTRACT
Conversational agents show the promise to allow users to interact with mobile devices using language. However, to perform diverse UI tasks with natural language, developers typically need to create separate datasets and models for each specific task, which is expensive and effort-consuming. Recently, pre-trained large language models (LLMs) have been shown capable of generalizing to various downstream tasks when prompted with a handful of examples from the target task. This paper investigates the feasibility of enabling versatile conversational interactions with mobile UIs using a single LLM. We propose a design space to categorize conversations between the user and the agent when collaboratively accomplishing mobile tasks. We design prompting techniques to adapt an LLM to conversational tasks on mobile UIs. The experiments show that our approach enables various conversational interactions with decent performances, manifesting its feasibility. We discuss the use cases of our work and its implications for language-based mobile interaction.

KEYWORDS
Large Language Models, Conversational Interaction, Mobile UI

1 INTRODUCTION
Interacting with computing devices using natural language is a long-standing pursuit in human-computer interaction [4, 14, 21]. Language, as both the input and the output, allows users to efficiently communicate with a computing system and access its functionalities when other I/O modalities are unavailable or cumbersome, e.g., users with motor or visual impairments, or situationally impaired when occupied by real-world tasks [44, 50]. Recently, intelligent assistants, such as Google Assistants and Siri, have significantly advanced language-based interaction for performing simple daily tasks, e.g., setting a timer. However, these intelligent agents still fall short of enabling conversational interaction in existing mobile user interfaces where many user tasks are performed [28]. For example, answering the user’s question regarding specific information on the screen. It often requires an agent to have a computational understanding of GUI screens and the tasks they support, which is lacking in existing intelligent assistants.

To enable conversational interactions in mobile UIs, prior work has investigated several important technical building blocks, e.g., summarizing a mobile screen for users to quickly understand its purpose [46], mapping language instructions to graphical user interface (GUI) actions [34, 38, 40] or modeling GUIs so that they are more amenable for language-based interaction [31, 36, 46, 51, 53]. Each of these works addresses a specific aspect of conversational interaction and requires significant effort, e.g., curating task-specific datasets with tens of thousands of examples and training dedicated models [34, 35, 37, 46]. There is a broad spectrum of conversational interactions that can happen on mobile UIs, as Todi et al. have revealed [43]. It is desirable to develop a more lightweight and generalizable approach to realize conversational interaction on GUIs.

Recently, pre-trained large language models (LLMs) such as GPT-3 [6] and PaLM [9] have shown abilities to adapt themselves to various downstream tasks when being prompted with a handful of examples of the target task. Such generalizability is promising for us to potentially support various conversational tasks on GUIs without having to develop a specific model for each task. However, the feasibility of doing so is unclear. Little work has been conducted to understand how LLMs, trained with natural languages, can work with GUIs for conversational interaction. Therefore, we investigate in this paper the viability and the how-to of utilizing LLMs to enable diverse language-based interactions with Mobile UIs.

Inspired by the theory of grounding in communication [11], we propose a design space consisting of four types of unit conversations between a user and a conversational agent to establish mutual understanding when working together to accomplish mobile tasks. We categorize the conversations as either human or agent-initiated and whether the goal is to solicit or provide information. Further, we propose a set of prompting principles specifically developed for prompting LLMs for mobile UI tasks. To demonstrate the feasibility of our approach, we experimented with tasks from each conversation category, including screen question-generation, screen summarization, screen question-answering, and mapping Instruction to UI action. Our evaluation shows that our approach can achieve decent performance on each UI task with only two examples or fewer. Lastly, we discuss the use cases of the proposed methods and the implications of our investigation. Specifically, our work showcases LLMs’ potential for interaction designers and developers to rapidly prototype conversational interactions and test them with users before investing efforts and budget into developing dedicated datasets and models.

In summary, our paper makes the following contributions:

- Our work is the first investigation for using LLMs to enable conversational interaction on mobile UIs, which advances the understanding of using LLMs for interaction tasks.
- We propose a design space that categorizes four types of user-agent conversation when they collaboratively accomplish a mobile task—that lays a conceptual framework for others to further study the topic.

∗This work was done while the author was an intern at Google Research.
• We designed a novel method for feeding GUIs to LLMs—that is pre-trained for natural language—and a set of techniques to prompt LLMs to perform a range of conversational tasks on mobile UI screens. These techniques produce competitive performance; others can immediately use them in their work.
• We experimented with our approach with four language-based interaction tasks, demonstrating our approach’s feasibility with LLMs for conversational GUI interaction and potentially lowering the threshold for developing conversational agents for GUIs.

2 RELATED WORK
Our work is related to the literature on bridging user interfaces and natural language, prompting techniques for LLMs, and the use of LLMs to facilitate interactive tasks.

2.1 Bridging GUIs with Natural Language
There has been increasing interest in using machine learning to bridge graphical user interfaces and natural language for use cases such as accessibility and multimodal interaction. For example, Widget Captioning [35] and Screen Recognition [53] predict semantically meaningful alt-text labels for GUI components. Screen2Words [46] took a step further to predict text summaries that concisely describe the entire screen. Li et al. [34] uses a transformer-based model to map natural language instructions to mobile UI action sequences. These prior works typically train a model dedicated to the task based on a sizeable dataset collected. In contrast, our work leverages the few-shot learning ability of LLMs to enable language-based UI tasks by providing a small number of examples. To achieve this, we propose a novel method to represent the UI so that an LLM pre-trained for natural language can efficiently process it. Another relevant body of work is to develop a conversational or multimodal agent that can help the user accomplish mobile tasks [28–30, 32]. For example, SUGILITE [28] enables users to create task automation on smartphones by user demonstration and perform the tasks through a conversational interface. KITE [30] helps developers create task-oriented bots templates from existing apps. Our work shows that LLMs can enable versatile language-based interactions when prompted with exemplars for different tasks, lowering the threshold for developing versatile multimodal agents.

2.2 Prompting Pre-trained Large Language Models
Finetuning pre-trained task-invariant models such as BERT has been a common practice to leverage and adapt large models for specific tasks. Yet, the GPT-3 [6] model with 175B parameters introduced a new norm for leveraging pre-trained language models for downstream tasks by so-called in-context few-shot learning. By prompting the pre-trained model with only a few examples, it can generalize to various downstream tasks without updating the parameters in the underlying model. Recently studies have shown that prompting is one of the emergent abilities that appear only when the model size is large enough [48]. While prompting LLMs may not outperform benchmark models on many tasks, it provides a lightweight method to achieve competitive performance on various tasks [6, 9]. Typically, when the prompt consists of N pairs of input and output exemplars from the target tasks, it is referred to as N-shot learning. When more shots are provided, the model generally performs better on the target tasks [6, 9]. Prior work has also shown that by using specific prompting language such as "Let’s think step by step", one can solicit reasoning from the model to perform tasks that require logical reasoning, such as solving math problems [23], in a zero-shot setting. In addition, various prompting paradigms have been proposed to solicit reasoning from the language model [47, 49, 54]. For example, chain-of-thought prompting [49] proposes to use the models to generate intermediate results (i.e., chain of thoughts) before generating the final output. The core idea resembles the divide-and-conquer method in algorithms which breaks more complicated problems into subproblems that can be solved more easily. Prompting LLMs remains an ongoing research topic in the community. Our work builds upon prior work to contribute a set of prompting techniques designed to prompt LLMs for conversational tasks on mobile UIs.

2.3 Interactive Applications of Large Language Models
LLMs have been applied to enable a broad range of language-related interactive applications in the HCI community [1, 10, 12, 19, 20, 22, 24–26, 39, 52]. For example, Chang et al. [10] proposes TaleBrush, a generative story ideation tool that uses line sketching interactions with a GPT-based language model for control and sensemaking of a protagonist’s fortune in co-created stories. Stylette [22] allows users to modify web designs with language commands and uses LLMs to infer the corresponding CSS properties. Lee et al [24] present CoAuthor, a dataset designed for revealing GPT-3’s capabilities in assisting creative and argumentative writing. Since LLMs can encode a wealth of semantic knowledge, they have also been used to support physical applications. For example, SayCan [1] extracts and leverages the knowledge priors within LLMs to execute real-world, abstract, long-horizon robot commands. Our work contributes the first effort of applying LLMs to enable various conversational interactions on mobile UIs.

3 CONVERSATION FOR MOBILE UI TASKS
To guide a systematic investigation of using LLMs to support conversational mobile interaction, we derived a design space to characterize conversations between users and agents when carrying out tasks on mobile UIs. Conversational interaction with mobile devices is typically embodied as human users conversing with an intelligent assistant. However, unlike chit-chat conversational bots that support free-form, open-ended conversations, intelligent assistants focus on goal-oriented conversations—for example, helping users accomplish mobile tasks such as booking hotels and checking emails. During the conversation, the user and the agent exchange information necessary to achieve the user goals, which resembles the process of grounding.

Communication theories [8, 11] define grounding as the process of building a common ground based on shared mutual information to communicate successfully. According to the original model, there are two phases of contributions to the conversation. 1) Presentation phase, in which the speaker produces an utterance addressed to a conversational partner, and 2) Acceptance phase, in which the
partner explicitly acknowledge the utterance or implicitly accept it by continuing with the following relevant utterance [5].

The original grounding theory addresses human-to-human conversation. We adopt the two-phase model to the context of human-agent conversations for mobile tasks. Since we consider only goal-oriented conversations, we assume that any contribution to the conversation is 1) based on the screen context and 2) made to exchange information necessary for goal completion. This excludes chit-chat-type conversations. We focus on unit conversations that are simplistic and fundamental but can be chained together to form a multi-turn conversation. We define a unit conversation as a single-turn conversation consisting of an agent turn and a user turn. As shown in figure 1, we categorize unit conversations with two dimensions: Initiative and Purpose. A conversation can be mixed-initiative [17], either initiated by the agent or the user. The purpose of initiated conversation could be either soliciting or providing information. This step resembles the Presentation phase. Once a conversation is initiated, the receiver would need to provide the inquired information or acknowledge that the message has been received. This step resembles the Acceptance phase. A unique characteristic of human-agent conversation is that, in addition to a language acknowledgment, the acceptance phase for the agent could be executing actions based on the user’s utterances, e.g., clicking on the home button. Users who see the screen updated by an executed action realize their messages have been well-received.

3.1 Agent-initiated Conversation

When an agent initiates a conversation, it can be either soliciting or providing information essential for the user to proceed on a mobile UI screen.

3.1.1 Agent solicits information from user. Mobile UIs usually require users to input information relevant to their goals. For example, destination city or travel dates for hotel booking. This information is typically requested by the input fields on mobile UIs. When users see a text field for the destination city, they know the UI expects them to enter where they plan to travel. A conversational agent should be able to similarly solicit essential information from users using language. For example, it should ask users questions like “Which hotel do you want to search for?” or “What is the check-in date of your stays?” We refer to this type of task as Screen Question-Generation since the questions should be generated based on the current screen contexts. After the questions are asked, the user could respond to provide the inquired information.

3.1.2 Agent provides information to user. An important purpose of GUIs is visually presenting information to users. Similarly, a conversational agent should be able to similarly solicit essential information from users using language. For example, it should ask users questions like “Which hotel do you want to search for?” or “What is the check-in date of your stays?” We refer to this type of task as Screen Summarization [46], which provides a short description of the purpose of the current screen, e.g. “A list of grocery stores nearby”, or
"A step-by-step recipes of butter chicken". The descriptions can help users quickly understand the UI when visual information is unavailable. To minimize interaction effort, the user is not required to acknowledge the agent’s information. However, the user can initiate follow-up conversations to solicit further information from the agent.

3.2 User-initiated Conversation
Similarly, the user can initiate conversations to request information or proactively provide information for the agent to process.

3.2.1 User solicits information from the agent. A single mobile screen can sometimes contain a significant amount of text information, e.g., product specifications. Finding the specific information that the user cares about, e.g., the size of a television, requires user effort to skim through the long texts and spot the relevant information. It becomes even inefficient if users have to access the texts through screen readers—they will need to wait for screen readers to scan through all the content on the screen until the relevant text is read. To make information access more efficient, the user should be able to request specific screen information from the agent using language. When the request is done with a question such as “What’s the TV’s size?”, the agent should respond “50 inches” based on the specifications presented on the screen. We call this type of conversation Screen Question-Answering, similar to the visual question-answering (VQA) [3] and standard text question-answering [41] but instead based on a mobile UI screen.

3.2.2 User provides information to the agent. Users can take the initiative to communicate new information to guide the agent in carrying out mobile tasks. The information can be slot values for input fields on the screen such as “My password is CHI2023.” It can also be language commands that convey user intent, such as “Turn on the WIFI,” or “Click on the search button.” After receiving information from the user, the agent could acknowledge with a language response such as “Sure, I will click on the search button.” and/or perform the UI actions to click the corresponding button—the task referred to as Mapping Instruction to UI Action.

4 PROMPTING LARGE-LANGUAGE MODELS FOR MOBILE UI TASKS
Pre-trained LLMs support in-context few-shot learning via prompting—instead of finetuning or re-training models for each new task, one can prompt an LLM with a few input and output exemplars of the desired task [6, 9, 49, 54]. For some NLP tasks such as question-answering or translation, prompting can perform on par with previous benchmark approaches [6]. However, language models take text input only, while mobile UIs are multimodal, containing text, image, and structural information in their view hierarchy data and screenshots. In addition, since the UIs of a mobile app encapsulates the logic of target user tasks [33], logical reasoning based on UI contexts is essential for the model to support conversations toward task completion. These unique aspects of mobile UIs pose two open problems for designing prompts:

- (1) How to represent mobile UIs in texts to leverage the few-shot prompting of LLMs?
- (2) How to elicit reasoning based on the mobile UIs?

We respond to these questions by describing our proposed prompting techniques and their design rationales below. Prompting LLMs remains an ongoing research problem. As the first to investigate prompting with mobile UI, we provide a strong baseline approach and encourage future work to build upon our design and further study the open problems.

4.1 Screen Representation
4.1.1 Representing View Hierarchy as HTML. There can be various ways to represent a mobile UI in text, e.g., concatenating all the text elements on the UI into a token sequence or using natural language sentences to describe UI elements, such as “a menu button in the top left corner.” To design our screen representation, we leverage the insight that if a prompt falls within the training data distribution of a large language model, it is more likely for few-shot learning to perform. This is because LLMs are trained to predict the subsequent tokens that maximize the probability based on the training data. LLMs’ training data is typically scraped from the web, including both natural language and code. For example, 5% of PaLM’s [9] training data was scraped from GitHub, including 24 common programming languages such as Java, HTML, and Python [9]. Therefore, we represent a mobile UI in texts by converting its view hierarchy data using HTML syntax. HTML is particularly suitable for representing mobile UIs as it is already a markup language representing web UIs. The conversion is conducted by traversing the view hierarchy tree using a depth-first search. We detail our conversion algorithm in the following sections. Note that since the view hierarchy is not designed to be represented in HTML syntax, a perfect one-to-one conversion does not exist. In contrast, our goal is to make the converted view hierarchy look similar to the HTML syntax to generate a data representation closer to the training data distribution.

4.1.2 View Hierarchy Properties. Converting a mobile UI’s view hierarchy into HTML syntax can preserve the detailed properties of UI elements and their structural relationship. The view hierarchy is a structural tree representation of the UI where each node, corresponding to a UI element, contains various properties such as the class, visibility-to-user, and the element’s bounds. However, using all element properties will result in lengthy HTML text, which may exceed the input length limit of the language model, e.g., 1920 tokens for PaLM and 2048 tokens for GPT-3. Therefore, we use a subset of properties related to the text description of an element:

- class: Android object type such as TextView or Button.
- text: element texts that are visible to the user.
- resource_id: text identifiers that describes the referenced resource.
- content_desc: content description that describes the element for accessibility purpose—the alt-text.

4.1.3 Class Mapping. We developed heuristics to map the Android classes to HTML tags with similar functionalities. We map TextView to the <p> tag as they are both used for presenting texts; all button-related classes such as Button or ImageButton are mapped to <button>. We map all image-related classes such as IMAGEVIEW to <img>, including icons and images. Lastly, we convert the text input class EditText to <input> tag. We focus on the
most common element classes for simplicity, and the rest of the Android classes, including containers such as LinearLayout are mapped to the `<div>` tag.

4.1.4 Text, Resource_Id, and Content Description. We insert the text properties of Android elements in between the opening and closing HTML tags, following the standard syntax of texts in HTML. The resource_id property contains three entities: package_name, resource_type, and resource_name. Among them, resource_name usually contains additional descriptions of an element’s functionality or purpose, written by the developers. For example, in the Gmail app, an element with resource_name of "unread_count_textView" shows how many emails are unread; whereas a "date" means the element shows the date of receiving a mail. Such information helps the model to better understand the screen context. We insert the resource_name tokens that describe each element’s purpose as additional identifiers in the "class" attributes, which originally contain identifiers linked to a style sheet or used by JavaScript to access the element. Word tokens in resource_name are typically concatenated with underscores, which we replace as spaces when inserting. Lastly, we insert the content_desc as the "alt" attribute in the HTML tags when the property is present.

4.1.5 Numeric Indexes for Referencing. To help model referencing specific UI elements, we insert numeric indexes to each element as the "id" attribute. The indexes are generated with the depth-first search order in the view hierarchy tree. For tasks such as predicting which button to click based on language instructions, the model can refer to elements using numeric indexes, which is more efficient and space-saving than spelling out the complete HTML tag.

4.2 Chain-of-Thought Prompting
Mobile UIs encapsulate the logic of user tasks [33]; therefore, it is vital for models to perform reasoning when used for conversational interaction. LLMs have demonstrated abilities to reason [1, 6, 9] as they captured real-world knowledge during training with a large number of texts. Recent work further shows that LLM’s reasoning ability can be improved by generating and chaining intermediate results to obtain the final answers, namely Chain-of-Thought prompting [49]. The idea is straightforward, i.e., simply appending a chain of thoughts describing intermediate results before the answers in the prompt. The model would then follow the patterns to generate a chain of thoughts during inference. Chain-of-Thought prompting has been shown to be helpful for reasoning tasks. The results are also more interpretable as the model would articulate its thought process before coming up with the answer. However, prior work has not investigated whether it can facilitate reasoning in generating conversations based on mobile UIs. Therefore, we incorporate the method in the task that is applicable in our experiments.

4.3 Prompt Structure
We follow a similar prompt structure shown effective in [6]. Each prompt starts with a preamble which explains the prompt’s purpose.
The preamble is followed by multiple exemplars consisting of the input, a chain of thought (if applicable), and the output for each task. Each exemplar's input is a mobile screen in the HTML syntax. To better leverage few-shot learning while complying with LLM's input length limits, we only show the leaf nodes visible to the users, as non-leaf nodes are usually containers that do not contain textual information. Following the input, a chain of thoughts is provided to elicit logical reasoning from LLMs, if applicable to the task. The output is the desired outcome for the target tasks, e.g., a screen summary or an answer of the question asked by the user. Figure 2-left shows an example of a 1-shot prompt. Few-shot prompting can be achieved with more than one exemplar included in the prompt. During prediction, we feed the model with the prompt with a new input screen appended at the end. Therefore, for N-shot learning, the prompt will consist of a preamble, N exemplars, and the test screen for prediction, as shown in Figure 2-right.

5 FEASIBILITY EXPERIMENTS

As shown in figure 3, we demonstrate the feasibility of using LLMs to enable conversations on GUIs through experiments with four tasks: 1) Screen Question-Generation, 2) Screen Summarization, 3) Screen Question-Answering, and 4) Mapping Instruction to UI Action. Following the common practices of few-shot prompting [6, 49], we select a handful of exemplar data to construct prompts for each task. We then evaluate the effectiveness using task-specific metrics detailed in each experiment. All the studies were conducted with the PaLM model [9], which performs similarly to other LLMs such as GPT-3 [49]. The PaLM model is trained with a maximum input length of 1920 tokens. Therefore, we limit the number of exemplars in a prompt to be two or fewer, excluding the test screen, to avoid exceeding the length limit. The experiments aim to understand what can be achieved by simply prompting LLMs with a few exemplars from the target tasks and compared to the baseline, or benchmark if available.

5.1 Screen Question-Generation

5.1.1 Task Formulation. Given a mobile UI screen, the goal of screen question-generation is to synthesize coherent, grammatically-correct natural language questions relevant to the UI elements requiring user input. The task occurs when the agent requests user input for the UI elements.

5.1.2 Prompt Construction. Figure 2 shows an example prompt we used to generate questions. We use a preamble of "Given a screen, the agent needs to identify the elements requiring user input and generate corresponding questions." We used chain-of-thought techniques to generate three intermediate results 1) input fields count, 2) screen summary, and 3) input enumeration. The input field counts were fed with a ground truth count extracted from the screen HTML. We found this step essential to prevent the model from omitting some input elements and only generating a subset of questions. Next, we asked the model to summarize the screen's purpose, which produces the screen context that provides details in the questions. After that, the model enumerates which elements are asking for what information. After the chain of thoughts, the model generates the questions, enclosed by <SOQ> and <EOQ> tokens, representing start-of-question and end-of-question, respectively. The tokens are used as delimiters for conveniently parsing the questions from the output texts generated by the model. We use similar ways to insert special tokens for parsing model output in the rest of the experiments. An example prompt can be found in appendix A.1.

5.1.3 Experimental Setup. We aim to understand the quality of LLMs for natural language generation based on UI elements. Since there is currently no existing dataset for screen question generation, we follow the common practice of evaluating language generation quality with human ratings. We randomly sampled 400 screens from the RICO dataset[13]. Each of these screens contains at least one EditText element, representing the text input field for users to enter information on the UI. We randomly select another two screens from the RICO dataset as exemplars to include in the prompt. An EditText element represents an input field for the user to enter information, and we generate questions for every input field. We labeled questions generated from the sampled screens using a prompt constructed with two exemplars. Some screens contain multiple input fields, and sometimes several of them are relevant and can be asked collectively. For example, three fields asking for the birth year, month, and date can be combined into a single question as "when is your birthday?". Combining questions can lead to a more efficient conversation between an agent and a user. Therefore, we include an exemplar that combines relevant questions in the prompt to see if the model will also learn to combine relevant questions.

We compare LLM's results with a rule-based approach that use words in resource_id, referred to as res_tokens, to fill in the template of "What is res_tokens?". We use res_token instead of text because most text input fields are blank by default, and res_token contains the most meaningful description of an input field. We recruited 17 raters who work as professional data labelers at a tech
company to provide ratings. To ensure the quality of the labels, a group of quality audits sampled and reviewed 5% of the total number of questions answered by every rater. We provide a UI screenshot of a mobile UI and a generated question for each labeling task. The EditText element associated with the question is highlighted with a bounding box. We solicit human ratings on whether the questions are grammatically correct and relevant to the input fields for which they were generated. In addition to the human-labeled language quality, we automatically test how well LLMs can cover all the elements that need question generation. The evaluation metrics include:

- **Grammar Correctness**: How correct is the grammar of a generated question? Are the sentences intelligible and plausible? This metric tests the language generation quality in general and is rated on a 5-point Likert scale with 1 as completely incorrect and 5 as completely correct.

- **UI Relevance**: Whether a generated question is relevant to the highlighted UI element. This metric tests whether the connection between a UI element and a question is correctly established by the model, which is rated on a binary scale as either relevant or not relevant.

- **Question Coverage**: How well can the model identify the elements on the screen that need question generation? This metric is automatically computed by comparing the indices of ground truth input elements with those identified by the model within the chain of thoughts.

5.1.4 Results. We evaluated 931 questions for both the LLMs and the template-based approach. Three different human raters examined each question to obtain aggregated scores. Table 1 shows the results of our evaluation. Our approach achieves an almost perfect average score of 4.98 on grammar correctness, while the rule-based approach receives a 3.6 average rating. A Mann–Whitney U test shows that the difference between the two methods is statistically significant (p < 0.0001). LLMs also generate 8.7% more relevant questions compared to the baseline. In terms of questions coverage, our approach achieves an F1 score of 95.9% (precision = 95.4%, recall = 96.3%). Since the rule-based method iterates through every input field to generate questions, its question coverage is naturally 100%. Altogether, the results show that our approach can precisely identify input elements and generate relevant questions that are intelligible.

We further analyzed the model behaviors and found that when generating a question for a field, the model considers both the field element and the screen context (information from other screen objects). For example, Figure 4 shows how the model leveraged screen contexts to generate four questions for the input fields on a credit card register screen. While the baseline outputs use the ref_tokens to convey somewhat relevant information, they are less intelligible than the LLM output and do not articulate the specific information requested by the fields. In contrast, all four questions generated by LLM are grammatically correct and ask for relevant information. For Question 3 in figure 4, the LLM additionally uses the texts above the input field to ask for the “last 4 digits of SSN”.

Note that the model does not simply copy the screen contexts. Instead, it blends the contexts into the generated question. For instance, Question 2 asks for “credit card expiration date,” while the texts above did not mention the word “credit.” We also observed that the model did exhibit the behavior of combining relevant fields into a single question on three test screens. For example, Figure 4 shows the model can combine two input elements asking for minimum and maximum values for price into a single question “What is the price range?”.

**Table 1: Grammar correctness, UI relevance, and question coverage results from the screen question-generation experiment.**

<table>
<thead>
<tr>
<th>Method</th>
<th>Grammar</th>
<th>Relevance</th>
<th>Coverage</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Template</td>
<td>3.60 (σ=0.69)</td>
<td>84.1%</td>
<td>100%</td>
<td></td>
</tr>
<tr>
<td>LLM</td>
<td>4.98 (σ=0.07)</td>
<td>92.8%</td>
<td>95.9%</td>
<td></td>
</tr>
</tbody>
</table>

**Figure 4: Example screen questions generated by the LLM.** Left: The LLM can utilize screen contexts to generate grammatically-correct questions relevant to each input field on the mobile UI, while the template approach falls short. Right: We observed that the LLM could use its prior knowledge to combine multiple related input fields to ask a single question. The example shows the LLM combing the fields for the minimum and maximum prices into a single question asking about the price range. Elements relevant to each question are highlighted in corresponding colors and numbered indexes.
Figure 5: Example summaries generated by prompting the LLM with 2 exemplars (2-shot learning). The LLM is likelier to use specific texts on the screen to compose summaries (top-left and bottom-right). Moreover, the LLM is more likely to generate more extended summaries that leverage multiple vital elements on the screen (top-right). We also observed that the LLM would use its prior knowledge to help summarize the screens. For example, the bottom-right shows the LLM had inferred the screen is for the London Tube system from the station names displayed on the screen. UI elements relevant to the highlighted phrases in the summaries are called out by bounding boxes with corresponding colors. Screen2Words outputs were obtained from the original paper authors.

<table>
<thead>
<tr>
<th>Model</th>
<th>BLEU-1</th>
<th>BLEU-2</th>
<th>BLEU-3</th>
<th>BLEU-4</th>
<th>CIDEr</th>
<th>ROUGE-L</th>
<th>METEOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-shot LLM</td>
<td>7.8</td>
<td>6.4</td>
<td>5.9</td>
<td>5.7</td>
<td>1.5</td>
<td>3.4</td>
<td>4.5</td>
</tr>
<tr>
<td>1-shot LLM</td>
<td>42.3</td>
<td>21.1</td>
<td>14.8</td>
<td>12.1</td>
<td>40.9</td>
<td>28.9</td>
<td>15.3</td>
</tr>
<tr>
<td>2-shot LLM</td>
<td>45.0</td>
<td>25.1</td>
<td>17.6</td>
<td>14.1</td>
<td>39.9</td>
<td>33.0</td>
<td>17.7</td>
</tr>
<tr>
<td>Screen2Words [46]</td>
<td>65.5</td>
<td>45.8</td>
<td>32.4</td>
<td>25.1</td>
<td>61.3</td>
<td>48.6</td>
<td>29.5</td>
</tr>
</tbody>
</table>

5.2 Screen Summarization

5.2.1 Task Formulation. Screen summarization was proposed in [46] as the automatic generation of descriptive language overviews that cover essential functionalities of mobile screens. The task helps users quickly understand the purpose of a mobile UI, which is particularly useful when the UI is not visually accessible.
We found that the LLM is more likely to use specific texts on the words frequently. Since these words also appear frequently in the test set's ground truth references, the model could also get a high score by favoring these words. During manually reviewing the results, we found that the summaries generated by the LLM are of high quality. However, since the model is not fine-tuned with the whole dataset, the language distribution that the LLM used to predict the summary differs from the Screen2Words model, leading to lower scores across automatic metrics when compared with the human labels from Screen2Words. Figure 5 shows screens with summaries annotated by human labelers and the output from both Screen2Words and the LLM model. We found that the LLM is more likely to use specific texts on the screen to compose summaries, such as San Francisco (top-left) and Tiramisu Cake Pop (bottom-left) while the Screen2Words dataset and the benchmark model output tend to be more generic. Moreover, the LLM is more likely to generate more extended summaries that leverage multiple vital elements on the screen. For example, the bottom-right photo shows a station search results page for London’s tube system. The LLM predicts "Search results for a subway stop in the London tube system." However, the input HTML does not contain the words 'London' nor 'tube.' Therefore, the model has utilized its prior knowledge about the station names, learned from large language datasets, to infer that they belong to the London Tube. Such summaries may not have been generated if the model is trained only on the Screen2Words dataset. This explains the gap between our automatic scores for LLM’s summaries and the observed actual quality.

### 5.3 Screen Question-Answering (QA)

#### 5.3.1 Task Formulation. Given a mobile UI and an open-ended question asking about information regarding the UI, the model should provide the correct answer. We focus on factual questions, which require answers based on facts presented on the screen.

#### 5.3.2 Prompt Construction. We use a preamble of "Given a screen and a question, provide the answer based on the screen information". We did not use chain-of-thought prompting as no intermediate result is needed to be generated for the task. In our experiments, N sets of screen HTML and question-answer pairs follow the preamble, where N = 0, 1, 2 represents N-shot learning. The output answers are enclosed by special tags <SOA> and <EOA>, meaning the start and end of an answer, respectively. The prompt we used in the study can be found in appendix A.2.

#### 5.3.3 Experiment Setup. We use a dataset of 300 human-labeled question-answer pairs from 121 unique screens in the RICO dataset [13]. It is a preliminary dataset obtained from the authors of a large-scale data collection for screen question-answering [18]. The data labeling process involves two stages: question annotation and answer annotation. For question annotation, the annotators were asked to frame questions given a screenshot as the context. The annotators were expected to compose questions that only inquire about information that requires no logical reasoning and can be directly read off from the screen. After that, another set of annotators answers the previously annotated questions given the associated screenshots. We randomly held out three screens associated with 12 QA pairs for prompt construction. We randomly select one QA pair from each screen to include in the prompt. In the remaining 288 screens, 57 ground truth answers are not present in the view hierarchy data. This is expected because the labeling is based on screenshots instead of view hierarchies, and many screens in the RICO dataset contain inaccurate view hierarchy data [27]. In this work, we focus on the answers that are present in the view hierarchy data, and incorporating screenshot information in the LLM will be a critical direction to investigate in the future.

Since the answers are generated instead of retrieved from screen HTML, some correct answers may not completely match the labels. For example, "2.7.3" versus "version 2.7.3". Therefore, we report performances on four metrics: 1) Exact Matches: the predicted answer is identical to the ground truth. 2) Contains GT: the answer is longer than the ground truth and fully contains it. 3) Sub-String of GT: the answer is a sub-string of the ground truth. 4) Micro-F1: the micro F1 scores, calculated based on the number of shared words between the predicted answer and the ground truth.
5.3.4 Results. Table 3 shows the QA results of different settings. Unlike screen summarization, we found that the LLM can already perform screen QA with the zero-shot setting. 30.7% of the generated answers match exactly with the ground truth, 6.5% answers contain the ground truth, and 5.6% answers are sub-strings of the ground truth. The zero-shot performance might be because the training data of LLMs already contain many QA-related data from the internet. Therefore, the model had already learned to perform question-answering. The off-the-shelf DistilBERT model achieves 36% Exact Match, 8.5% Contains GT scores, and 9.9% Contains GT scores, slightly better than the zero-shot performance of LLMs.

DistilBERT model performs much poorly on our tasks compared to standard question-answering benchmarks, which might be because it was not trained with HTML data. Similar to screen summarization, by providing a single exemplar, the performance boosted significantly, achieving 65.8% Exact Match, 10% Contains GT, and 7.8% Sub-String of GT–summing up to 83.6% answers relevant to the ground truth. However, we again found that the 2-shot setting only leads to a moderate performance boost compared to the 1-shot setting. Figure 6-left shows example QA results from our experiment using 2-shot learning. The LLM can effectively understand the screen and generate a correct or relevant answer. For the shown screen, the LLM correctly answers Q1, Q2, and Q4. For Q3, the LLM generates an answer containing the ground truth “Dec 23rd, 2016” but includes the time within the day “4:50” am. In contrast, the baseline model trained with standard text question-answering corpus only managed to answer Q4 correctly; for Q3, its answer only contains “2016”, a sub-string of the ground truth. Q2 shows that the baseline model sometimes incorrectly retrieves HTML code from the input screen. Figure 6-right shows different relationships between the predicted answer and ground truth.
5.4 Mapping Instruction to UI Action

5.4.1 Task Formulation. Given a mobile UI screen and a natural language instruction to control the UI, the model needs to identify the correct object to perform the instructed action. For example, when instructed with “Open Gmail,” the model should correctly identify the Gmail icon on the home screen. This task is useful for controlling mobile apps using language input such as voice access.

5.4.2 Prompt Construction. We use a preamble of “Given a screen, an instruction, predict the id of the UI element to perform the instruction.”. The preamble is followed by $N$ exemplars consisting of the screen HTML, instruction, and ground truth id. Here $N = 0, 1, 2$ representing $N$-shot learning. We did not use chain-of-thought prompting as no intermediate result is needed to be generated for the task. The output answers are enclosed by special tags <SOI> and <EOI>, meaning the start and end of the predicted element id, respectively. An example prompt can be found in appendix A.4.

5.4.3 Experiment Setup. We use the PixelHelp dataset [34], which contains 187 multi-step instructions for performing everyday tasks on Google Pixel phones, such as switching Wi-Fi settings or checking emails. We randomly sampled one screen from each unique app package in the dataset as prompt modules. When constructing prompts, we randomly sampled from the prompt modules. We conducted experiments under two conditions: 1) in-app and 2) cross-app. In the former, the prompt contains a prompt module from the same app package with the test screen, while in the latter, it does not. We expect the in-app case will have better performance. Following the original paper, we report the percentage of partial matches and complete matches of target element sequences.

5.4.4 Results. Our experimental results show that the 0-shot setting cannot perform the task at all, with nearly zero partial or complete accuracy. In the cross-app condition, one-shot prompting significantly achieves 74.69 partial and 31.67 complete, meaning 75% of elements associated with the instructions were correctly predicted, and more than 30% tasks are entirely correct. The 2-shot setting offers incremental boosts for both metrics. In the in-app condition, both the 1-shot and 2-shot settings achieve higher scores than their counterparts in the cross-app condition. Our best performing setting is the 2-shot LLM & in-app, which achieves 80.36 partial and 45.0 complete accuracy scores, as shown in Table 4. While our approach underperforms the benchmark results from the Seq2Act model [34], it shows impressive performance of few-shot learning, which uses only two examples in the prompt. At the same time, Seq2Act was trained on several dedicated datasets with hundreds of thousands of examples [34]. Few-shot prompting is challenging as the model can only see a few examples from the target tasks and does not update its parameters [6]. Therefore, we do not expect prompting LLMs can consistently achieve better performance than the dedicated models across all tasks.

Table 4: Mapping Instruction to UI Action Results

<table>
<thead>
<tr>
<th>Model</th>
<th>Partial</th>
<th>Complete</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-shot LLM</td>
<td>1.29</td>
<td>0.00</td>
</tr>
<tr>
<td>1-shot LLM (cross-app)</td>
<td>74.69</td>
<td>31.67</td>
</tr>
<tr>
<td>2-shot LLM (cross-app)</td>
<td>75.28</td>
<td>34.44</td>
</tr>
<tr>
<td>1-shot LLM (in-app)</td>
<td>78.35</td>
<td>40.00</td>
</tr>
<tr>
<td>2-shot LLM (in-app)</td>
<td>80.36</td>
<td>45.00</td>
</tr>
<tr>
<td>Seq2Act [34]</td>
<td>89.21</td>
<td>70.59</td>
</tr>
</tbody>
</table>

6.1 Implications for Language-Based Interaction

Many language-based UI tasks, such as screen question generation and screen question answering, do not have feasible heuristics alternatives. Therefore, achieving these tasks requires significant effort and budget to create the dataset and develop the model, making fast prototyping and iteration of these interaction capabilities difficult. On the other hand, our feasibility experiments demonstrate that our proposed techniques can achieve decent performances on various UI tasks without requiring sizeable datasets. An important takeaway from our studies is that prototyping novel language interactions on mobile UIs can be as easy as designing an exemplar. As a result, an interaction designer can quickly create functioning mock-ups to test new ideas with end users. Moreover, developers and researchers can explore different possibilities of a target task before investing significant efforts into developing new datasets and models. For example, as discussed in Section 5.2, the summaries in the Screen2Words dataset follow a particular sentence structure, while there are many other ways to summarize a screen. Researchers could write various summary exemplars to prompt an LLM and inspect how well these exemplars work on different screens before spending efforts to develop dedicated datasets and models. Another use case of our approach is end-user programming. When a conversational agent fails to understand or carry out the tasks associated with the user’s commands, prior work has leveraged programming by demonstration technique that asks users to provide tasks demonstrations to teach the agent [28, 32]. Our work offers an alternative way for users to teach the model by specifying the desired outcome of language commands. The model could then use the user input as prompting exemplars to adapt to new tasks.

6.2 Chaining Unit Conversations for Real-World Scenarios

We have proposed four types of unit conversations in our conversation categorization and conducted experiments to demonstrate LLMs’ capability to enable each of them. In real-world use cases, multiple of these unit conversations can be chained together to fulfill complex UI tasks. For example, when a user wants to book a hotel, they can announce the command “Open hotel booking app.” The model would then predict which element on the screen to click. Once the app is open, the model can provide a summary of “Currently showing the search page of the hotel booking app.” Following the summary, the model could ask questions based on the input fields on the screen, such as “Which city do you plan to stay in?”

6 DISCUSSIONS AND LIMITATIONS

We discuss the implications of our investigation, the limitations of our approach, and how future work can build upon our work.
or "When do you plan to check in?" When the search results are shown, the user could ask, "which hotel has the highest ratings?" The model would provide the answer based on the screen information. The conversational interaction enabled by our approach is beneficial for accessibility. It can also be blended with other input/output modalities, such as touch input and screen readers, to offer new possibilities for developing multi-modal interaction. For the next step, we plan to use the proposed method to build an agent that can assist users end-to-end to finish a task. In addition, future work could explore techniques such as model distillation [16] or model compression [15] to achieve a more efficient inference of LLMs.

6.3 Shots, Input Lengths, and Model Performance

Language models often have input length constraints, which limit the number of effective exemplars that can be included in the prompt. The length of a screen HTML has a considerable variance, depending on how much information was conveyed through the view hierarchy and the inherent complexity of the screen. A possible way to avoid exceeding the input length limit is to configure the prompt screens dynamically—when the test screen’s HTML is more extended, one can choose prompt screens with a shorter HTML length. However, imbalanced lengths between prompt and test screens may also lead to inferior performances. On the other hand, while experiments on general NLP tasks showed that including more shots in the prompt increase the performance, our studies showed that including the second example sometimes only marginally improves the performance. In contrast, the first exemplar usually leads to a significant boost. Prior work by Reynolds and McDonnell [42] suggests that the effectiveness of few-shot prompting lies within guiding LLMs to locate specific task locations in the model’s existing space of learned tasks. Therefore, the first shot may be the most helpful, and more examples may only marginally help the model narrow the focus. That said, few-shot prompting and understanding of LLMs’ behaviors are an ongoing research problem in the community. Future work could investigate the trade-off between the number of shots and the length of each shot and their impact on the model performance of different UI tasks.

6.4 Screen Representation

Mobile UI screens contain multiple modalities, including pixels, texts, and even audio when media content is present. A limitation of our investigation is that we only use the view hierarchy information, which is converted to an HTML representation, and leave other modalities unused. This limitation is imposed by the type of input expected by LLMs. While our studies showed LLMs could perform decently on various UI tasks, they could fail on cases requiring information not present in the view hierarchy but available as pixels. For instance, many icons or images on UI screens have missing captions or alt texts (text description of a visual element), as pixels. For instance, many icons or images on UI screens have missing captions or alt texts (text description of a visual element), and LLMs may not be able to perform tasks based on these elements. Moreover, visual information is particularly crucial for some apps, e.g., photo editing tools, and our approach may fall short of enabling conversational interaction based on these apps. Many models have started using multiple modalities, including visual and text information of UIs [7, 35, 46, 53]. Future work could exploit these prior models to generate missing captions or alt-text of elements, which can lead to more comprehensive screen information in the HTML input to LLMs. Our approach can also be extended by leveraging large-scale vision language models such as Flamingo [2] to encode a screen’s visual and structural information for few-shot learning.

7 CONCLUSION

We investigated the feasibility of using large language models to enable various conversational interactions on mobile UIs. We proposed a design space to categorize types of conversation between the users and the agent when they perform UI tasks collaboratively. We proposed a set of prompting techniques to adapt large language models to various conversational UI tasks. To understand the effectiveness of our approach, we conducted feasibility experiments on four language-based UI tasks, one for each unit conversation in our categorization. The results showed that compared to traditional machine learning pipelines that consist of expensive large-scale data collection and model training, one could quickly realize novel language-based interaction using large language models while achieving decent performance.

REFERENCES


A.1 Screen Question-Generation

Given a screen, the agent needs to identify the elements requiring user input and generates corresponding questions.

```
Given a screen, the agent needs to identify the elements requiring user input and generates corresponding questions.

Screen:
<p id=5 class="refundHeaderText"> Check your refund status </p>
<p id=3 class="dash1"> - </p>
<p id=2 class="titleRefund"> Refund Status </p>
<button id=1 alt="Open navigation drawer"> </button >
</div>
</div>

1. id=4 asks for first 3 digits of SSN
2. id=3 asks for middle 2 digits of SSN
3. id=8 asks for last 4 digits of Social Security Number
4. id=9 asks for Filing Status

To help the user proceed with the screen, an agent will ask:
<SOQ>Q: What is your email in case you need to reset the password? (id=5)<EOQ>

Listing 1: 2-shot example prompt for screen question-generation.
```

A.2 Screen Summarization

Given a screen, summarize its purpose.

Screen:
<p id=0 class="alertTitle"> Create password </p>
<button id=1 alt="Open navigation drawer"> </button >
</div>
</div>

It's a create password page and there are 4 input tags, including:
1. id=2 asks for password.
2. id=3 asks to confirm password.
3. id=4 asks for hint.
4. id=5 asks for email address.

To help the user proceed with the screen, an agent will ask:
<SOQ>Q: What is the purpose of the screen? (id=9)<EOQ>

Listing 2: 2-shot example prompt for screen question-generation.
A.3 Screen Question-Answering

Given a mobile screen and a question, provide the answer based on the screen information.

Listing 2: 1-shot example prompt for screen summarization.

A.4 Mapping Instruction to UI Action

Given a screen, an instruction, predict the id of the UI element to perform the instruction.

Listing 3: 1-shot example prompt for screen question-answering

Listing 4: 1-shot example prompt for mapping instruction to UI action.