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@gmail.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 designed prompting techniques to adapt an LLM to mobile UIs. We experimented with four important modeling tasks that address various scenarios in conversational interaction. Our method achieved competitive performance on these challenging tasks without requiring dedicated datasets and training, offering a lightweight and generalizable approach to enable language-based mobile interaction.

CCS CONCEPTS
• Human-centered computing → Human computer interaction (HCI).

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 [6, 13, 22]. 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. The interaction paradigm is particularly useful for users with motor or visual impairments or situationaly impaired when occupied by real-world tasks [48, 54]. Intelligent assistants, e.g., Google Assistant and Siri, have significantly advanced language-based interaction for performing simple daily tasks such as setting a timer. Despite the progress, these assistants still face limitations in supporting conversational interaction in mobile UIs, where many user tasks are performed [31]. For example, they cannot answer a user’s question about specific information displayed on the screen [18]. Achieving such capabilities requires an agent to have a computational understanding of graphical user interfaces (GUIs), which is absent in existing assistants.

Prior research has investigated several important technical building blocks to enable conversational interaction with mobile UIs, including summarizing a mobile screen for users to quickly understand its purpose [50], mapping language instructions to UI actions [37, 41, 44] and modeling GUIs so that they are more amenable for language-based interaction [34, 39, 50, 55, 58]. However, each of them only addresses a limited aspect of conversational interaction and requires considerable effort in curating large-scale datasets and training dedicated models. [37, 38, 40, 50]. Furthermore, there is a broad spectrum of conversational interactions that can occur on mobile UIs, as Todi et al. revealed [49]. Therefore, it is imperative to develop a lightweight and generalizable approach to realize conversational interaction.

Recently, pre-trained large language models (LLMs) such as GPT-3 [7] and PaLM [9] have demonstrated abilities to adapt themselves to various downstream tasks when being prompted with a handful of examples of the target task. Such generalizability is promising to support diverse conversational interactions without requiring task-specific models and datasets. However, the feasibility of doing so is unclear. Little work has been conducted to understand how LLMs, trained with natural languages, can be adapted to GUIs for interaction tasks. Therefore, we investigate in this paper the viability and the how-to of utilizing LLMs to enable diverse language-based interactions with mobile UIs.

We categorized four mobile UI conversational interaction scenarios, which guided our experimental task selection. We developed a set of prompting techniques to prompt LLMs with mobile UIs. Since LLMs only take text tokens as input, we contribute an algorithm to generate the text representation of mobile UIs. Our algorithm uses depth-first search traversal to convert the Android UI’s view hierarchy, i.e., the structural data containing detailed properties of UI elements, into HTML syntax. We conducted comprehensive experiments with four pivotal modeling tasks, including Screen Question-Generation, Screen Summarization, Screen Question-Answering, and Mapping Instruction to UI Action. The experimental results show that our approach achieves competitive performance.
To achieve this, we propose a novel method to represent the UI so that an LLM pre-trained for natural language can efficiently process language-based UI tasks by providing a small number of examples. In summary, the evaluation results from a suite of modeling tasks validate the feasibility and effectiveness of our approach in enabling conversational interaction with mobile UIs.

More broadly, our study demonstrates LLMs’ potential to fundamentally transform the future workflow of conversational interaction design. Using our prompting method, interaction designers and developers can quickly prototype and test novel language interactions with users, saving time and resources before investing in dedicated datasets and models. Our experiments are based on open-source Android UI datasets, including RICO [12], Screen2Words [50], and PixelHelp [37]. We also open-source the code of our algorithm that converts the Android view hierarchy to HTML syntax to allow future work to replicate and build upon our work. Example prompts created using our algorithm can be found in the appendix.

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 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; we open-source the code so others can immediately use them in their work.
- We experimented with four pivotal modeling tasks, demonstrating the feasibility of our approach in adapting LLMs for conversational GUI interaction and potentially lowering the barriers to developing conversational agents for GUIs.

## 2 RELATED WORK

### 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 [38] and Screen Recognition [58] predict semantically meaningful alt-text labels for GUI components. Screen2Words [50] took a step further to predict text summaries that concisely describe the entire screen using multimodal learning. Leiva et al. proposed a vision-only approach to generate templated language descriptions of UI screenshots [28]. Li et al. [37] 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 [31–33, 35]. For example, SUGILITE [31] enables users to create task automation on smartphones by user demonstration and perform the tasks through a conversational interface. KITE [33] 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 adapt large models for specific tasks. However, GPT-3 [7] introduced a new norm for leveraging pre-trained language models for downstream tasks through in-context few-show learning, i.e., prompting. By prompting a pre-trained model with only a few examples, it can generalize to various 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 [52]. While prompting LLMs may not always outperform benchmark models, it provides a lightweight method to achieve competitive performance on various tasks [7, 9]. When the prompt consists of \( N \) pairs of input and output exemplars from the target tasks, it is referred to as \( N \)-shot learning, and providing more shots typically improves performance [7, 9]. Additionally, various prompting paradigms have been proposed to elicit logical reasoning from the language model [24, 51, 53, 59], which is useful for tasks like solving math problems. For example, Chain-of-Thought prompting [53] proposes to use the models to generate intermediate results (i.e., a 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 adapt LLMs to 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 [2, 10, 11, 20, 21, 23, 25–27, 42, 56]. For example, Chang et al. [10] proposed 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 [23] allows users to modify web designs with language commands and uses LLMs to infer the corresponding CSS properties. Lee et al. [25] present CoAuthor, a dataset designed to reveal 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 [2] extracts and leverages the knowledge priors within LLMs to execute real-world, abstract, long-horizon robot commands. Our work represents the first contribution of applying LLMs to enable conversational interactions on mobile UIs.
3 CONVERSATION FOR MOBILE UI TASKS

Conversational interaction with mobile devices is typically embodied as human users exchanging information with a conversational agent. We developed a conceptual framework categorizing four conversation scenarios between users and agents when performing mobile tasks. Our categorization has two dimensions: Initiative and Purpose. As shown in Figure 1, a conversation can be mixed-initiative [17], either initiated by the agent or the user. The purpose of initiated conversation can be either soliciting or providing information. The categorization lays the foundation for determining important modeling tasks to investigate. We focus on studying how LLMs can enable the interaction capabilities of a conversational agent based on a mobile UI, e.g., providing language responses or performing UI actions on behalf of the user. More complex, multi-turn conversations are beyond the scope of this paper and will be explored in future research, which we discuss in section 6.3. We now introduce each conversation category and the associated example modeling tasks, including 1) Screen Question-Generation, 2) Screen Summarization, 3) Screen Question-Answering, and 4) Mapping Instruction to UI Action.

3.1 Agent-initiated Conversation

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

3.1.1 Agent solicits information from user. Mobile UIs often request users to input information relevant to their goals. For example, destination city or travel dates in the scenario of booking a hotel. On mobile UIs, the information request is typically made through input text fields. A conversational agent should be able to similarly solicit essential information from users using natural language. For example, asking users questions like “Which hotel do you want to search for?” or “What is the check-in date of your stay?” We refer to this type of task as Screen Question-Generation since the questions should be generated based on the UI screen contexts.

3.1.2 Agent provides information to user. A key function of GUIs is to convey information to users through visual means. Similarly, conversational agents should be able to articulate the information presented in UIs using language. Given that UIs contain a vast amount of information and user needs vary [49], there are numerous ways to deliver screen information. An example is Screen Summarization [50], which provides a short description of the purpose of the current screen, e.g., “A list of hotel rooms available at W San Francisco”, or “A step-by-step recipes of butter chicken.” The descriptions can help users quickly understand the UI when visual information is unavailable.

3.2 User-initiated Conversation

Users can also initiate conversations to request information or proactively provide information for the agent to process.

3.2.1 User solicits information from the agent. Users should be able to solicit screen information from the agent through conversation. When the information request is done through questions such as “What’s the rate of the hotel room with a king-size bed?”, the agent should respond with appropriate answers such as “$330 per night”,...
We introduce a class of prompting techniques designed to adapt where there is a large amount of text on a screen, such as search view hierarchy data of a mobile screen into LLMs is not feasible. As the first to investigate prompting techniques and their design rationales. 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.

3.2.2 User provides information to the agent. Users can initiate conversations with the agent by providing new information. After receiving the messages, the agent should respond accordingly based on the current UI context using language and/or mobile actions. A representative task of this type of conversation is Mapping Instruction to UI Action. For example, when a user who is presented with a hotel booking screen says, “Click on the Reserve button to book the Fabulous King room,” the agent should understand the user’s intent and the UI contexts in order to click the corresponding button. This type of interaction allows users to control the devices when touch inputs are unavailable.

4 PROMPTING LARGE-LANGUAGE MODELS FOR MOBILE UI TASKS

We introduce a class of prompting techniques designed to adapt LLMs to mobile UIs to enable conversational interaction. 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 data exemplars from the target task [7, 9, 53, 59]. For many natural language processing tasks such as question-answering or translation, few-shot prompting performs competitively with benchmark approaches [7]. However, the methodology for prompting LLMs with mobile UIs has yet to be established and presents several challenges. Firstly, language models can only take text input, while mobile UIs are multimodal, containing text, image, and structural information in their view hierarchy data and screenshots. Moreover, directly inputting the view hierarchy data of a mobile screen into LLMs is not feasible as it contains excessive information, such as detailed properties of each UI element, which can easily exceed the input length limits of LLMs. Furthermore, the mobile UIs can encapsulate the logic of target user tasks [36]. Therefore, 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 capability of LLMs?
2. How to elicit reasoning based on the mobile UIs when needed?

We respond to these questions by describing our proposed prompting techniques and their design rationales. 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 are 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 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 into 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 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 text that is visible to the user.
- **resource_id**: text identifiers that describe the referenced resource.
- **content_desc**: content description that describes the element for accessibility purposes, i.e., 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`
Enabling Conversational Interaction with Mobile UI using Large Language Models
CHI '23, April 23–28, 2023, Hamburg, Germany

Figure 2: Left: An example illustrating the proposed prompt structure. A prompt starts with the preamble, which describes the task. Following the preamble, there will be zero or more task exemplars from the target tasks. Each exemplar consists of an input screen HTML, a chain of thoughts (if applicable), and a task-specific output. Additional exemplars are appended to the end of the previous ones. Right: An illustration of prompting large language models in our use cases. The prompt contains N exemplars from the target tasks. We append the screen HTML of the test screen to the end of the prompt. We then feed the prompt + test screen HTML as input into the LLM. The LLM then generates word tokens in an auto-regressive manner, which helps them capture the exemplars’ pattern to produce a chain of thoughts (if applicable) and task-specific output.

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 [36]; therefore, it is vital for models to perform reasoning when used for conversational interaction. LLMs have demonstrated abilities to reason [2, 7, 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 [53]. 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 our experiments. Chain-of-Thought prompting has been shown effective for tasks that require the model to reason across multiple steps. In our early investigations, we found that this technique does not improve performance for tasks where the output can be directly obtained from the input screen HTML. Therefore, in our experiments, we only used the technique for Screen Question-Generation whose task setup requires multi-step reasoning.

4.3 Prompt Structure

We follow a similar prompt structure proposed in [7]. 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 to 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 we introduced in section 3: 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 [7, 53], 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 [53]. 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 at most, 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 to proceed on the UI.

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 generates 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 helps provide details in the generated questions. Subsequently, 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 followed the common practice of evaluating language generation quality with human ratings. We randomly sampled 400 test screens from the RICO dataset [12]. 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 selected 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 generate questions from the test 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 uses 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 Google to provide ratings. To ensure the quality of the labels, a group of
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 combining the minimum and maximum prices fields into a single question asking about the price range. Elements relevant to each question are highlighted in corresponding colors and numbered indexes.

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

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

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 our results revealed interesting emergent abilities of LLMs. When generating a question for a field, the model considers both the input 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”. The model also blends the screen contexts into the generated questions. 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 exhibits 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?” In practice, combining questions can lead to more efficient communication between users and agents.
was conducted on the full test dataset, with each screen receiving a rating from different annotators.

We test the performance of using LLMs to summarize screens with the zero-shot setting (not providing any exemplar). The model could not generate meaningful summaries in the zero-shot setting. The rating study choose the third option only when it was difficult to judge the differences between the quality of the two summaries. The rating study was conducted on the full test dataset, with each screen receiving three ratings from different annotators.

Table 2: Screen summarization performance on automatic metrics.

<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 [50]</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 [50] 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.

5.2.2 Prompt Construction. We use a preamble of “Given a screen, summarize its purpose.” We did not use chain-of-thought prompting as no intermediate result is needed to be generated for the task. We used N pairs of screen HTML and corresponding summary following the preamble, where N = 0, 1, 2 represents N-shot learning. The output summaries are enclosed by special tags <SOS> and <EOS>, meaning the start and end of a summary, respectively. An example prompt can be found in appendix A.2.

5.2.3 Experiment Setup. We use the Screen2Words dataset [50] to test LLM’s ability to summarize screens. The dataset contains human-labeled summaries for more than 24k mobile UI screens, each with five summary labels. To gauge the quality of our approach, we test the performance of using LLMs to summarize screens with Screen2Words’ test set, consisting of 4310 screens from 1254 unique apps, which was used by the benchmark. We randomly sampled two screens from the dataset and one of their corresponding summaries as the exemplars for prompt construction. We use the same automatic metrics reported in the original paper, including BLEU, CIDEr, ROUGE-L, and METEOR. As prior work [14, 50] has found that automatic scores may not correlate well with human perception of summary quality, we additionally conducted a human evaluation to solicit subjective feedback on the summaries generated by both LLM and the benchmark model Screen2Words [50]. We recruited 37 annotators using the same process specified in section 5.1.3. We presented annotators with a mobile UI screen and two screen summaries during the labeling process. To avoid bias, we randomly assigned the summaries generated by LLM and Screen2Words to be either Summary 1 or Summary 2, without revealing which summary was generated by which model. We instructed annotators to choose the summary that best accurately summarizes the mobile screen. The annotators were given three options: (a) Summary 1, (b) Summary 2, and (c) Equal or Very Similar. They were instructed to choose the third option only when it was difficult to judge the differences between the quality of the two summaries. The rating study was conducted on the full test dataset, with each screen receiving three ratings from different annotators.

5.2.4 Results. Table 2 shows the results of screen summarization on automatic metrics. The model could not generate meaningful summaries in the zero-shot setting (not providing any exemplar). This result is expected as the LLM’s training data may not have covered the task of screen summarization. When provided with the model one exemplar, the performance significantly boosted across all metrics. More examples in the prompt provide marginally higher scores. The average length of summaries generated by our two-shot LLM model was 7.15 words (STD=2.68). In comparison, the average length of summaries generated by the benchmark model was 6.64 words (STD=1.98). Additionally, the LLM used a significantly larger number of unique words, at 3062, compared to the 645 unique words used by the benchmark model. The results indicate that LLM is capable of generating longer summaries with a wider range of language. When evaluating these models against automatic metrics, LLM generally scored less than the benchmark model. This is expected because the benchmark model was trained with the Screen2Words dataset, and the automatic metrics are based on token matching. A model trained on the dataset could achieve high scores by prioritizing the high-frequency words in the dataset. However, many high-frequency words and phrases, such as “display of,” “screen,” and “app”, in the Screen2Words dataset do not meaningfully contribute to the summarization accuracy. As a result, a specialized model that learned to prioritize these frequent words may score well by synthesizing generic summaries, indicating the limitations of existing evaluation metrics.

In contrast, our human evaluation revealed that LLM generates summaries with higher perceived quality than those generated by the benchmark model, Screen2Words. Among all the human annotations, 63.5% of them rated LLM summaries as more accurate, while 28.6% voted for benchmark model summaries. Moreover, 7.9%

![Figure 5: Annotator vote distribution across all test screens. LLM summaries are more accurate than those of the benchmark model for 64.1% of screens across the test set. This choice is unanimous between three labelers for more than half of the screens (52.8%).](image)
Enabling Conversational Interaction with Mobile UI using Large Language Models

Figure 6: Example summaries generated by prompting the LLM with two 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.

Figure 6 shows example 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 top-right screen shows that the LLM composes a longer summary by leveraging the app name, the send file button, and the recipient fax button: “FaxFile app screen where a user can select a file to send via fax and pick recipients.”

We also observed that the LLM’s prior knowledge is beneficial for screen summarization. For example, the bottom-right photo shows
We did not use chain-of-thought prompting as no intermediate work, we focus on the answers that are present in the view hierarchy data, and incorporating screenshot information in the LLM RICO dataset contain inaccurate view hierarchy data [29]. In this study, we observed that the 2-shot setting provided only a modest performance boost over the zero-shot setting. The LLM can already generate answers in aliases equivalent to the ground truth, 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 across the entire dataset. We consider Exact Match answers correct, and those fall within Contains GT and Sub-String of GT relevant. The three metrics are exclusive to each other, and examples of each can be found in Figure 7-right. We compare the LLM with an off-the-shelf text QA model DistilBERT [47]. We use the distilbert-base-cased-distilled-squad implementation on huggingface.co, which achieved 79.6% Exact Match and 87.0% F1 score on the SQuAD dataset [45]. Unlike existing QA models that extract answers from the input text, the LLM may generate answers in aliases equivalent to the ground truth, for example, 9/15 and September 15th. However, our metrics, which rely on string matching, may not detect these aliases and could result in under-reported scores for our approach.

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 DistillBERT 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. DistillBERT 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. 1-shot LLM achieved a 62.9% Micro-F1 score, significantly outperforming the baseline’s score of 37.2%. Once again, we observed that the 2-shot setting provided only a modest performance improvement, with relevant answers reaching 84.5% and Micro-F1 achieving 64.8%. Figure 7-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.

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 across the entire dataset. We consider Exact Match answers correct, and those fall within Contains GT and Sub-String of GT relevant. The three metrics are exclusive to each other, and examples of each can be found in Figure 7-right. We compare the LLM with an off-the-shelf text QA model DistilBERT [47]. We use the distilbert-base-cased-distilled-squad implementation on huggingface.co, which achieved 79.6% Exact Match and 87.0% F1 score on the SQuAD dataset [45]. Unlike existing QA models that extract answers from the input text, the LLM may generate answers in aliases equivalent to the ground truth, for example, 9/15 and September 15th. However, our metrics, which rely on string matching, may not detect these aliases and could result in under-reported scores for our approach.

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 DistillBERT 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. DistillBERT 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. 1-shot LLM achieved a 62.9% Micro-F1 score, significantly outperforming the baseline’s score of 37.2%. Once again, we observed that the 2-shot setting provided only a modest performance improvement, with relevant answers reaching 84.5% and Micro-F1 achieving 64.8%. Figure 7-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.
Enabling Conversational Interaction with Mobile UI using Large Language Models

CHI ’23, April 23–28, 2023, Hamburg, Germany

Figure 7: Left: Example results from the screen QA experiment. The LLM significantly outperforms the off-the-shelf QA model DistillBert (Baseline). Right: Example of the three metrics used to assess the LLM’s performance on screen QA, including Exact Match, Contains Ground Truth, and Sub-String of Ground Truth. For each example question, the UI element related to its ground truth is highlighted with a bounding box with the corresponding color and question index.

"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.

5.4 Mapping Instruction to UI Action

5.4.1 Task Formulation. Given a mobile UI screen and natural language instruction to control the UI, the model needs to predict the id of the 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 [37], 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. We then randomly sampled from the prompt modules to construct the final prompts. We conducted experiments under two conditions: 1) in-app and cross-app. In the former, the prompt contains a prompt module from the same app package as the test screen, while in the latter, it does not. 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 (Table 4) show that the 0-shot setting cannot perform the task 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. Our approach underperforms the benchmark results from the Seq2Act model [37], which was trained on several dedicated datasets with hundreds of thousands of examples. We do not expect prompting LLMs to consistently

<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 [37]</td>
<td>89.21</td>
<td>70.59</td>
</tr>
</tbody>
</table>
outperform benchmarks across all tasks, as the model can only view a handful of data examples and does not update its underlying parameters [7]. Nevertheless, our approach still demonstrated competitive performances for the task using only two examples.

6 DISCUSSIONS AND FUTURE WORK

In this section, we discuss the implications of our investigation, the limitations of our approach, and how future work can build upon our work.

6.1 Implications for Language-Based Interaction

An important takeaway from our studies is that prototyping novel view a handful of data examples and does not update its underlying experiments. Notably, our work offers a set of new techniques and findings for using boost the accuracy of this task. As the first work in this direction, our method still achieved competitive accuracy with only two examples instead of extensive modeling on dedicated datasets like the benchmark. One possible reason for the underperformance is that our work contributes to the literature as

Our approach showed promising results on each task in the feasibility experiments. Notably, our work contributes to the literature as

6.2 Feasibility Experiments

Our approach showed promising results on each task in the feasibility experiments. Notably, our work contributes to the literature as

6.3 Extending to Multi-Screen, Multi-Turn Conversational Interaction

We focus on unit conversation tasks involving single-turn interactions in this work. Future research can build upon our techniques to enable multi-screen [8] and multi-turn [30] conversational interactions. Multi-screen interactions can allow the agent to carry out complex tasks across various screens. For example, to help a user book a hotel room, an agent could start by asking which hotel the user wants to book and then search for that hotel (Figure 1-1: Screen Question-Generation). Once the search results are obtained, the agent could announce a summary of the search result page (Figure 1-2: Screen Summary). After that, the user could then ask for specific information from the search results, such as the room rates (Figure 1-3: Screen Question-Answering), and finally, the user could command the agent to book the desired room (Figure 1-4: Mapping Instruction to UI Actions). Multi-turn interactions can enable the agent to ask for clarifications on ambiguous cases [32] or recover from errors. For example, given the screen in Figure 3 with two emails sent from Google Play, if the user says, "Open the email sent by Google Play", our current approach would likely select one of them directly without clarifying with the users. A sophisticated agent should identify ambiguity and ask the user for clarification, such as, "Do you mean the one sent on Dec. 4th or Nov. 12th?" To achieve multi-screen and multi-turn interactions with the proposed approach, future work will need to investigate methods to store and represent past screens and interactions as contexts [43].

6.4 Shots, Input Lengths, and Model Performance

Our findings indicate that the first prompt exemplar usually significantly boosts LLM performance in mobile UI tasks. In contrast, the inclusion of a second example only results in a marginal improvement. Previous research by Reynolds and McDonell [46] 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 additional examples may provide only marginal benefits in narrowing the model’s focus. Relevantly, language models often have input length constraints, which limit the number of 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. Future work could explore approaches to prevent exceeding the input length limit. One potential approach is to dynamically select prompt screens with different lengths. When the test screen HTML is lengthy, shorter prompt screens could be selected. However, it is unclear whether imbalanced lengths between prompt and test screens would lead to inferior performances. Future research could investigate the trade-off between the number of shots, the length of each shot, and their impact on the model’s performance in different UI tasks.
As modeling techniques advance, the input length restrictions of language models would likely be lifted [57], allowing for more information to be included in prompts. Another direction for future work is to develop methods for condensing screen HTML into a more concise syntax while still achieving comparable performance. In addition, to enable real-time applications, future research may investigate strategies such as model distillation [16] and model compression [15] for more efficient inference of LLMs.

6.5 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 in 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), 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 textual information of UIs [8, 38, 50, 58]. 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 utilizing large-scale vision language models such as Flamingo [4] to encode a screen’s visual and structural information for few-shot learning.

6.6 Stearability and Reliability of LLM Predictions
Our experiments uncovered several promising capabilities of LLMs for mobile UI tasks. For example, the model showcased the behavior of question combination, i.e., merging relevant input fields into a single question, in the Screen Question-Generation task. Furthermore, our study shows that the model can utilize its embedded prior knowledge to provide additional information in the Screen Summarization task. While these capabilities are intriguing and potentially useful, there is currently a lack of direct controls for steering LLMs in terms of when and how these behaviors should occur. Additionally, LLMs, as other natural language generation models, could sometimes produce unintended text, known as hallucinations [19]. This means that the model may produce incorrect or irrelevant information. Our current method does not explicitly address this issue. As the research surrounding LLMs develops, it is crucial to find ways to improve the steerability and reliability of their predictions and behaviors, especially when incorporating LLMs into user-facing technology.

6.7 Generalizing Beyond Mobile UIs
This paper focuses on mobile UIs, but the proposed approach with LLMs can also be applied to other UI types, such as web UIs [1] (which are already in HTML syntax) and popular UI systems like iOS and macOS that possess view hierarchy data or equivalence. However, for more sophisticated and feature-rich UIs like video editors, a direct adaptation of our approach may be challenging due to the difficulty in representing key features in text format and ensuring the representation complies with model length constraints, as highlighted in the previous discussions. Nonetheless, our research provides a solid foundation for further studies to build upon and develop innovative methods that enable language interaction with graphical user interfaces.

7 CONCLUSION
We investigated the feasibility of prompting LLMs to enable various conversational interactions on mobile UIs. By categorizing the conversation scenarios between users and agents during mobile tasks, we identified four crucial UI tasks to study. We proposed a suite of prompting techniques for adapting LLMs to mobile UIs. We conducted extensive experiments with the four selected tasks to evaluate the effectiveness of our approach. The results showed that compared to traditional machine learning pipelines that consist of expensive data collection and model training, one could rapidly realize novel language-based interactions using LLMs while achieving competitive performance.

ACKNOWLEDGMENTS
We thank the anonymous reviewers for their valuable feedback during the R&R process that enhanced the paper’s quality. We appreciate the discussions and feedback from our team members Chin-Yi Cheng and Tao Li. We also acknowledge the early-stage feedback from Michael Terry and Minsuk Chang. Special thanks to the Google Data Compute team for their invaluable assistance in data collection.

REFERENCES
A.1 Screen Question-Generation

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

Screen:

```html
<p id=0 class="alertTitle"> Create password </p>
<div id=1 class="titleLabelDivider" /> 
<div id=2 class="passwordInput"> Crowd3116 </div>
<input id=3 class="confirmPassword"> Crowd3116 </input>
<input id=4 class="hintInput"> c3 </input>
<input id=5 class="emailInput"> appcrawler@gmail.com </input>
<input id=6 class="emailInput"> tvEmailAddressInfo </input> 
This email address will be used to reset your password. 

<button id=7 class="button2"> Cancel </button>
<button id=8 class="button1"> OK </button>
```

Now reasoning starts:

Q: How many input tags are there on the screen?
A: 4
Q: What is the purpose of the screen?
A: Create password.
Q: How many password input tags are there?
A: 2
Q: What input is used to reset password?
A: tvEmailAddressInfo

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, the agent will ask:

Q: What password do you want to create? (id=2)<EQQ>
Q: Could you enter the password again to confirm? (id=3)<EQQ>
Q: What hint do you want to set? (id=4)<EQQ>
Q: What is your email in case you need to reset the password? (id=5)<EQQ>

Screen:

```html
<p id=0 class="IR2SOg"> </p>
<button id=1 alt="Open navigation drawer"> </button>
<input id=2 class="titleRefund"> Refund Status </input>
<input id=3 class="refundAmountInput"> Check your refund once every 24 hours, usually overnight. </input>
```

A: 4
Q: How many input tags are there on the screen?
A: 2
Q: What input is used to input Social Security Number?
A: tvEmailAddressInfo

A  EXAMPLE PROMPTS FOR FEASIBILITY EXPERIMENTS TASKS

We show the prompt examples used in each task from the feasibility experiments. For screen summarization, screen QA, and mapping instruction to UI action, we include 1-shot prompts for simplicity. For the screen question-generation study, we use the 2-shot prompt—one exemplar uses question combination (the second screen), and the other does not. Both exemplars contain a chain of thoughts consisting of intermediate results.

A.1 Screen Question-Generation

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

Screen:

```html
<p id=0 class="alertTitle"> Create password </p>
<div id=1 class="titleLabelDivider" /> 
<div id=2 class="passwordInput"> Crowd3116 </div>
<input id=3 class="confirmPassword"> Crowd3116 </input>
<input id=4 class="hintInput"> c3 </input>
<input id=5 class="emailInput"> appcrawler@gmail.com </input>
<input id=6 class="emailInput"> tvEmailAddressInfo </input> 
This email address will be used to reset your password. 

<button id=7 class="button2"> Cancel </button>
<button id=8 class="button1"> OK </button>
```

Now reasoning starts:

Q: How many input tags are there on the screen?
A: 4
Q: What is the purpose of the screen?
A: Create password.
Q: How many password input tags are there?
A: 2
Q: What input is used to reset password?
A: tvEmailAddressInfo

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, the agent will ask:

Q: What password do you want to create? (id=2)<EQQ>
Q: Could you enter the password again to confirm? (id=3)<EQQ>
Q: What hint do you want to set? (id=4)<EQQ>
Q: What is your email in case you need to reset the password? (id=5)<EQQ>

Screen:

```html
<p id=0 class="IR2SOg"> </p>
<button id=1 alt="Open navigation drawer"> </button>
<input id=2 class="titleRefund"> Refund Status </input>
<input id=3 class="refundAmountInput"> Check your refund once every 24 hours, usually overnight. </input>
```

A: 4
Q: How many input tags are there on the screen?
A: 2
Q: What input is used to input Social Security Number?
A: tvEmailAddressInfo

A  EXAMPLE PROMPTS FOR FEASIBILITY EXPERIMENTS TASKS

We show the prompt examples used in each task from the feasibility experiments. For screen summarization, screen QA, and mapping instruction to UI action, we include 1-shot prompts for simplicity. For the screen question-generation study, we use the 2-shot prompt—one exemplar uses question combination (the second screen), and the other does not. Both exemplars contain a chain of thoughts consisting of intermediate results.
Given a mobile screen and a question, provide the answer based on the screen information.

A.3 Screen Question-Answering

Given a screen, a mobile screen and a question, provide the answer.

Screen:
- Invite for T20 Fans Live Chat
- Add recipients
- Suggestions from Google

Q: What email addresses are there?
A: T, test, appcrawler4@gmail.com

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

A.4 Mapping Instruction to UI Action

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

Screen:
- Apps list
- Search bar

Instruction: Close search
Q: What is the refund amount?
A: (id = 10) asks for the amount of refund.

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

A.2 Screen Summarization

Given a screen, summarize its purpose.

Screen:
- Create new contact
- Add to a contact
- Send SMS

Q: What is the purpose of the screen?
A: Check refund status.

It's a check refund status page and there are 4 input tags, including:
1. id=4 asks for first 3 digits of SSN
2. id=6 asks for middle 2 digits of SSN
3. id=8 asks for last 4 digits of SSN
4. id=10 asks for the amount of refund.

To help the user proceed with the screen, an agent will ask:
- What is your SSN? (id=4, id=6, id=8)<EQ>
- What is the refund amount? (id=10)<EQ>

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

CHI '23, April 23–28, 2023, Hamburg, Germany
Instruction: Open your device's Clock app.