# META-GUI: Towards Multi-modal Conversational Agents on Mobile GUI

**Liangtai Sun**[*], **Xingyu Chen**[*], **Lu Chen**[†], **Tianle Dai, Zichen Zhu** and **Kai Yu**[†]
X-LANCE Lab, Department of Computer Science and Engineering
MoE Key Lab of Artificial Intelligence, AI Institute, Shanghai Jiao Tong University
Shanghai Jiao Tong University, Shanghai, China
{slt19990817, galaxychen, chenlusz}@sjtu.edu.cn,
{daitl2000, jameszhuthethird, kai.yu}@sjtu.edu.cn

## Abstract

Task-oriented dialogue (TOD) systems have been widely used by mobile phone intelligent assistants to accomplish tasks such as calendar scheduling or hotel reservation. Current TOD systems usually focus on multi-turn text/speech interaction, then they would call back-end APIs designed for TODs to perform the task. However, this API-based architecture greatly limits the information-searching capability of intelligent assistants and may even lead to task failure if TOD-specific APIs are not available or the task is too complicated to be executed by the provided APIs. In this paper, we propose a new TOD architecture: GUI-based task-oriented dialogue system (GUI-TOD). A GUI-TOD system can directly perform GUI operations on real APPs and execute tasks without invoking TOD-specific backend APIs. Furthermore, we release **META-GUI**, a dataset for training a **M**ulti-modal conv**E**rsa**T**ional **A**gent on mobile **GUI**. We also propose a multi-model action prediction and response model, which show promising results on META-GUI. The dataset, codes and leaderboard are publicly available[‡].

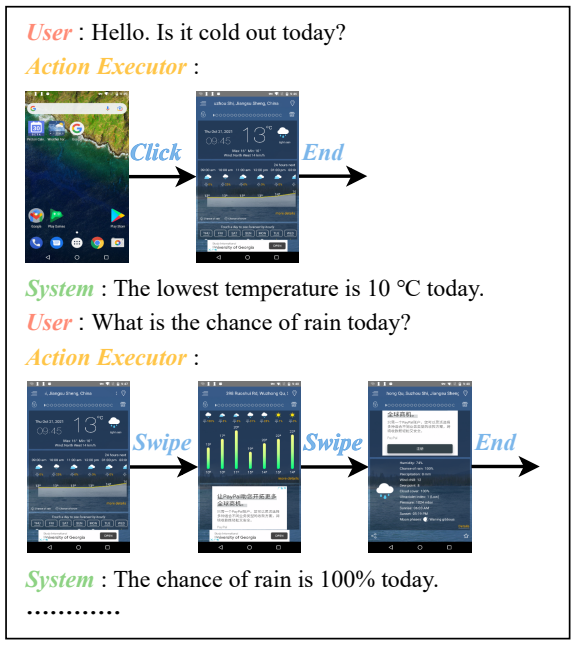



Figure 1: An example of the GUI-based task-oriented dialogue system(GUI-TOD). The Action Executor will execute tasks on GUI and the system will generate a response based on the execution result.

## 1 Introduction

Recent years have witnessed the rapid development of task-oriented dialogue systems (Zhang et al., 2020; Ni et al., 2022; Chen et al., 2022, 2017). They have been widely applied to customer support, booking system and especially intelligent personal assistant. These task-oriented dialogue systems work in a similar pipeline: firstly identify the user intent, then extract necessary information by the process of slot-filling. After getting enough information for the task, the agent will call the backend APIs (provided by APP developers) to fetch information, and then generate a response based on the query result.

There are some drawbacks of this framework. Firstly, TODs rely on publicly accessible APIs or APIs designed for TODs to perform tasks, but such APIs may not exist in real-life APPs, which hinders the application of TODs. Secondly, a system should be customized to recognize the pre-defined API-related slots, which limits the generality.

Consider how humans perform tasks on smartphones They don't need a parametric API but finish tasks by interacting with the GUI (graphical user interface), indicating that GUI is a more general interface. Previous studies explore how to translate natural language commands into GUI operations (Mazumder and Riva, 2021; Pasupat et al., 2018; Xu et al., 2021a). These studies focus on single query and step-by-step operations, while in

[*]Equal contributions.
[†]The corresponding authors are Lu Chen and Kai Yu.
[‡]https://x-lance.github.io/META-GUI-Leaderboard/



*Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 6699 - 6712
December 7-11, 2022 

| Action | Description |
|---|---|
| `Click(item =` $x$`)` | Click the item with index $x$ on the screen. |
| `Swipe(direction =` $x$`)` | Swipe screen towards direction $x$, which includes "up" and "down". |
| `Input(text =` $x$`)` | Input the text $x$ to the smartphone. |
| `Enter( )` | Press the "Enter" button on the keyboard. |
| `Clear( )` | Clear the current input box. |
| `Back( )` | Press the "back" button on the smartphone. |
| `End( )` | Turn has been finished and it will go to Response Generator module. |

Table 1: The actions in our dataset. There are 7 different actions with 3 different parameters.

real scenarios the query would be multi-turn interaction and there is no clear instruction about how to execute the task. Etan (Riva and Kace, 2021) and SUGILITE (Li et al., 2017) are two systems that support learning GUI operations from demonstrations, but these systems are script-based and are sensitive to the change in GUI and workflow. Duplex on the web (Crunch, 2019) can directly operate the website to perform the required task, for example booking a movie ticket. However, it only supports limited websites, and it's more a unified GUI interface than a task-oriented dialogue system that enables general GUI operation.

To this end, we propose the task of GUI-based task-oriented dialogue system (GUI-TOD). It supports multi-turn conversation and direct GUI operation. All tasks would be performed on the GUI of real APPs, which means we no longer need TOD-specific APIs to communicate with APPs, and it would be possible to apply TOD on any APPs. Since there is no available benchmark published, We collect META-GUI, a dataset with dialogues and GUI traces on real Android APPs. A GUI trace is a series of GUI operations, including screenshots, Android view hierarchies as well as actions. Android view hierarchy is an XML-style file, which organizes the content of GUI through a hierarchical structure. It also contains the types of items on the screen and their bounding boxes. An example is shown in Appendix C. When a user requests a task, the system should open the related APP and execute the task through multiple operations on GUI. It requires a comprehensive understanding of GUI structure and interaction logic. An interaction example is shown in Figure 1.

We focus on building an agent with general ability to operate GUI, rather than optimize for specific APPs. Our proposed GUI-TOD system leverages both the visual information and textual information on the screen to predict the next action to be executed and generate the system response. Our experiments show that the GUI-TOD outperforms heuristic baselines by a large margin, with an action completion rate of 82.74%.

Our contributions are followings:

- We propose a GUI-based task-oriented dialogue system, which can perform tasks on mobile APPs through multiple operations on GUI.
- We collect META-GUI, a dataset with dialogues and GUI operation traces serving as the benchmark for the proposed system.
- We conduct thorough experiments on our dataset and validate the importance of multi-modal information and history information. We show that it is a promising task but needs further exploration.

## 2 Task Definition

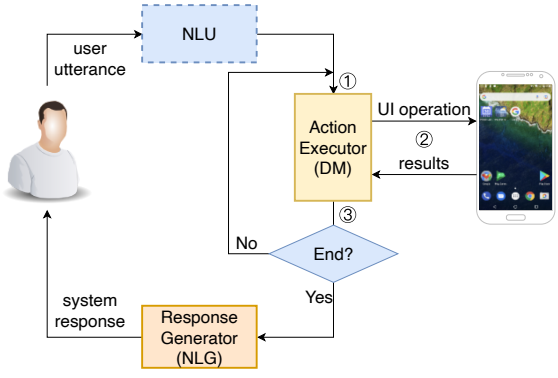



Figure 2: The overview of GUI-based task-oriented dialogue system (GUI-TOD).

The overview of GUI-TOD is shown in Figure 2. It consists of two sub-modules: Action Executor (AE) and Response Generator (RG). The traditional task-oriented dialogue system (Chen et al., 2017; Zhang et al., 2020; Yu et al., 2014) splits the task

into natural language understanding (NLU) (Zhu et al., 2021), dialogue manager (DM) (Chen et al., 2020a; Zhu et al., 2020; Chen et al., 2018, 2019, 2020b), and natural language generation (NLG) (Keskar et al., 2019). We omit the NLU module and directly send user utterances to AE. The AE module has similar features with DM, it executes the requested task by interacting with the GUI for multiple rounds, while DM accomplishes this by calling TOD-specific APIs. The RG module will generate the system response based on the execution results, which is the same as NLG. The process of executing a task is a series of GUI operations, including click, swipe, etc. The task of AE module is action prediction, which aims at predicting the next action to be performed on GUI, and the RG module focuses on generating system's response after executing a task. A major improvement of GUI-TOD is that it does not rely on a pre-defined domain ontology. Conventionally, the DM module will identify a set of slot-value from the user utterance, which serves as the parameter for backend APIs. However, GUI-TOD handles task-specific slot-values during the execution of tasks. When the APP requires a certain input (for example, entering the time and destination), the system can obtain the information by understanding the current user utterance or generating a response for further asking. Compared with CUED actions (Young, 2007) in traditional TOD, actions in GUI-TOD are GUI-related operations rather than communication actions between user and system.

Formally, the action prediction task can be defined as: given the GUI trace and dialogue history, predict the next action to be performed. We define the set of actions that can be performed on the APPs in Table 1. All the actions would take the form of $Action(parameter = *)$. There are seven types of $Action$, including six physical actions: *click, swipe, input, enter, clear, back*, and one virtual action: *end*. The corresponding parameters are listed in Table 1. The $end$ action is the last action for every GUI trace, which means the end of GUI operations. After an $end$ action is generated, the GUI-TOD would move to the RG module. We denote the *j*th action in turn *i* as $\mathcal{A}_{i,j} = (t, p)$, where *t* is the action type and *p* is the corresponding parameter. $\mathcal{S}_{i,j} = (s, v)$ is the *j*th screen in turn *i*, including the screenshot *s* and the view hierarchy *v*. The dialogue in turn *i* is represented as $\mathcal{D}_i = (U_i, R_i)$ where $U_i$ is the $i$th user utterance and $R_i$ is the $i$th system response. The action prediction task is formulated as:

$$\mathcal{A}_{i,j} = \mathcal{F}\left(\mathcal{S}_{1:i,1:j}, \mathcal{A}_{1:i,1:j-1}, \mathcal{D}_{1:i-1}, U_i\right), \quad (1)$$

where $1:i$ means from turn 1 to $i$, $\mathcal{F}$ is a trainable action model, which we discuss in 4.1. The RG module takes the GUI trace and dialogue history as input, then generates a response based on the execution result and context. Denote the set of actions in turn $i$ as $\mathcal{A}_i$, the screens in turn $i$ as $\mathcal{S}_i$, the response generation task is formulated as:

$$\mathcal{R}_i = \mathcal{G}\left(\mathcal{S}_{1:i}, \mathcal{A}_{1:i}, \mathcal{D}_{1:i-1}, U_i\right), \quad (2)$$

where $\mathcal{G}$ is the response generator model, which we discuss in 4.2.

## 3 Meta-GUI Creation

Our dataset consists of two kinds of data: dialogues and GUI operation traces. In each dialogue, user would ask the agent to complete a certain task through multi-turn interaction. Our tasks involve six different domains: weather, calendar, search, taxi, hotel and restaurant. In this paper, we consider APPs that accomplish the same kind of tasks to be in the same domain. To enhance the diversity of our dataset, we use multiple Apps from the calendar and weather domains. The details of APPs are listed in Appendix A.

### 3.1 Collecting GUI traces

We collected our data in two-stage: first we collected GUI traces for existing dialogues, then we collected both dialogues and GUI traces.

In the first stage, we provided dialogues to annotators and instructed them to perform tasks on real APPs. We started from extracting dialogues from the SMCalFlow dataset (Andreas et al., 2020). SMCalFlow contains multi-turn task-oriented dialogues, which is known for complex reference phenomenon that requires a comprehensive understanding of context. We extract dialogues from calendar, weather and search domains. Six annotators were recruited to label the GUI traces. We built a web-based annotation system, which was connected to a real Android smartphone (see Appendix B). Annotators can see the current screen of the smartphone in the system, and control the smartphone by clicking buttons. A dialogue would be shown in the system. Annotators should first read the dialogue, then they were allowed to explore

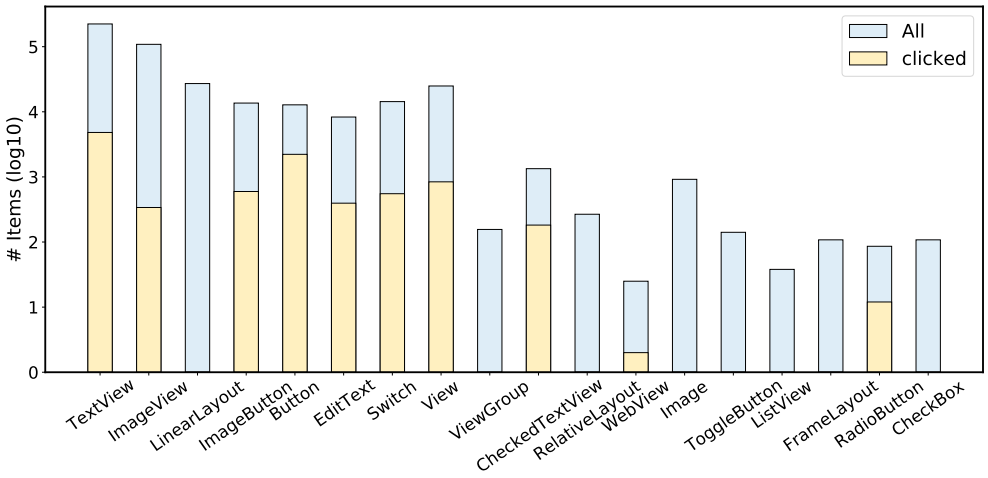



Figure 3: The distribution of the total number of items versus the clicked one for each item type.

how to finish the task (e.g. check the weather) on smartphone. If the task requirement in the dialogue conflicted with the real-world scenario (for example, creating an event in the past), the annotators could change the content of the dialogue to make the task achievable. After they were ready, they need to use the annotation system to record the actual process of executing the task. Each operation would be recorded, and the screenshot after each operation was also saved together with the view hierarchy.

In the second stage, we collected dialogues and GUI traces for domains of hotel, restaurant and taxi. Because there are no available dialogues of these domains in previous datasets, we asked annotators to write new dialogues. We selected three experienced annotators from the last stage. Different from the last stage, the annotator was shown a task objective, which was generated randomly from all available conditions in APPs. The annotators should act as user and system alternatively to write dialogues according to the task objectives. To avoid annotators writing short and simple dialogues, we added constraints about the number of turns and the behaviors in dialogue, e.g. adding a condition or changing a condition. An example of the generated target is shown in Appendix E. After writing dialogues, the annotators should also record the corresponding GUI operation traces for each turn, which is the same as the last stage.

### 3.2 Data Review

After annotation, we manually reviewed the data. The checklist includes: whether the recorded GUI traces match the dialogues, whether there are invalid operations due to the system error or misoperation, and whether there are redundant operations in the GUI trace. We manually fixed annotations that only have small mistakes, and discarded the task requiring significant modification. The dialogue level pass rate is about 63.6%, and finally we got 1125 dialogues in total. For more information, please refer to Appendix D.

### 3.3 Post-processing

The dialogues collected in the second state were created by three annotators, which lack diversity in expression. Therefore, we published a dialog rewritten task on AMT* (Amazon Mechanical Turk) to polish the dialogues.

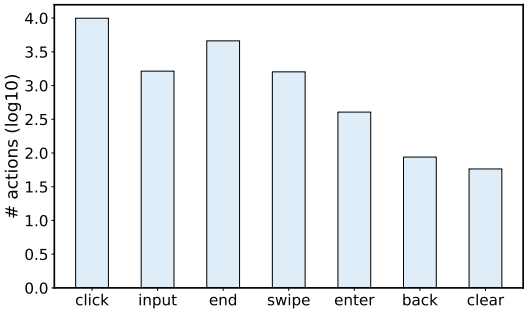



Figure 4: The distribution of actions.

During GUI trace annotation, some APPs can not obtain valid Android hierarchy. To handle this problem, we used the online Optical Character Recog-

*https://www.mturk.com/

nition (OCR) service, provided by Baidu Cloud [†], to detect all texts on the image with their corresponding positions and generate a pseudo layout file.

We extract items from screen using the corresponding layout file. An item is a clickable leaf node. Similar to (Zhou and Li, 2021), we consider an item to be clickable if its `clickable` attribute is true or its parent node is clickable. An item consists of text content, item type and bounding box. We extract the text content of an item by looking at its `text` property first. If it is empty, we use its `content-desc` attribute, otherwise we would use the `resource-id` property. Based on the extracted items, we can locate the target item for the `click` action by comparing the click position and the bounding boxes of items.

### 3.4 Data Analysis

The total number of dialogues in our dataset is 1125, including 4684 turns. The average number of images for each turn is 5.30, and the average number of words for each utterance is 8. On average, there are 23.80 items for each image, and the item text length is 2.48 words. The distribution of item types is shown in Figure 3. We also provide an example for each item type in Appendix F. It is clear that `TextView` and `ImageView` are the two most frequent type, which indicates that our dataset is informative.

The distribution of actions is listed in Figure 4. The `click` is the most frequent action, while `clear` is the least action for the reason that only a small number of tasks require clearing the current input box. For `click` action, we further compute the type distribution of target items, which is shown in Figure 3. `TextView` and `Button` type are mostly clicked, while there are 8 item types never been operated. This implies that the item types may supply some hints for predicting the target items. Besides, the average numbers of words for `response` and `input` action are 9 and 3 respectively.

## 4 Model Design

The overview of our system is illustrated in Figure 5. It's composed of four components: encoder, image feature extractor, multi-modal information fusion module and the output module. The output module can be the Action Module or the Response Module.

### 4.1 Action Model

We call the combination of encoder, image feature extractor, multi-modal information fusion module and the Action Module as Action Model, which is used to predict the next GUI action based on the history. Next, we will describe these modules respectively. For simplify, for the screen history we only consider the last screen here. We will discuss adding more screen histories later.

**Encoder** The input of encoder consists of two parts: dialog history $\{\mathcal{D}_{1:i-1}, U_i\} = \{w_1, ..., w_n\}$ and texts in the items $\{m_{1,1:l_1}, \ldots, m_{k,1:l_k}\}$. Items are extracted from the last screen, $k$ is the number of items and $l_i$ is the length of the $i$th item's text:

$$\begin{aligned} X &= \{w_{1:n}; m_{1,1:l_1}, \ldots, m_{k,1:l_k}\}, \\ \mathbf{H} &= \text{TransformerEncoder}(X), \end{aligned} \tag{3}$$

where $\mathbf{H} = [\mathbf{D}; \mathbf{M}]$ and $\mathbf{D} = \{\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_n\}$ represents encoder outputs of the dialogue history, $\mathbf{M} = \{\mathbf{m}_{1,1:l_1}; \ldots; \mathbf{m}_{k,1:l_k}\}$ represents encoder outputs of item texts.

**Image feature extractor** Given a screenshot and its corresponding layout file, we use Faster R-CNN (Ren et al., 2015) to extract the feature map. Then we apply ROI pooling based on the bounding box of each item, and get the item-level image features $\mathbf{I} = \{\mathbf{I}_1, ..., \mathbf{I}_k\}$.

**Multi-modal information fusion module** Given the encoder output and the regional image feature extracted above, we concatenate them together. The text features from one item $\mathbf{m}_{i,1:l_k}$ are concatenated with the same item feature $\mathbf{I}_i$, and the $\mathbf{w}_{1:n}$ are concatenated with zeros. Then we use a Transformer encoder with $M$ layers to fuse the multi-modal features. For each layer, to enhance the image information, we will concatenate the image features and the output from the last layer again to form the input for the next layer.

**Action Module** For the Action model, we need to predict the action type and its corresponding parameters. As shown in Table 1, there are 7 action types with 3 different parameters. We show some examples of parameter predictions in Appendix G.

We use the encoder output of the `[CLS]` token for action type prediction. We apply a feed-forward

[†] https://cloud.baidu.com/

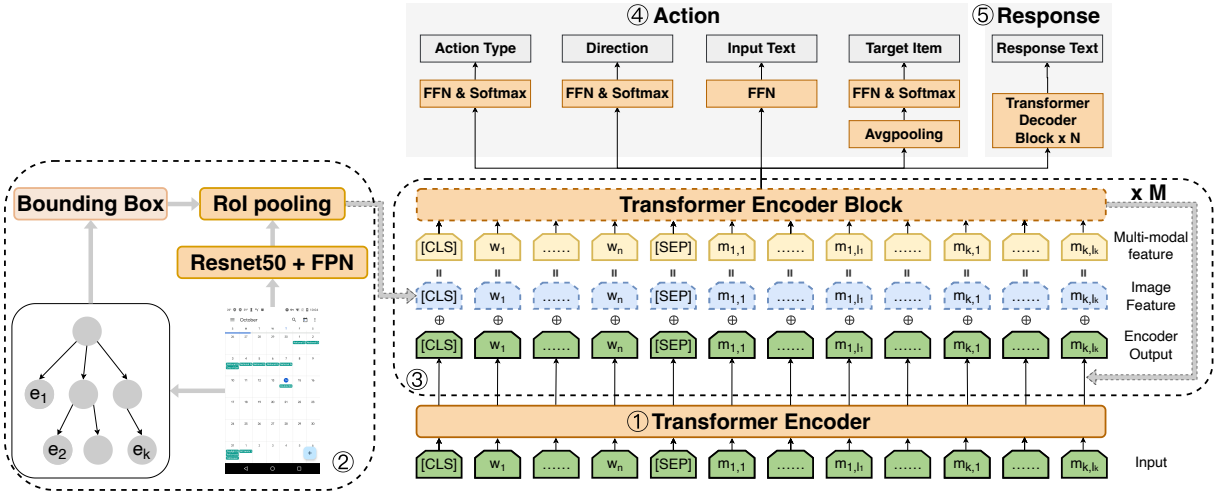



Figure 5: The illustration of our proposed model. There are five parts in this figure: (1) encoder; (2) image feature extraction; (3) multi-modal information fusion; (4) the Action Module; (5) the Response Module.

network followed by a Softmax layer to predict the action type:

$$\mathbf{p}_a = \text{Softmax}(\text{FFN}_1(\mathbf{E}_{[\text{CLS}]})), \quad (4)$$

where $\mathbf{p}_a$ is the probability distribution of action, and FFN represents the Feed-Forward Network.

For the action parameter, we use three different classifiers:

1) *Input Text Prediction* We assume that the input to the APPs must be part of the user utterance, so we formulate the prediction of input text as a span prediction task. We use two classifiers to predict the begin and end positions in the dialogue:

$$\mathbf{p}_{ds} = \text{FFN}_2(\mathbf{D}), \mathbf{p}_{de} = \text{FFN}_3(\mathbf{D}), \quad (5)$$

where the $\mathbf{p}_{ds}$ and $\mathbf{p}_{ds}$ are the probability of start and end position respectively.

2) *Target Item Prediction* The target item classifier is based on the encoding outputs of items. We first computed the item representation by applying average pooling on the encoding outputs, then we use a feed-forward layer to compute the probability of selecting an item followed by a Softmax layer:

$$\begin{aligned} \bar{\mathbf{m}}_i &= \text{Avgpooling}(\mathbf{m}_{i,1:l_i}) \;\; 1 \le i \le k, \\ \bar{\mathbf{m}} &= [\bar{\mathbf{m}}_1, \ldots, \bar{\mathbf{m}}_k]\,. \\ \mathbf{p}_m &= \text{Softmax}(\text{FFN}_4(\bar{\mathbf{m}})), \end{aligned} \quad (6)$$

where $\mathbf{p}_m$ is the probability distribution of items.

3) *Direction Prediction* The direction classifier is a two-classes classification layer for the direction *up* and *down*:

$$\mathbf{p}_d = \text{Softmax}(\text{FFN}_5(\mathbf{E}_{[\text{CLS}]})), \quad (7)$$

where $\mathbf{p}_d$ is the probability distribution of swipe direction.

**Adding history information** According to the task definition, besides dialogue histories, we can still use action histories and screen histories. To verify this, we add them to the action model. For action histories, we regard action types as special tokens and add them to the dictionary. We concatenate the most recent $H$ action types $\{t_{1:H}\}$ before the dialogue history as input:

$$X = \{t_{1:H}; w_{1:n}; m_{1,1:l_1}, \ldots, m_{k,1:l_k}\}, \quad (8)$$

where $X$ stands for the input of Encoder, $t$ represents the action type.

For screenshot histories, we encode all the screenshot in a recurrent way. Assume $\hat{\mathbf{I}}_i = [\mathbf{I}_{i,1}, ..., \mathbf{I}_{i,k}]$ is the image feature for $i$th screenshot, and $\bar{\mathbf{I}}_i$ is the history image feature for time step $i$. We compute $\bar{\mathbf{I}}_{i+1}$ by:

$$\begin{aligned} \bar{\mathbf{I}}_{i+1} = \text{Attn}(\mathbf{W}_1\hat{\mathbf{I}}_{i+1}, \mathbf{W}_2\bar{\mathbf{I}}_i, \mathbf{W}_3\bar{\mathbf{I}}_i), \\ 1 \le i \le H-1, \end{aligned} \quad (9)$$

where $\bar{\mathbf{I}}_1 = \hat{\mathbf{I}}_1$, $H$ is the length of history, Attn is the attention mechanism (Vaswani et al., 2017), and $\mathbf{W}_*$ are trainable parameters. We use the $\bar{\mathbf{I}}_H$ to replace the image features in Figure 5.

### 4.2 Response Model

The Response Model aims to generate the response to user. We use the Response Module as the output module and the other parts are the same as Action Model. Considering the prediction of response is mainly decided by the execution results

and dialogues, we do not use action histories for the Response Model. For the Response Module, we use a Transformer Decoder with N layers:

$$\mathbf{R} = \text{TransformerDecoder}([\mathbf{D}; \mathbf{M}]), \quad (10)$$

where $\mathbf{R}$ represents the predicted response text.

## 5 Experiment

### 5.1 Data Preprocess

| | Train | Dev | Test |
|---|---|---|---|
| # dialogues | 897 | 112 | 116 |
| # turns | 3692 | 509 | 483 |
| # data | 14539 | 1875 | 1923 |

Table 2: Dataset Statistics

We process the dataset in the granularity of action. Each data point takes as input the screenshot history, action history, dialogue history and predicts the action to be performed. We obtained 18337 data points in total, and we randomly divide the data into the training set, development set and test set with the ratio of 8:1:1. The data statistics are shown in Table 2.

### 5.2 Experiment Setup

We train our baselines on the training set and select the best models on the dev set based on the Action completion rate. We use pretrained BERT (Devlin et al., 2019), LayoutLM (Xu et al., 2020) and LayoutLMv2 (Xu et al., 2021b) as our encoder models. [‡] BERT is pretrained on pure text corpus by masked languages modeling task, while LayoutLM and LayoutLMv2 are pretrained on scanned documents by masked visual-language modeling task and incorporate image features.

We use a batch size of 4 and fine-tune for 8 epochs. We use Adam optimizer with the learning rate of 1e-5. For Response Model, the number of Transformer Decoder Block is 4. Furthermore, we use three heuristic methods in our experiments:

**Random** We randomly predict action type and its corresponding parameters.

**Frequency Method (FM)** We first calculate the frequency of each action type and its corresponding parameters. Then, we apply the results to the development set and generate the prediction according to the frequency.

[‡]There are some pre-trained models about GUI understanding, like ActionBERT (He et al., 2021) and UIBERT (Bai et al., 2021). But they are not open-source.

**Most Frequent Method (MFM)** Similar to the frequency method, we generate the prediction with the most frequent result.

For the evaluation, we use completion rate for action prediction. We first define two completion rate metrics: action completion rate and turn completion rate. One action is regarded as completed only if the action type and its parameters are correctly predicted. And if all actions in the same turn are completed, then the corresponding turn will be considered completed. For action type prediction, item prediction and direction prediction, we use accuracy. For input prediction, we use token level exact match and F1. And we use BLEU score to evaluate the Response Model.

### 5.3 Experiment Result

The experiment results of the Action Model are listed in Table 3. We can find that the deep learning methods outperform the heuristic methods by a large margin, which is expected. Comparing the results of BERT backbone and LayoutLM backbone, we find that BERT model yields better performance. The reason is that LayoutLM model was pre-trained on a scanned document image dataset, and there exists a large gap between the Android GUI and the scanned document images. Furthermore, we can find that LayoutLMv2 performs worse than LayoutLM. We hypothesize that LayoutLMv2 uses early-fusion method, which will bring more noises. We can also find that adding multi-modal information to BERT leads to a better performance ($52.08\% \rightarrow 53.96\%$), and the improvements are mainly from the action type prediction, target item prediction and swipe direction prediction. The reason why adding images would help is that the image information contains some action histories that cannot be represented by text. For example, when filtering conditions on hotel reservations, the conditions selected in the previous action can be seen through the image (as a highlighted text), but they can not be reflected through text. An example is illustrated in Appendix H. Besides, the image information can help the model to locate the item more accurately. For example, for a screen with multiple radio buttons, since the BERT model does not take the item position as input, the model cannot distinguish the corresponding button for each option by only textual input. However, we also find that the performance of input text prediction degrades after adding image information. We

| Method | Information | | | Action | | | | | | Turn CR |
|---|---|---|---|---|---|---|---|---|---|---|
| | mm | act_h | scr_h | Action Type Acc. | Input EM | Input F1 | Item Acc. | Direction Acc. | CR | |
| Random | | | | 14.02 | 8.72 | 17.96 | 9.08 | 51.26 | 5.37 | 3.99 |
| MFM | | | | 53.71 | 14.02 | 37.78 | 16.58 | 89.31 | 8.91 | 0.00 |
| FM | | | | 37.48 | 6.65 | 14.02 | 9.94 | 81.51 | 10.00 | 6.76 |
| LayoutLMv2 | ✓ | | | 85.60 | 47.37 | 70.76 | 64.38 | 92.95 | 64.48 | 36.88 |
| LayoutLM | | | | 82.22 | 83.04 | 90.56 | 71.98 | 94.87 | 67.76 | 38.12 |
| BERT | | | | 87.52 | **93.57** | **97.24** | 82.84 | 93.59 | 78.42 | 52.08 |
| +mm | ✓ | | | 88.35 | 92.98 | 96.42 | 84.51 | 94.23 | 80.45 | 53.96 |
| +act_h | | ✓ | | 88.87 | 91.81 | 94.86 | 84.23 | **95.51** | 80.97 | 55.42 |
| +scr_h | ✓ | | ✓ | 89.86 | 90.06 | 95.30 | 84.32 | 94.87 | 81.54 | 55.62 |
| m-BASH | ✓ | ✓ | ✓ | **90.80** | 91.23 | 96.42 | **85.90** | 94.23 | **82.74** | **56.88** |

Table 3: The experiment results of the Action Model on the test set. **Acc.**: accuracy. **EM**: Exact Match. **F1**: F1 score. **CR**: completion rate. **MFM**: Most Frequent Method. **FM**: Frequency Method. **mm**: use the multi-modal information fusion module to add image information. **act_h**: add action histories. **scr_h**: add screenshot histories.

assume that BERT itself can successfully model text information, but adding visual information will affect the model's ability to understand text.

We further verify the importance of history information by adding action histories and screenshot histories. From the experiment results, we find that adding history information to BERT can improve the performance ($52.08\% \rightarrow 55.42\%$ after adding action history to BERT, $53.96\% \rightarrow 55.62\%$ after adding screenshot history to BERT+mm). Adding action histories leads to greater performance improvement, which means action sequence is a more effective way to represent history. The screenshots contain higher-level history information, but the screenshot changes a lot before and after operation (sometimes one click may change the screen completely), which will bring difficulties to the information fusion.

Finally, we add all information, including multi-modal information, action histories and screenshot histories, to the BERT model and get the m-BASH (**m**ulti-modal **B**ERT with **A**ction histories and **S**creenshot **H**istories), which results in the state-of-the-art performance ($56.88\%$).

The results of the Response Model are shown in Table 4. BERT outperforms LayoutLM and LayoutLMv2 by a large margin, which is consistent with the results of Action Model. We also find that adding multi-modal information and screenshot histories can improve performance, which means the model leverage the information from history to generate response.

| Method | Response BLEU score |
|---|---|
| Random | 0.0071 |
| MFM | 0.0929 |
| FM | 0.0788 |
| LayoutLM | 0.5043 |
| LayoutLMv2 | 0.5820 |
| BERT | 0.6219 |
| +mm | 0.6224 |
| +scr_h | **0.6311** |

Table 4: The experiment results of Response BLEU score on the test set.

### 5.4 Generality

According to the design of our system, it does not need to pre-define API-related slots, therefore our system has a strong generality and can be easily adapted to new APPs. To demonstrate this, we re-partition our dataset as followings:

**app generality** Since we use multiple apps in weather domain and calendar domain, we use the data from one APP as the test set, and the other data forms the training set.

**domain generality** We use the data from one domain as the test set, and the other data forms the training set.

We evaluate the performance of m-BASH on these datasets. The results are shown in Table 5. We can find that our system can still obtain a reasonable performance, and the results of app generality

experiments are even comparable to the main experiment results of LayoutLM. This result shows that common operation logic does exist in APPs, and our system can gain a general comprehension of GUI operations. It is easily applied to a new app or a new domain without modification, which shows the effectiveness and potential of our system.

| Data Domain of Test Set | Action Completion Rate (%) | Turn Completion Rate (%) |
|---|---|---|
| *app generality* | | |
| an app of weather | 56.45 | 45.71 |
| an app of calendar | 69.84 | 23.17 |
| *domain generality* | | |
| weather | 41.96 | 21.04 |
| calendar | 62.39 | 19.20 |
| search | 59.40 | 16.24 |
| taxi | 37.68 | 21.72 |
| restaurant | 30.26 | 15.42 |
| hotel | 31.24 | 16.26 |

Table 5: The results of generality experiments.

## 6 Related Work

### 6.1 Natural Language Commands on GUI

Executing natural language commands on GUI is getting research interests recently. Some studies focused on semantic parsing (Mazumder and Riva, 2021; Pasupat et al., 2018; Xu et al., 2021a), whose task is mapping the natural language query to the operations on websites. Google Duplex (Crunch, 2019) can operate websites to finish tasks like booking movie tickets or making restaurant reservations. However, it only supports limited websites and it's more a unified interface than a general dialogue system with GUI operating ability. Our proposed dataset contains real-world APPs and aims at training models with general GUI understanding.

### 6.2 Programming by Demonstration on GUI

Programming by Demonstration (PbD) systems focus on learning GUI tasks from human demonstration (Riva and Kace, 2021; Li and Riva, 2021, 2018; Li et al., 2019). SUGILITE (Li et al., 2017) records user's operations on GUI and generates a script for the learned task. APPINITE (Li et al., 2018) proposed to add descriptions for ambitious actions to enhance the robustness of the generated script. These systems generate scripts based on handcrafted rules and XML analysis, which is sensitive to GUI changes and exceptions. In this work, we aim to build a robot that can work with general mobile GUI, rather than repeating operations.

### 6.3 Visual Dialogue

More and more researchers combine CV and NLP into the dialogue system and are involved inß a more challenging task, visual dialogue(Le and Hoi, 2020; Agarwal et al., 2020; Le et al., 2020). It can be seen as a multi-step reasoning process over a series of questions (Gan et al., 2019). Gan et al. (2019) updated the semantic representation of the question based on the image and dialogue history. Wang et al. (2020) proposed VD-BERT, a simple yet effective framework of unified vision-dialog Transformer that leverages the pre-trained BERT language models for Visual Dialog tasks. Visual dialogue focuses on understanding the image contents. Besides this, our tasks also require understanding the interactions between UIs.

## 7 Conclusion

In this paper, we proposed the task of GUI-based task-oriented dialogue system, which replaces the traditional TOD-specific API calls with GUI operations on real APPs. The advantage is that intelligent agents can perform tasks without the need of backend TOD-specific APIs and it doesn't rely on a domain-specific schema, which means it can be applied to a new domain easily. We collect META-GUI, a dataset with dialogues and GUI traces to serve as a benchmark. Our model shows promising results on the dataset, and we hope this work could stimulate more advanced methods on GUI-TOD. In the future, we will explore how to better incorporate GUI traces into our model and build the GUI semantics based on interactions.

## Limitations

We propose a GUI-based task-oriented dialogue system, which can perform GUI operations on real APPs to complete tasks. To verify the validity of the system, we collect META-GUI dataset, which contains dialogues and GUI operation traces. In real scenarios, an agent may not know how to complete the task presented by the user. In these cases, an agent might reply "It's too hard for me.", or something like this, which are not included in our dataset. In the future, we will augment the dataset

to include such cases. Furthermore, the models we used are too large to be applied in mobile phones. It is important to compress the models, which we will attempt in the future.

## Acknowledgments

We sincerely thank the anonymous reviewers for their valuable comments. This work has been supported by the China NSFC Projects (No.62120106006, No.62106142), Shanghai Municipal Science and Technology Major Project (2021SHZDZX0102), CCF-Tencent Open Fund and Startup Fund for Youngman Research at SJTU (SFYR at SJTU).

## References

Shubham Agarwal, Trung Bui, Joon-Young Lee, Ioannis Konstas, and Verena Rieser. 2020. History for visual dialog: Do we really need it? In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 8182–8197, Online. Association for Computational Linguistics.

Jacob Andreas, John Bufe, David Burkett, Charles Chen, Josh Clausman, Jean Crawford, Kate Crim, Jordan DeLoach, Leah Dorner, Jason Eisner, et al. 2020. Task-oriented dialogue as dataflow synthesis. *Transactions of the Association for Computational Linguistics*, 8:556–571.

Chongyang Bai, Xiaoxue Zang, Ying Xu, Srinivas Sunkara, Abhinav Rastogi, Jindong Chen, and Blaise Agüera y Arcas. 2021. Uibert: Learning generic multimodal representations for ui understanding. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, pages 1705–1712. International Joint Conferences on Artificial Intelligence Organization. Main Track.

Hongshen Chen, Xiaorui Liu, Dawei Yin, and Jiliang Tang. 2017. A survey on dialogue systems: Recent advances and new frontiers. *Acm Sigkdd Explorations Newsletter*, 19(2):25–35.

Lu Chen, Cheng Chang, Zhi Chen, Bowen Tan, Milica Gašić, and Kai Yu. 2018. Policy adaptation for deep reinforcement learning-based dialogue management. In *2018 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pages 6074–6078. IEEE.

Lu Chen, Zhi Chen, Bowen Tan, Sishan Long, Milica Gašić, and Kai Yu. 2019. Agentgraph: Toward universal dialogue management with structured deep reinforcement learning. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 27(9):1378–1391.

Lu Chen, Boer Lv, Chi Wang, Su Zhu, Bowen Tan, and Kai Yu. 2020a. Schema-guided multi-domain dialogue state tracking with graph attention neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 7521–7528.

Zhi Chen, Lu Chen, Xiaoyuan Liu, and Kai Yu. 2020b. Distributed structured actor-critic reinforcement learning for universal dialogue management. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 28:2400–2411.

Zhi Chen, Jiabao Ji, Lu Chen, Yuncong Liu, Da Ma, Bei Chen, Mengyue Wu, Su Zhu, Xin Dong, Fujiang Ge, Qingliang Miao, Jian-Guang Lou, and Kai Yu. 2022. Dfm: Dialogue foundation model for universal large-scale dialogue-oriented task learning. *arXiv preprint arXiv:2205.12662*.

Tech Crunch. 2019. Google is bringing ai assistant duplex to the web.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Zhe Gan, Yu Cheng, Ahmed Kholy, Linjie Li, Jingjing Liu, and Jianfeng Gao. 2019. Multi-step reasoning via recurrent dual attention for visual dialog. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 6463–6474.

Zecheng He, Srinivas Sunkara, Xiaoxue Zang, Ying Xu, Lijuan Liu, Nevan Wichers, Gabriel Schubiner, Ruby Lee, and Jindong Chen. 2021. Actionbert: Leveraging user actions for semantic understanding of user interfaces. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 5931–5938.

Nitish Shirish Keskar, Bryan McCann, Lav R Varshney, Caiming Xiong, and Richard Socher. 2019. Ctrl: A conditional transformer language model for controllable generation. *arXiv preprint arXiv:1909.05858*.

Hung Le and Steven C.H. Hoi. 2020. Video-grounded dialogues with pretrained generation language models. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 5842–5848, Online. Association for Computational Linguistics.

Hung Le, Doyen Sahoo, Nancy Chen, and Steven C.H. Hoi. 2020. BiST: Bi-directional spatio-temporal reasoning for video-grounded dialogues. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1846–1859, Online. Association for Computational Linguistics.

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

Toby Jia-Jun Li, Igor Labutov, Xiaohan Nancy Li, Xiaoyi Zhang, Wenze Shi, Wanling Ding, Tom M Mitchell, and Brad A Myers. 2018. Appinite: A multi-modal interface for specifying data descriptions in programming by demonstration using natural language instructions. In *2018 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pages 105–114. IEEE.

Toby Jia-Jun Li, Marissa Radensky, Justin Jia, Kirielle Singarajah, Tom M Mitchell, and Brad A Myers. 2019. Pumice: A multi-modal agent that learns concepts and conditionals from natural language and demonstrations. In *Proceedings of the 32nd Annual ACM Symposium on User Interface Software and Technology*, pages 577–589.

Toby Jia-Jun Li and Oriana Riva. 2018. Kite: Building conversational bots from mobile apps. In *Proceedings of the 16th Annual International Conference on Mobile Systems, Applications, and Services*, pages 96–109.

Yuanchun Li and Oriana Riva. 2021. Glider: A reinforcement learning approach to extract ui scripts from websites. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 1420–1430.

Sahisnu Mazumder and Oriana Riva. 2021. Flin: A flexible natural language interface for web navigation. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2777–2788.

Jinjie Ni, Tom Young, Vlad Pandelea, Fuzhao Xue, and Erik Cambria. 2022. Recent advances in deep learning based dialogue systems: A systematic survey. *Artificial Intelligence Review*, pages 1–101.

Panupong Pasupat, Tian-Shun Jiang, Evan Zheran Liu, Kelvin Guu, and Percy Liang. 2018. Mapping natural language commands to web elements. In *EMNLP*.

Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. 2015. Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems*, 28.

Oriana Riva and Jason Kace. 2021. Etna: Harvesting action graphs from websites. In *The 34th Annual ACM Symposium on User Interface Software and Technology*, pages 312–331.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008.

Yue Wang, Shafiq Joty, Michael Lyu, Irwin King, Caiming Xiong, and Steven C.H. Hoi. 2020. VD-BERT: A Unified Vision and Dialog Transformer with BERT. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 3325–3338, Online. Association for Computational Linguistics.

Nancy Xu, Sam Masling, Michael Du, Giovanni Campagna, Larry Heck, James Landay, and Monica Lam. 2021a. Grounding open-domain instructions to automate web support tasks. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1022–1032.

Yang Xu, Yiheng Xu, Tengchao Lv, Lei Cui, Furu Wei, Guoxin Wang, Yijuan Lu, Dinei Florencio, Cha Zhang, Wanxiang Che, Min Zhang, and Lidong Zhou. 2021b. LayoutLMv2: Multi-modal pre-training for visually-rich document understanding. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 2579–2591, Online. Association for Computational Linguistics.

Yiheng Xu, Minghao Li, Lei Cui, Shaohan Huang, Furu Wei, and Ming Zhou. 2020. Layoutlm: Pre-training of text and layout for document image understanding. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1192–1200.

Steve Young. 2007. Cued standard dialogue acts. *Report, Cambridge University Engineering Department, 14th October*, 2007.

Kai Yu, Lu Chen, Bo Chen, Kai Sun, and Su Zhu. 2014. Cognitive technology in task-oriented dialogue systems: Concepts, advances and future. *Chinese Journal of Computers*, 37(18):1–17.

Zheng Zhang, Ryuichi Takanobu, Qi Zhu, MinLie Huang, and XiaoYan Zhu. 2020. Recent advances and challenges in task-oriented dialog systems. *Science China Technological Sciences*, pages 1–17.

Xin Zhou and Yang Li. 2021. Large-scale modeling of mobile user click behaviors using deep learning. In *Fifteenth ACM Conference on Recommender Systems*, pages 473–483.

Su Zhu, Lu Chen, Ruisheng Cao, Zhi Chen, Qingliang Miao, and Kai Yu. 2021. Few-shot nlu with vector projection distance and abstract triangular crf. In *CCF International Conference on Natural Language Processing and Chinese Computing*, pages 505–516. Springer.

Su Zhu, Jieyu Li, Lu Chen, and Kai Yu. 2020. Efficient context and schema fusion networks for multi-domain dialogue state tracking. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 766–781, Online. Association for Computational Linguistics.

## A Details of Apps

We list the information of applications used in Table 6. To ensure the diversity of our dataset, we use 4 apps for weather domain, 3 apps for calendar domain, and 1 app each for the last 4 domains. We also list the number of turns belonging to each app. The total number of turns is larger than the actual number of turns, since that one turn may involve several Apps.

| Domain | Package | #Turn |
|---|---|---|
| Weather | com.dailyforecast.weather | 182 |
| | com.accurate.weather.forecast.live | 291 |
| | com.graph.weather.forecast.channel | 115 |
| | com.channel.weather.forecast | 129 |
| Calendar | com.simplemobiletools.calendar | 81 |
| | me.proton.android.calendar | 777 |
| | com.google.android.calendar | 52 |
| Search | com.google.android.googlequicksearchbox | 1616 |
| Taxi | com.ubercab | 750 |
| Restaurant | com.yelp.android | 947 |
| Hotel | com.booking | 942 |

Table 6: The information of Apps. The total number of turns is larger than the actual number of turns because some turns involve several APPs.

## B Annotation System

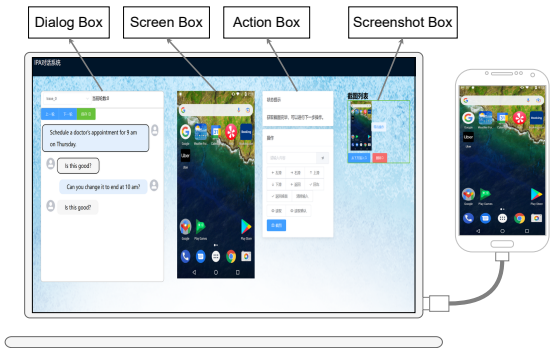



Figure 6: The illustration of our Annotation System.

The annotators can see dialogues in the Dialog Box and the current screen of smartphone in the Screen Box. Action Box proves buttons to control the smartphone, and the Screenshot Box records and displays the operation process.

## C Example of View Hierarchy

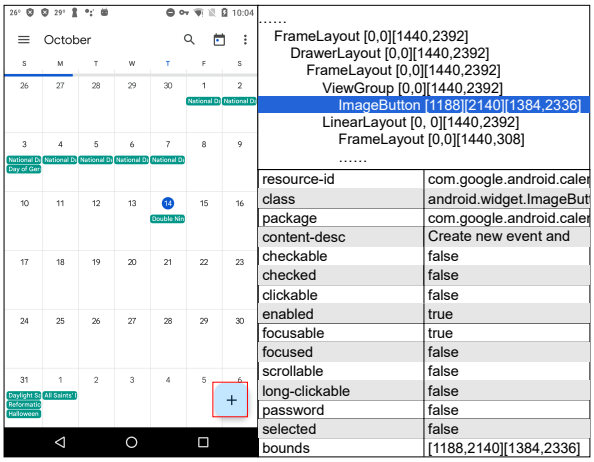



Figure 7: An example of the View Hierarchy for a given screen. The "+" button with a red border on the left-hand side corresponds to the highlighted element in the view hierarchy on the right-hand side.

## D Data Review

After annotation, we manually reviewed the data. The checklist includes: (1) whether the recorded GUI traces match the dialogues: we will check whether the GUI operations match the tasks proposed by the users, for example, whether the scheduled time is correct. (2) whether there are invalid operations due to the system error or misoperation: during annotation, some annotators may click a wrong position or swipe the screen mistakenly. The annotation system may sometimes run into failure. (3) whether there are redundant operations in the GUI trace: for example, some annotators may take screenshots of the same screen multiple times.

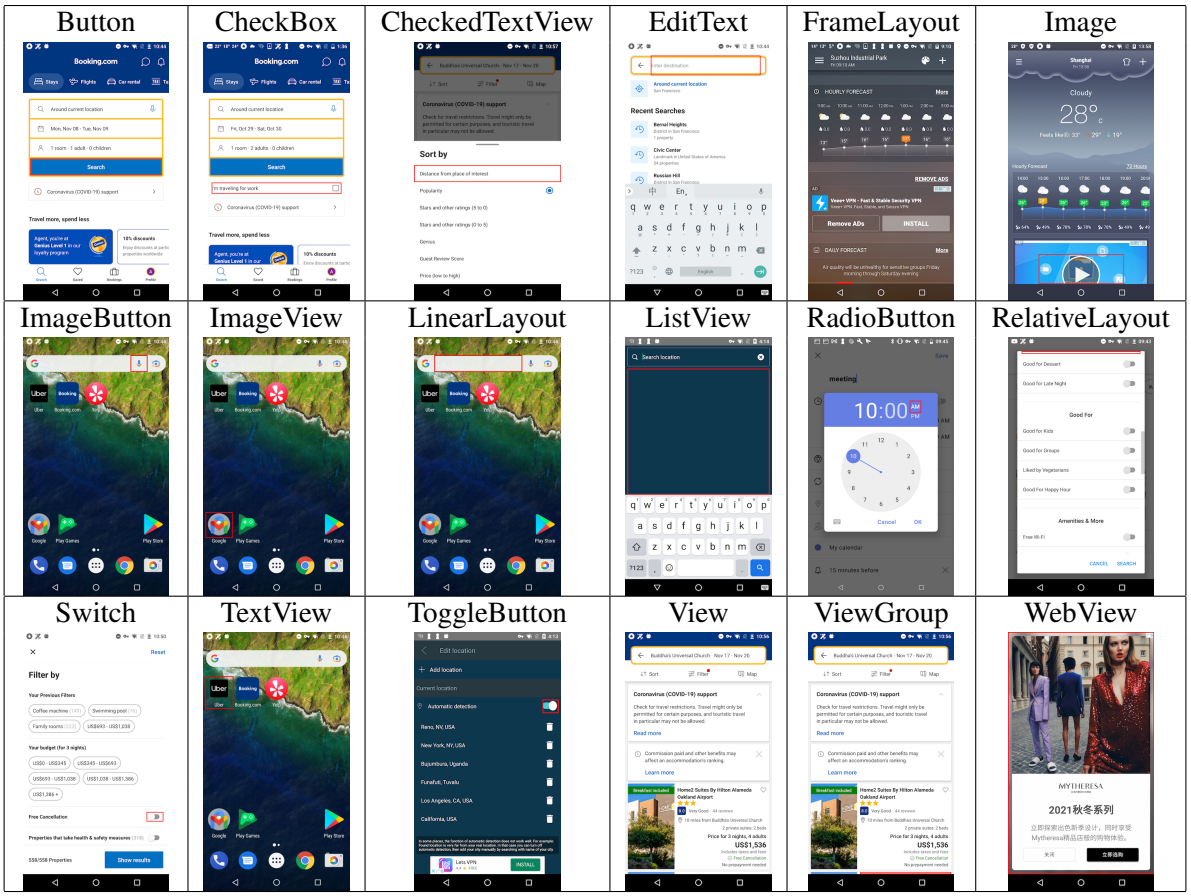



Table 7: Examples of Item types.

## E Example of the generated target during collecting

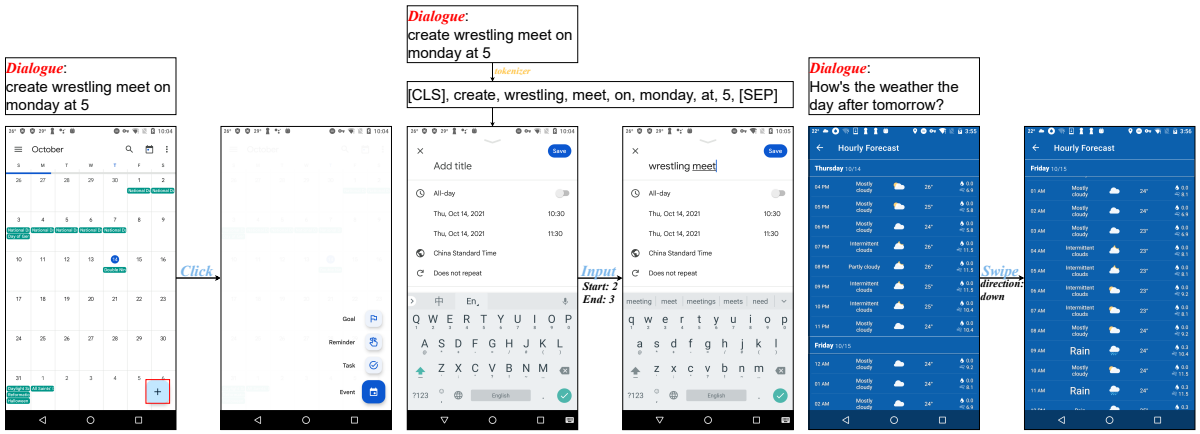
**Dialogue Target :**
You want to take a taxi from "Philz Coffee" to "Flash Sport Fishing Charters of San Francisco"
Time : 17:45
Price : please set price according to the actual situation

**Condition :**
Change: You wanted to take to taxi to "San Francisco Bay Trail" originally.

Figure 8: An example of the generated target.

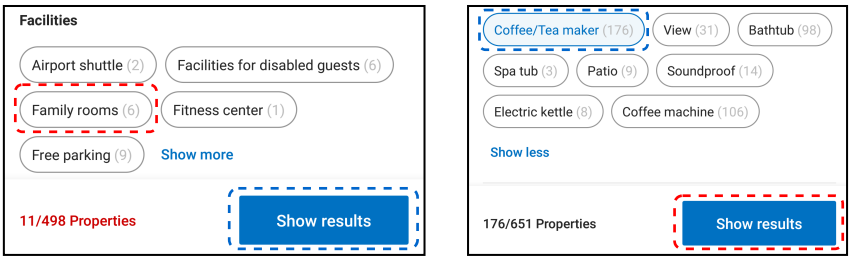
## F Examples of Item types

We list an example for each of the item type in Table 7. There are 18 kinds of item types in total. And the corresponding items are highlighted with a red border.




(a) Target Item prediction (b) Input Text prediction (c) Direction prediction

Figure 9: Examples of parameter predictions.




**User**: Oh, forgot to mention, i need a family room.

**User**: I need non-smoking rooms and Coffee/Tea maker.

Figure 10: Case study. The predictions of m-BASH are marked by red boxes, which are the true answers, while the predictions of the BERT backbone model are marked by blue boxes.

## G Examples of parameter predictions

We show some examples of parameter predictions in Figure 9. Figure 9(a) shows an example of the prediction of target item. The left part shows the current screenshot, where the target item is highlighted with a red border. And the right part shows the screenshot after clicking the target item. Figure 9(b) shows an example of input text prediction. We first split the dialog into the token level, and then predict the text span. Figure 9(c) shows examples of direction prediction.

## H Case study

To further show the importance of multi-modal information and history information, we select two samples, whose action type is *click*, from our dataset and mark the predicted target items made by the BERT backbone model and m-BASH respectively. The result is shown in Figure 10. The predictions of m-BASH are marked by red boxes, which are the true answers, while the predictions of the BERT backbone model are marked by blue boxes. It can be found that the reason why BERT backbone model makes mistakes is that it cannot distinguish whether the conditions are selected or not from text, which can be compensated by images and history information.