UGIF-DataSet: A New Dataset for Cross-lingual, Cross-modal Sequential actions on the UI

Sagar Gubbi Venkatesh
Google
gubbi@google.com

Partha Talukdar
Google
partha@google.com

Srini Narayanan
Google
srinin@google.com

Abstract

Help documents are supposed to aid smartphone users in resolving queries such as "*How to block calls from unknown numbers?*". However, given a query, identifying the right help document, understanding instructions from the document, and using them to resolve the issue at hand is challenging. The user experience may be enhanced by converting the instructions in the help document to a step-by-step tutorial overlaid on the phone UI. Successful execution of this task requires overcoming research challenges in retrieval, parsing, and grounding in the multilingual-multimodal setting. For example, user queries in one language may have to be matched against instructions in another language, which in turn needs to be grounded in a multimodal UI in yet another language. Moreover, there isn't any relevant dataset for such a task. In order to bridge this gap, we introduce **UGIF-DataSet**[1], a multi-lingual, multi-modal UI grounded dataset for step-by-step task completion on the smartphone, containing 4,184 tasks across 8 languages. The instruction steps in UGIF-DataSet are available only in English, so the challenge involves operations in the cross-modal, cross-lingual setting. We compare the performance of different large language models for this task and find that the end-to-end task completion rate drops from 48% in English to 32% for other languages, demonstrating significant overall headroom for improvement. We are hopeful that UGIF-DataSet and our analysis will aid further research on the important problem of sequential task completion in the multilingual and multimodal setting.

1 Introduction

Smartphone users often struggle to navigate the UI to get things done. This problem is particularly acute in developing countries due to varying literacy levels, high cost of phone ownership, etc. (Ranjan, 2022). Many of the tasks users struggle with are documented as frequently asked questions (FAQs) on support sites[2] with step-by-step instructions describing what the user should do on the UI. We explore the problem of harnessing such help documents to create step-by-step tutorials overlaid on the phone UI as an instance of cross-lingual, cross-modal sequential action prediction.

To create step-by-step tutorials on the UI using help documents, research challenges in several natural language processing components including retrieval, parsing, and grounding have to be overcome. But no relevant dataset exists for this task in the multilingual setting. We build on prior work in the NLP community in this area (Li et al., 2020a) and extend it in the multilingual and multimodal directions. We collect a new multi-lingual, multi-modal UI grounded dataset called UGIF-DataSet to evaluate how well models can predict sequential actions on the phone UI. The dataset consists of 523 how-to queries per language and for each query, step-by-step instructions in English and a sequence of UI screenshots and actions that show how to complete the task. Each how-to query and UI sequence is available in 8 languages. An outline of the structure of this dataset is shown in Fig. 1.

The tutorial task poses both multi-lingual and multi-modal challenges. Many smartphone users are bilingual and ask queries in their native language, but the help documents are often available only in English. Hence the need for cross-modal, cross-lingual retrieval. Furthermore, users may use a non-English UI / System language which necessitates cross-lingual UI grounding to map the instruction steps in English to UI screens containing different languages. While current multi-modal models have tended to focus on tasks related to a single image, such as caption generation (Alayrac

[1] pronounced with a soft-g: U-JIF

[2] https://support.google.com

Findings of the Association for Computational Linguistics: NAACL 2024, pages 1390–1399

et al., 2022) or grounding the user's command to a UI element on the screen (Li and Li, 2022), the tutorial task introduces another challenge by requiring the model to perform a sequence of actions across UI screens while referencing a help document. Finally, since the UI changes often, the help documents are often out-of-date, which introduces the additional difficulty of utilizing potentially unreliable help instructions to complete the task.

We propose an initial uni-modal approach that splits this task into **retrieval**, **parsing**, and **grounding** and use existing large language models (Chowdhery et al., 2022; Feng et al., 2020) as a baseline to explore the challenges in this task and estimate the headroom available for improvement. The contributions of this work are as follows:

- We release `UGIF-DataSet`[3], a new multi-lingual, multi-modal dataset of how-to queries and sequences of UI screens and actions recorded by human annotators (Fig. 1). This is the first such multi-modal dataset of its kind.
- We evaluate the parsing of step-by-step how-to instructions with large language models and UI grounding with multi-lingual BERT sentence embedding (LaBSE).
- Our results indicate that there is considerable room to improve performance, especially in non-English languages.

UGIF-DataSet Structure

```
[
  {
    "query": "How to block calls from unknown numbers?",
    "query_i18n": {
      "hi": ["...", "..."],
      ...
    },
    "query_i18n_speech": {
      "hi": ["...<wav_file>..."],
      ...
    }
    "instruction_txt": "1. Open the Phone app...",
    "instruction_mark": "1. Open the 'Phone' app...",
    "macros": "tap('Phone');...",
    "url": "https://support.google.com/accessibility/…",
    "url_content": "<!DOCTYPE html>..."
    "ui_screens": [
      {
        "ui_xml": "<?xml ...",
        "ui_screenshot": "...<png_file>...",
        "ui_elements": [
          {
            "ui_str": "Phone",
            "ui_str_i18n": {
              "kn": "ದೂರವಾಣಿ",
              "hi": "फ़ोन",
              ...
            },
            "ui_bbox": [0, 0, 10, 20],
            "ui_type": 0, // button, switch, etc.
          },
          ...
        ],
        "ui_action": 0 // index into "ui_elements"
      },
      ...
    ]
  },
  ...
]
```

Figure 1: An outline of the `UGIF-DataSet` dataset, which consists of 523 pairs of tutorials and sequences of UI screens and actions (Section 1).

2 Related Work

Natural Language Instruction Following for UI navigation: There have been several previous efforts at natural language conditioned UI navigation for desktop operating systems (Branavan et al., 2009, 2010; Xu et al., 2021) and image editing applications such as Adobe Photoshop (Manuvinakurike et al., 2018). More recently, there has been work on grounding natural language instructions to mobile user interfaces for automatically generating videos of help articles (Zhong et al., 2021). Our work is an enhanced and updated successor to the `PixelHelp` dataset released in Li et al. (2020a) with voice and text queries in eight languages, instruction steps in English, and UI screens in eight system languages.

Imitation learning and Reinforcement learning for UI navigation: One can think of broadly two approaches to building a UI navigation agent: (a) scaling horizontally by building an agent that can handle a few simple tasks like searching for something, deleting an item, etc. that are useful across many different apps, and (b) scaling vertically by exposing a greater depth of functionality but only for a few applications. Li (2021) takes the former approach and uses behavior cloning and reinforcement learning to train agents for two specific skills: to install the specified app from the Play Store and another agent to find the search box in any app. To enable reinforcement learning research on Android UIs, Toyama et al. (2021) introduces `AndroidEnv`, an open source platform for training RL agents. Similar to that, `WorldOfBits` is an open platform for training web navigation agents (Shi et al., 2017; Liu et al., 2018). In our work, we take the lat-

[3]UGIF-DataSet is available under CC-BY 4.0 International license at https://github.com/google-research/google-research/tree/master/ugif

ter approach of exposing deeper functionality of a few popular apps by relying on help articles in the Android support site. We chose this because new users often ask goal oriented questions that require greater knowledge about how to navigate a particular app. Moreover, app developers often provide FAQs with common tasks in mind, so we can exploit the support pages to create UI grounded tutorials for new users.

Pre-training for UI tasks: In the past few years, there has been a paradigm shift in deep learning towards pre-training on broad unlabelled datasets and fine-tuning on task specific data. Bai et al. (2021); He et al. (2021) pre-train a transformer model on a large number of screenshots obtained by crawling apps in smartphones in a manner similar to web crawling. Since our focus is on multilingual UI screens, we chose to use the pre-trained LaBSE (Feng et al., 2020) for UI grounding, but utilizing broad UI data will be critical for future improvements.

Large language models: Large language models (LLMs) pre-trained on large corpora of text scraped from the web have shown remarkable few-shot generalization capability (Chowdhery et al., 2022; Brown et al., 2020). We employ LLMs for parsing help articles but not for UI grounding since we prefer to do it on-device for privacy reasons.

Language grounding in human-robot interaction: Language guided robot actions for human-robot interaction (Lynch and Sermanet, 2020; Venkatesh et al., 2021) is a broadly related problem. However, taking actions on real robots is much more complex with uncertain outcomes, whereas precise actions can be performed on the UI with near certainty. As a result, the difficulty with UI grounded interactions is less about sensing and actuation and more about understanding user intent and navigating the app by understanding its structure using external resources such as support pages.

Icon and widget captioning: Although Android allows developers to provide content description for images, not all app developers do so. To support a wide range of apps, it becomes necessary to recognize icons and widgets (Li et al., 2020b; Baechler and Sunkara, 2021). In our work, all the apps provide the necessary description, so icon captioning is not necessary.

3 UGIF-DataSet: A New Multilingual Multimodal UI-grounded Instruction Following Dataset

To build and evaluate an Android UI navigation agent that can teach users how to use the UI, we collect a new multi-lingual, multi-modal UI grounded dataset called `UGIF-DataSet`. It is a corpus of how-to queries in text and speech in multiple languages, instruction steps for each tutorial paired with sequences of UI screens and actions as the tutorial is completed by human annotators on Android devices (Fig. 1).

The Pixel Help support pages provide step-by-step instructions for performing common tasks on Android. This is an example task: "*How to block unknown numbers?*" for which the instruction text is "*1. Open your Phone app 2. Tap More. 3. Tap Settings and then Blocked numbers. 4. Turn on Unknown*". We crawl the Android support site and extract the tutorial steps using simple rules that look for ordered lists under a header. Annotators translate and speak out loud the how-to query. They also parse the tutorial steps to a sequence of macros in Table 1. Additionally, for each tutorial task, annotators are asked to operate a virtual Android device to carry out the steps in the tutorial while the screen of the device and the annotator's actions are recorded. Just before each action taken by the annotator is forwarded to the virtual device and executed using `UIAutomator` (Android, 2022), we record a screenshot of the device, the view hierarchy in XML, and the action taken by the annotator at that step.

We used an internal platform for crowd-sourcing annotations from annotators in India, Kenya, and Mexico. Annotators were screened by asking qualifying questions with multiple-choice answers. Only those who successfully answered the qualifying questions were allowed to participate in the data collection. The qualifying questions tested linguistic capability: simple questions to gauge whether they can understand written content and speech in the target language. There was no Android UI or task specific training involved. The task description explained the purpose of this data collection effort and how it would be used for research. To protect the privacy of the annotators, the annotator ID is not included in this dataset release. The task design was reviewed by privacy, ethics, and legal committees. The price set for each task was in compliance with local laws.

The manual annotation process for collecting UI

Macro	Function
`tap(`e`)`	Taps on the UI element specified in the argument (e)
`toggle(`e`, val=True)`	Finds the UI element in the argument (e) and then searches for the nearest Switch element and taps on that
`home()`	Presses the home button in Android
`back()`	Presses the back button
`prompt(`a`)`	Requests the user to take some action (a) and waits until an action is performed

Table 1: List of all macros that can be generated from instruction steps (Section 3).

Dataset characteristic	Value
# of tutorials / language	523
# of train samples / language	152
# of dev samples / language	106
# of test samples / language	265
Number of languages	8
Total # of UI screens	3312
Avg # of UI screens / tutorial	6.3
% of tutorials failing due to UI drift	29.9%
Max # of tasks / annotator	50

Table 2: `UGIF-DataSet` statistics (Section 3).

screens from the Android emulator scales linearly with the number of UI languages. To mitigate this, we collect UI screens from annotators only in English and search for each UI string in the resources directory of the app's APK and replace it with the translation provided by the developer in the APK wherever it is available. If a translation is unavailable, we default to English. A typical UI screen has a mixture of strings in English and other languages, but this is distinct from code mixing where two languages are used in a single sentence.

The `UGIF-DataSet` dataset includes tasks in the following apps: Settings, Google One, Gmail, Play Store, Contacts, Messages, Chrome, Maps, Camera, Google Photos, Google Earth, and Files (Table 2). It differs from the `PixelHelp` dataset (Li et al., 2020a) in the following ways. It:

- Contains UI elements in seven non-English languages: Hindi, Kannada, Marathi, Gujarati, Bengali, Swahili, Spanish.
- Is a multi-modal dataset that includes not only the view hierarchy of the screens but also a screenshot at each step of the execution.
- Does not assume that the UI element is visible on the screen. The annotator is allowed to scroll and find the UI element referred in the instruction text.
- Includes samples where the instruction text is outdated and does not correspond to the current version of the UI. In such cases, annotators can either adapt the instructions to the current UI or declare an error if they are unable to complete the task.

4 Model

UGIF has three components: Retrieval, Parsing, and Grounding (Fig. 2). Based on text or speech input, the most relevant how-to instruction in English is retrieved and then parsed to generate macros. These macros are executed on the Android device by grounding them in the UI (Alg. 1).

Algorithm 1 UGIF end-to-end description

```
steps ← retrieve_howto(user_query)
macros ← parse(steps)
i ← 0
while i < len(macros) do
    macro ← macros[i]
    action ← ground(macro, screen)
    if action ≠ SCROLL then
        i ← i + 1
    end if
end while
```

Retrieval We use Google Cloud Speech[4] as an off-the-shelf speech recognizer to convert speech to text. A multilingual sentence embedding model (Feng et al., 2020) is used to obtain a vector corresponding to the query, which is then used to retrieve the most similar how-to by cosine similarity of the how-to page title in the UGIF-DataSet corpus.

Parsing The parsing model takes how-to instructions and generates a sequence of macros (Table 3). We tried various language models such as PaLM (Chowdhery et al., 2022), GPT-3 (Brown et al., 2020), T5 (Raffel et al., 2020), and UL2 (Tay et al., 2022)) to generate the macro given the instruction text. For parsing with finetuned models, the how-to instruction steps was provided as input (see Fig. 5).

[4]https://cloud.google.com/speech-to-text

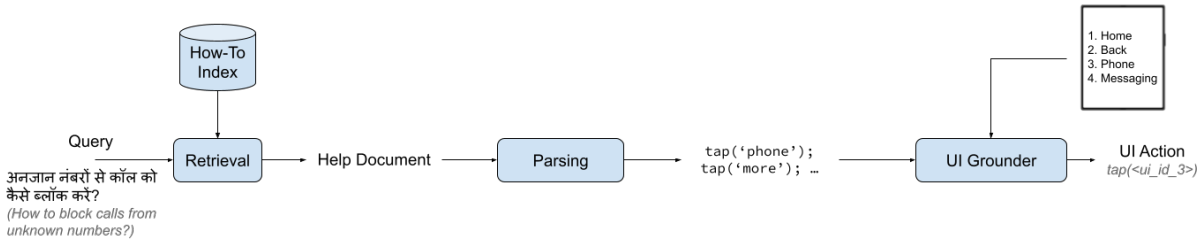

Figure 2: Our initial approach using Large Language Models (LLM) for UI Grounded Instruction Following. The user's query "*How to block calls from unknown numbers?*" is matched against how-to articles in the index to find the closest matching help document. The instruction steps in the help document are parsed using an LLM to a sequence of macros like `tap()`, `toggle()`, `home()`, etc. that must be executed on the UI. The phone grounder grounds each macro in the UI and selects the UI element that the user must act on (Section 4).

Instruction text	**Macro sequence**
Open the Phone app. Tap Recents.	`tap("Phone");` `tap("Recents");`

Open the Settings app. Tap Network & Internet. Turn off wi-fi.	`tap("Settings");`
	`tap("Network & Internet");`
	`toggle("wi-fi", False);`

Table 3: Sample instructions and corresponding macro sequences (Section 4).

In the case of the few-shot prompted models, the prompt preamble "Use these examples to generate code." was followed by few-shot examples in the same format (Text: ...\nCode: ...\n) as shown in Fig. 5.

Grounding The grounding model takes a macro, potentially with arguments, as input along with the current UI screen and performs a series of actions on the UI to complete the task specified by the macro. The macros in our setup are described in Table 1.

For both `tap()` and `toggle()`, it is necessary to locate the UI element being referred to in the argument of these macros. i.e., we are given a macro with its argument referring to a UI element and a list of UI elements currently visible on the screen, and we must decide which element to pick (or to not pick at all and scroll for a better match). For finding the closest matching UI element, we experiment with jaccard similarity, UiBERT (Bai et al., 2021), and multi-lingual BERT sentence embedding (LaBSE) (Feng et al., 2020). The jaccard similarity between a UI element and the referring expression is measured by splitting the words in the UI string and the referring expression and finding the jaccard similarity between these two sets. The LaBSE model generates embeddings for entire sentences, which we utilize to compute embeddings for each UI element and also for the input referring expression in the macro. The cosine similarity between the embeddings for the referring expression and the UI element is used as a scalar measure of the similarity between the arugment to the macro and the UI element. When the app developer has not provided translation for some UI element, the system defaults to showing the English label for that UI element. Since the multilingual sentence embedding model (LaBSE) does not rely on language identification, we can utilize the sentence embedding without regard to the language in the UI element.

We use a scrolling threshold T to decide whether to scroll or to accept a UI element currently on the screen. If the similarity metric is less than T, we choose to scroll down looking for a better match, whereas if the similarity metric is above T, the best matching UI element is chosen for interaction (either tapping or toggling). The appropriate value for T is determined through experimentation on the development set. Likewise, we also use UiBERT to generate embeddings for all the UI elements on the screen along with the input referring expression, but with UiBERT we introduce an additional "Not found" UI element that the model is trained to choose if the scroll action is taken.

For the tapping macro, it is sufficient to look for the UI element most similar to the argument in the macro. However, for the toggle macro, when using LaBSE embeddings we first find the UI element referred to by the argument to the `toggle()` macro, and then look for an Android Switch element nearby in the view hierarchy (Fig. 3). This works as long as the app is using the standard An-

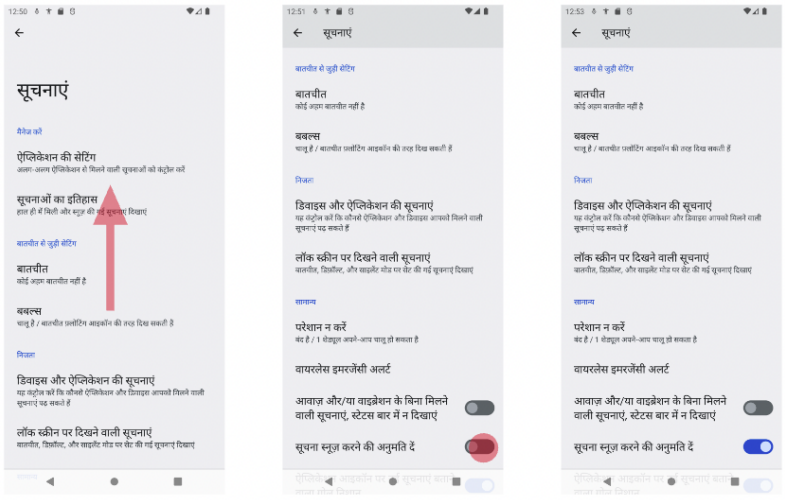

Figure 3: A sample sequence of UI screens and actions resulting from the execution of the macro: `toggle("Allow notification snoozing", True)`. The UI grounding model recognizes that none of the UI elements is a sufficiently close match to the string in the argument of the macro, scrolls down, finds a match, and taps on the nearest switch to turn it on (Section 4).

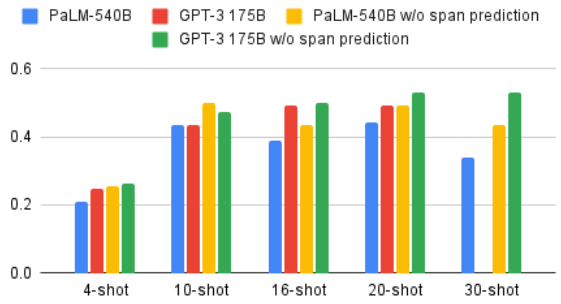

Figure 4: Parsing accuracy on the development set of `UGIF-DataSet` (Section 5.2).

droid Switch element and a straightforward XML layout of the mobile UI where the text field is close to the Switch element. Nevertheless, such heuristics are brittle and could be resolved by multimodal models which we leave for future work.

5 Experiments

The `UGIF-DataSet` dataset contains manually annotated oracle parses (macro sequences) for each how-to instruction text. We measure parsing accuracy by looking for an exact match between the generated parses and the oracle parses.

The dataset also contains manually annotated screen-action sequences for the entire how-to, but it does not have such sequences for each macro. So, to evaluate the grounding model, we consider the end-to-end task completion success rate. Although it is possible to complete each task in more than one way, we want to follow the how-to instruction text exactly, so we consider a task to be completed successfully only if the entire sequence of actions predicted by the model exactly matches the sequence of actions taken by the annotator.

5.1 How well does retrieval work across languages?

The multilingual sentence embedding model (Feng et al., 2020) is excellent at matching how-to queries in non-EN languages to how-to queries in English (Table 6). Examination of the failures with non-EN text queries revealed noise in the dataset where a small percentage of queries are repetitions with minor variations such as punctuation. When Google Cloud Speech API is used as an off-the-shelf automated speech recognizer (ASR) to convert speech input to text, there is a measurable drop in performance across all languages, but the reduction is large for Swahili. We also noticed that ASR failures were due to poor voice clarity, background noise, and more common with technical terms such as "cache".

5.2 How does parsing performance scale with dataset and model size?

There is a steep increase in parsing performance from 4-shot prompting to 10-shot prompting (Fig. 4). At 30 examples, the number of tokens in the input exceeds the maximum that the model can handle and performance deteriorates. Marking salient spans in the instruction text as an intermediate step for chain of thought prompting (Wei et al.,

Model configuration	Parsing accuracy
PaLM 540B 20-shot ICL	46%
GPT-3 175B 20-shot ICL	50.9%
PaLM 8B soft prompt tune	49.1%
PaLM 62B soft prompt tune	64.9%
PaLM 540B soft prompt tune	66.8%
UL2 20B full finetune	66.8%
T5 11B full finetune	66.8%
PaLM 8B full finetune	64.5%
PaLM 62B full finetune	67.5%
PaLM 540B full finetune	**70.1%**

Table 4: Parsing accuracy of pre-trained models on the `UGIF-DataSet` test set. In-context learning (ICL) is with 20 randomly selected training samples (single run). Fine-tuning and soft prompt-tuning with a 50-token soft prompt prefix (Section 5.2) is performed with all 158 training samples and hyper-paramter search over dropout values 0.0, 0.02, 0.05, 0.1, and 0.2 on 256 TPUs for about 24 hrs each. For fine-tuning, the best dropout was 0.1 with training for 10k steps, and for soft prompt-tuning, the best dropout was 0.0 with training for 17.5k steps.

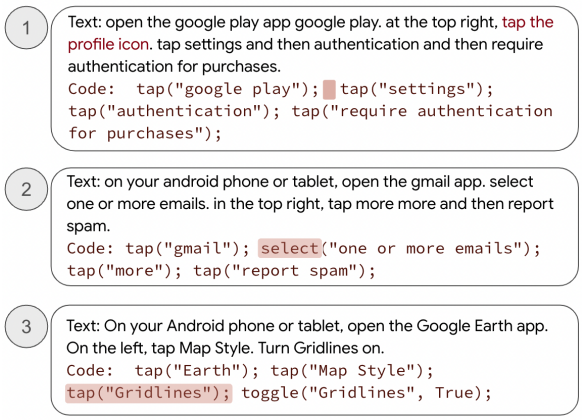

Figure 5: Incorrect sequences of macros generated by the 20-shot prompted PaLM 540B model. In the first example, the macro `tap("profile icon")` is omitted in the output. In the second example, the model hallucinates the non-existent `select()` macro. In the last example, it has generated an un-necessary tap: `tap("Gridlines")` (Section 5.3).

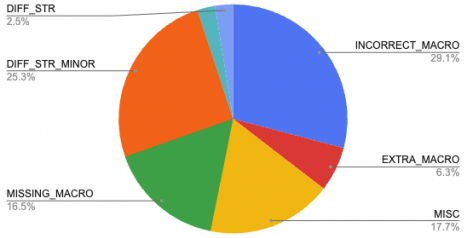

Figure 6: The types of parsing errors made by the PaLM 540B finetuned model on the 265 EN test samples in UGIF-DataSet (Section 5.3).

2022) degrades parsing performance. When all the available training samples are used with full fine-tuning or soft prompt tuning (Lester et al., 2021), the resulting performance is significantly better than few-shot prompting (Table 4). The parsing accuracy increases only modestly with model size when full fine-tuning is used. However, with soft prompt tuning, there is more benefit to using larger models.

5.3 What are the common failure modes of large language models for parsing?

We examined the test samples where the model's predictions were incorrect (Fig. 5) and found the PaLM 540B finetuned model (a) generated incorrect macros, (b) made minor errors in predicting the span of the argument such as including the full stop, (c) missed salient parts of the input instruction resulting in skipped macros, and (d) hallucinated non-existent macros (Fig 6).

5.3.1 How well do existing models work for UI grounding?

We find that even simple string matching models can offer good performance when the language in the how-to matches the UI language (Table. 5). To our surprise, UiBERT underperformed this baseline. When the instruction text and the UI language are different, we have to use LaBSE which is a multilingual model, but we find that performance with English is still better than other languages. An examination of the incorrectly predicted samples (Fig. 7) using LaBSE revealed these modes of failure (Fig. 8): (a) Inexact string matching fails and the model keeps scrolling in the hope of a better match which it never finds (84.5%), (b) the model overtriggers and chooses an inexact match instead of scrolling and looking for a better match (5.2%), (c) the model lacks knowledge of common UI patterns and app names, so it gets confused between "Play Store" and "Google One" when trying find the closest match for "Google Play" (5.2%).

The cases where the grounding model overtriggers and chooses a partially matching UI element and fails to either scroll down or recognize that the how-to is outdated results in incorrectly executed steps on the UI. These are of the most serious concern since they lead to a poor user experience.

Model configuration	UI Language							
	en	kn	mr	gu	hi	bn	es	sw
Oracle parse, Jaccard ground	55.4	—	—	—	—	—	—	—
Oracle parse, UiBERT ground	31.7	—	—	—	—	—	—	—
Oracle parse, LaBSE ground	52.8	36.6	39.2	41.5	43.7	40.7	49.8	35.4
PaLM 540B parse, LaBSE ground	48.6	33.6	36.6	38.5	40	37.7	46.4	32.1

Table 5: End-to-end task completion success rate of different model configurations on the UGIF-DataSet test set (Section 5.3.1).

Query Language	Oracle text P@1	ASR text P@1
en	100	94.4
kn	97.9	88.6
mr	98.1	91.7
gu	97.3	89.6
hi	94.6	91.3
bn	97.3	91.2
sw	93.0	76.4
es	96.5	94.8

Table 6: Comparison of performance for retrieving the closest matching how-to in English from queries in different languages (Section 5.1).

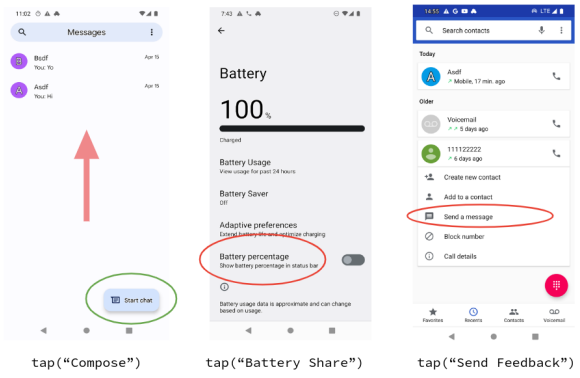

Figure 7: The UI grounding model chooses incorrect actions given the UI state and the macro. In the first example, the model should have tapped on "Start chat" as the matching element for "Compose" but instead tries scrolling down and throws an error that a matching UI element is not found. In the second example, the model should have scrolled down to find "Battery share" but instead erroneously selects the partially matching "Battery percentage". In the last example, the model should have recognized that the "Send feedback" button is missing in the UI and thrown an error, but instead erroneously selects the partially matching "Send a message" button (Section 5).

Moreover, help articles frequently become out-of-date as evidenced by the fact that 29% of the samples in UGIF-DataSet are marked by annotators as having instruction text not matching the UI in Android 12.

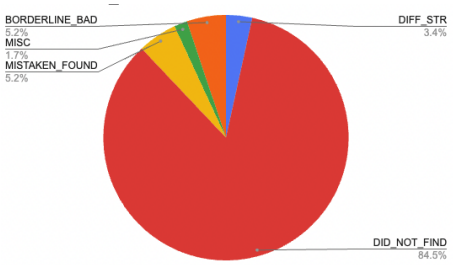

Figure 8: Categories of UI grounding errors using LaBSE on the 265 EN test samples in UGIF-DataSet (Section 5.3.1).

Model, Dataset	Success rate
Li et al. (2020a), PixelHelp (en)	70.5%
Ours, PixelHelp (en)	71.1%
Ours, UGIF-DataSet (en)	48.6%
Ours, UGIF-DataSet (sw)	32.1%

Table 7: Comparison of our best performing model (PaLM 540B for parsing and LaBSE for grounding) on different datasets. There is a wide gap between the model performance on the PixelHelp (en) dataset and UGIF-DataSet (sw) which suggests considerable headroom for improvement (Section 5).

We also evaluated our best performing model on the PixelHelp dataset (Li et al., 2020a). Table 7 shows that UGIF-DataSet is a harder dataset with significantly greater headroom for improvement especially in non-EN languages.

6 Conclusion

We proposed helping new smartphone users by showing them how to perform tasks on the UI based on voice queries. We evaluated existing language and sentence similarity models for the task of retrieving and executing how-to instructions on the UI where the UI language potentially differs from the language used in the instruction text. The

models we build for this task must be capable of adapting to minor variations in the UI as the newer versions of the app are frequently released and instructions become outdated. Multilingual UIs pose the challenge of having to simultaneously work with multiple languages in a single UI screen since app developers may not have provided translations for all UI elements. Finally, our evaluation of current pre-trained models suggests that there is significant room for improvement and that a multimodal language-UI foundation model could lead to substantial gains.

7 Limitations

UGIF-DataSet contains UI tasks on only a few popular Android smartphone apps. It does not include tasks in other form factors such as tablets or watches. Although a model trained exclusively on this dataset may not be sufficient for UI tasks on other OSes and form factors, by describing the challenges we faced, we hope this work contributes towards building a well-lit path to collect similar datasets and build models for other OSes and form factors.

The user interface evolves much more frequently than natural language or images. As a result, the dataset and models trained on it may need adaptations, which we leave as a topic of future exploration. Since we have captured UI screens at a particular point in time, we were unable to quantify the reduction in task completion rate due to UI drift or investigate methods specifically aimed at addressing such UI changes. An important future direction is to capture such UI changes as apps evolve over time and investigate how well models generalize to these changes.

Our dataset contains only one speech sample per query in each language, so the diversity of speech samples is limited. Moreover, the speech samples were crowd-sourced by asking annotators to speak out loud the how-to query in the title of the FAQ page, so this may not match how users might ask queries with the same intent without being prompted with the FAQ page. All the instructions have been scraped from the Google support site, so our evaluation of parsing does not cover instruction text on forums and other support sites. Furthermore, all the UI captures in our dataset start at the home screen, but it would be desirable to also evaluate UI grounding from arbitrary starting points.

8 Ethical Considerations

Automated agents that operate over the UI could potentially be misused and pollute the global digital commons by making it harder for app developers to trust that the user is a real user. As a result, it is possible that many developers may choose to mitigate this by requiring some form of identification to use the app, which could hurt marginalized communities and users who struggle with such entry barriers. Further investigations and user studies on the benefits of automated UI agents will be helpful.

Acknowledgments

We would like to thank Alok Talekar, Yang Zhou, and Tai Vu for reviewing the code for UGIF, Robert Berry and Anton Vayvod for helping us understand the intricacies of the Android accessibility API and debug issues we faced with it, and Lukas Zilka for providing us with the scraped Android support pages. Many thanks to Emily Pitler, Shikhar Vashishth, and Fernando Pereira for reviewing drafts of this paper and offering their helpful suggestions. We are grateful to Jason Pensy for guiding us in setting up the annotation tasks. We also thank Dinesh Tewari for helping us with the dataset release.

References

Jean-Baptiste Alayrac, Jeff Donahue, Pauline Luc, Antoine Miech, Iain Barr, Yana Hasson, Karel Lenc, Arthur Mensch, Katherine Millican, Malcolm Reynolds, et al. 2022. Flamingo: a visual language model for few-shot learning. *Advances in Neural Information Processing Systems*, 35:23716–23736.

Android. 2022. Write automated tests with ui automator. *Android Documentation*.

Gilles Baechler and Srinivas Sunkara. 2021. Improving mobile app accessibility with icon detection. *Google AI Blog*.

Chongyang Bai, Xiaoxue Zang, Ying Xu, Srinivas Sunkara, Abhinav Rastogi, Jindong Chen, et al. 2021. Uibert: Learning generic multimodal representations for ui understanding. *arXiv preprint arXiv:2107.13731*.

Satchuthananthavale RK Branavan, Harr Chen, Luke Zettlemoyer, and Regina Barzilay. 2009. Reinforcement learning for mapping instructions to actions. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, pages 82–90.

SRK Branavan, Luke Zettlemoyer, and Regina Barzilay. 2010. Reading between the lines: Learning to map high-level instructions to commands. In *Proceedings of the 48th annual meeting of the association for computational linguistics*, pages 1268–1277.

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

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. 2022. Palm: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*.

Fangxiaoyu Feng, Yinfei Yang, Daniel Cer, Naveen Arivazhagan, and Wei Wang. 2020. Language-agnostic bert sentence embedding. *arXiv preprint arXiv:2007.01852*.

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.

Brian Lester, Rami Al-Rfou, and Noah Constant. 2021. The power of scale for parameter-efficient prompt tuning. *arXiv preprint arXiv:2104.08691*.

Gang Li and Yang Li. 2022. Spotlight: Mobile ui understanding using vision-language models with a focus. *arXiv preprint arXiv:2209.14927*.

Wei Li. 2021. Learning ui navigation through demonstrations composed of macro actions. *arXiv preprint arXiv:2110.08653*.

Yang Li, Jiacong He, Xin Zhou, Yuan Zhang, and Jason Baldridge. 2020a. Mapping natural language instructions to mobile ui action sequences. *arXiv preprint arXiv:2005.03776*.

Yang Li, Gang Li, Luheng He, Jingjie Zheng, Hong Li, and Zhiwei Guan. 2020b. Widget captioning: Generating natural language description for mobile user interface elements. *arXiv preprint arXiv:2010.04295*.

Evan Zheran Liu, Kelvin Guu, Panupong Pasupat, Tianlin Shi, and Percy Liang. 2018. Reinforcement learning on web interfaces using workflow-guided exploration. *arXiv preprint arXiv:1802.08802*.

Corey Lynch and Pierre Sermanet. 2020. Language conditioned imitation learning over unstructured data. *arXiv preprint arXiv:2005.07648*.

Ramesh Manuvinakurike, Jacqueline Brixey, Trung Bui, Walter Chang, Doo Soon Kim, Ron Artstein, and Kallirroi Georgila. 2018. Edit me: A corpus and a framework for understanding natural language image editing. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, Peter J Liu, et al. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, 21(140):1–67.

Peeyush Ranjan. 2022. An anthology of insights, for a more inclusive internet. *Google Blog*.

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

Yi Tay, Mostafa Dehghani, Vinh Q Tran, Xavier Garcia, Dara Bahri, Tal Schuster, Huaixiu Steven Zheng, Neil Houlsby, and Donald Metzler. 2022. Unifying language learning paradigms. *arXiv preprint arXiv:2205.05131*.

Daniel Toyama, Philippe Hamel, Anita Gergely, Gheorghe Comanici, Amelia Glaese, Zafarali Ahmed, Tyler Jackson, Shibl Mourad, and Doina Precup. 2021. Androidenv: A reinforcement learning platform for android. *arXiv preprint arXiv:2105.13231*.

Sagar Gubbi Venkatesh, Anirban Biswas, Raviteja Upadrashta, Vikram Srinivasan, Partha Talukdar, and Bharadwaj Amrutur. 2021. Spatial reasoning from natural language instructions for robot manipulation. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 11196–11202. IEEE.

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

Nancy Xu, Sam Masling, Michael Du, Giovanni Campagna, Larry Heck, James Landay, and Monica S Lam. 2021. Grounding open-domain instructions to automate web support tasks. *arXiv preprint arXiv:2103.16057*.

Mingyuan Zhong, Gang Li, Peggy Chi, and Yang Li. 2021. Helpviz: Automatic generation of contextual visual mobile tutorials from text-based instructions. In *The 34th Annual ACM Symposium on User Interface Software and Technology*, pages 1144–1153.