A Dataset for Interactive Vision-Language Navigation with Unknown Command Feasibility

Andrea Burns[1], Deniz Arsan[2], Sanjna Agrawal[1], Ranjitha Kumar[2], Kate Saenko[1,3], and Bryan A. Plummer[1]

[1] Boston University, Boston MA 02215, USA
{aburns4,sanjna,saenko,bplum}@bu.edu
[2] University of Illinois Urbana-Champaign, Champaign IL 61820, USA
{darsan2,ranjitha}@illinois.edu
[3] MIT-IBM Watson AI Lab, Cambridge MA 02142, USA

Abstract. Vision-language navigation (VLN), in which an agent follows language instruction in a visual environment, has been studied under the premise that the input command is fully feasible in the environment. Yet in practice, a request may not be possible due to language ambiguity or environment changes. To study VLN with unknown command feasibility, we introduce a new dataset Mobile app Tasks with Iterative Feedback (MoTIF), where the goal is to complete a natural language command in a mobile app. Mobile apps provide a scalable domain to study real downstream uses of VLN methods. Moreover, mobile app commands provide instruction for interactive navigation, as they result in action sequences with state changes via clicking, typing, or swiping. MoTIF is the first to include feasibility annotations, containing both binary feasibility labels and fine-grained labels for why tasks are unsatisfiable. We further collect follow-up questions for ambiguous queries to enable research on task uncertainty resolution. Equipped with our dataset, we propose the new problem of feasibility prediction, in which a natural language instruction and multimodal app environment are used to predict command feasibility. MoTIF provides a more realistic app dataset as it contains many diverse environments, high-level goals, and longer action sequences than prior work. We evaluate interactive VLN methods using MoTIF, quantify the generalization ability of current approaches to new app environments, and measure the effect of task feasibility on navigation performance.

Keywords: Vision-language navigation, task feasibility, mobile apps

1 Introduction

Vision-language navigation (VLN) has made notable progress toward natural language instruction following [5,17,31,32,38,39,44]. While navigation datasets exist for home environments [3,8,19,38] and digital environments like mobile apps and websites [25,26,33,37], none capture the possibility that the language request may not be feasible in the given environment. When high-level natural language goals are requested, they may not be feasible for various reasons: the

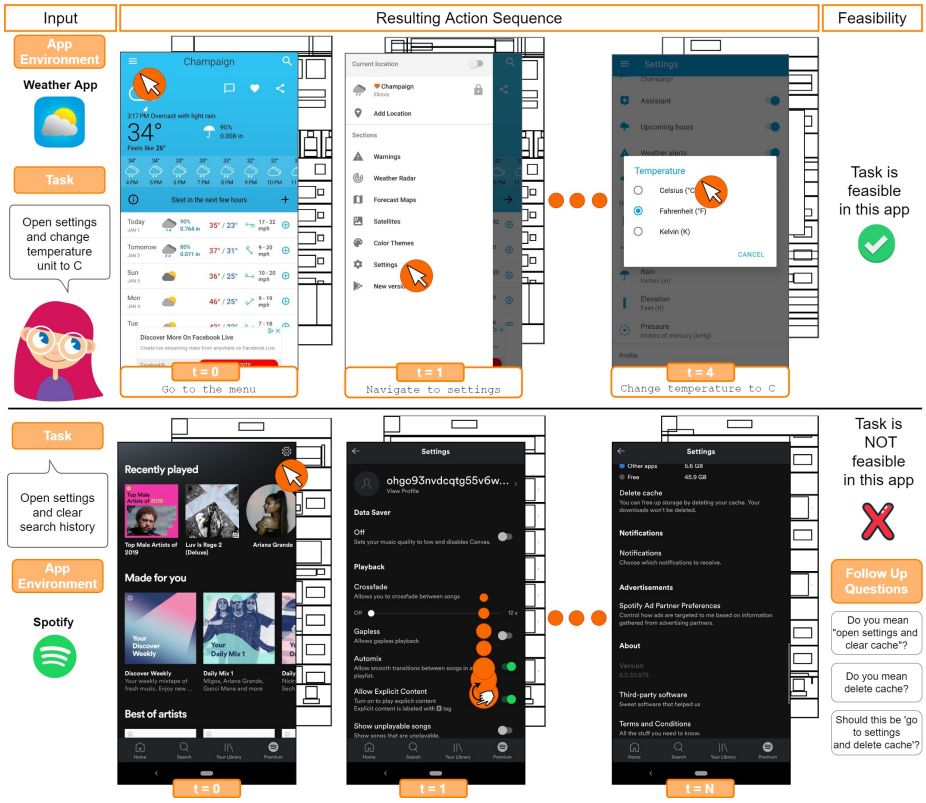

Fig. 1. MoTIF natural language commands which may not be possible. At each time step, action coordinates (*i.e.*, where clicking, typing, or scrolling occurs), the app screen, and view hierarchy (*i.e.*, the app backend, illustrated behind it) are captured

request may be ambiguous or state dependent, refer to functionality that is no longer available, or is reasonable in a similar environment but not satisfiable in the current. Task feasibility has been studied to determine question relevance for text-only [12] and visual question answering [14,30,36], but it has not been explored in interactive multimodal environments.

To study interactive task feasibility, we propose Mobile app Tasks with Iterative Feedback (MoTIF)[4], the largest dataset designed to support interactive methods for completing natural language tasks in mobile apps. As illustrated in Figure 1, a sample includes the natural language command (*i.e.*, task), app view hierarchy, app screen image, and action coordinates for each time step. MoTIF contains both feasible and infeasible requests, unlike any VLN dataset to date. In addition to these binary feasibility labels for each task, we collect subclass an-

[4] `https://github.com/aburns4/MoTIF`

notations for why tasks are infeasible and natural language follow-up questions. Our dataset provides a domain with practical downstream applications to study vision-language navigation, as well as data for investigating app design [9,10,27], human-computer interfaces [21,22,23], and document understanding [4,20,43].

We propose a baseline model for task feasibility prediction and confirm app exploration is necessary, with visual inputs key to accuracy. Surprisingly, prior representation learning approaches specific to the mobile app domain (*e.g.*, app icon features) do not result in the best performance. We then evaluate methods for automating MoTIF's commands and find MoTIF's diverse test set are challenging for prior work. Performance trends between seen and unseen app environments point to the need for more in-app exploration during training and qualitative failures in the best baseline model demonstrate the importance of visual understanding for MoTIF.

We summarize our contributions below:

- A new vision-language navigation dataset, Mobile app Tasks with Iterative Feedback (MoTIF). MoTIF has free form natural language commands for interactive goals in mobile apps, a subset of which are infeasible. It contains natural language tasks for the most app environments to date. MoTIF also captures multiple interactions including clicking, swiping and typing actions.
- A new vision-language task: interactive task feasibility classification, along with subclass annotations on why tasks are infeasible and follow-up questions for research toward resolving task uncertainty via dialogue.
- Benchmarks for feasibility classification and task automation with MoTIF. A thorough feature exploration is performed to evaluate the role of vision and language in task feasibility. We compare several methods on mobile app task automation, analyze generalization, and examine the effects of feasibility.

2 Related Work

We now discuss the key differences between MoTIF and existing datasets; we provide a side-by-side comparison in Table 1.

Task Feasibility Vision-language research has recently begun to study task feasibility. Gurari *et al.* introduced VizWiz [14], a visual question answering dataset for images taken by people that are blind, resulting in questions which may not be answerable. To the best of our knowledge, VizWiz is the only vision-language dataset with annotations for task feasibility, but it only addresses question answering over static images. Additionally, images that cannot be used to answer visual questions are easily classified, as they often contain blurred or random scenes (*e.g.*, the floor). Gardner *et al.* [12] explored question-answer plausibility prediction, but the questions used were generated from a bot, which could result in extraneous questions also easy to classify as implausible. Both are significantly different from the nuanced tasks of MoTIF with human generated queries, for which exploration is necessary to determine feasibility. MoTIF's infeasible tasks are always relevant to the Android app category, making it more challenging to discern feasibility compared to the distinct visual failures present in VizWiz.

Table 1. Comparison of MoTIF to existing datasets. We consider the number of natural language commands, command granularity, existence of feasibility annotations, the number of environments and whether the visual state is included in annotations

Dataset	Language Annotations			Dataset Environment	
	# Human Annotations	Task Granularity	Feasibility	# Environments	Visual State
(a) House					
R2R [3]	21,567	Low	✗	90	✓
IQA [13]	✗	High	✗	30	✓
ALFRED [38]	25,743	High & Low	✗	120	✓
(b) Webpage					
MiniWoB [37]	✗	High	✗	100	✗
PhraseNode [33]	50,000	Low	✗	1,800	✗
(c) Mobile App					
RicoSCA [25]	✗	Low	✗	9,700	✗
PIXELHELP [25]	187	Low	✗	4	✗
MoTIF (Ours)	6,100	High & Low	✓	125	✓

Vision-Language Navigation There are datasets that strictly navigate to locations like Room-to-Room [3] and Room-Across-Room [19], as well as interactive datasets where agents perform actions in the environment to complete a goal like ALFRED [38]. MoTIF is most similar to interactive VLN, as the natural language instructions are intended to complete a goal for the user, which requires clicking, typing, or swiping actions in the environment. However, an advantage of MoTIF is that it is a real, non-simulated domain to study interactive navigation, unlike all VLN prior work which uses simulated data [13,34,38,45].

Digital Task Automation Prior work has not studied web task automation in a multimodal setting, ignoring the rendered website image [33,37]. The existing datasets MiniWoB [37] and PhraseNode [33] also lack realism, as MiniWoB consists of handcrafted HTML and PhraseNode only captures single action commands on the home screen of websites. Unlike these datasets which limit interaction to a single screen, MoTIF contains action sequences with many different states (as shown in Figure 1), with a median of eight visited screens.

RicoSCA and PIXELHELP were introduced for mobile app task automation by Li *et al.* [25]. RicoSCA makes use of the mobile app dataset Rico [9], which captures random exploration in Android apps. Li *et al.* synthetically generate random commands with templates like "*click on* **x**" and stitch multiple together to any prescribed length. These generated step-by-step instructions do not reflect downstream use, where users ask for a high-level goal. For MoTIF, we instead collect free form high-level goals, and then post-process our data to automatically generate the low level subgoal instructions. PIXELHELP is a small mobile app dataset, but most commands are device specific. *I.e.*, the tasks refer to the phone itself, such as "*in the top control menu click the battery saver*," and are not in-app tasks like those in Figure 1. PIXELHELP also only contains clicking, while MoTIF has clicking, typing and swiping actions.

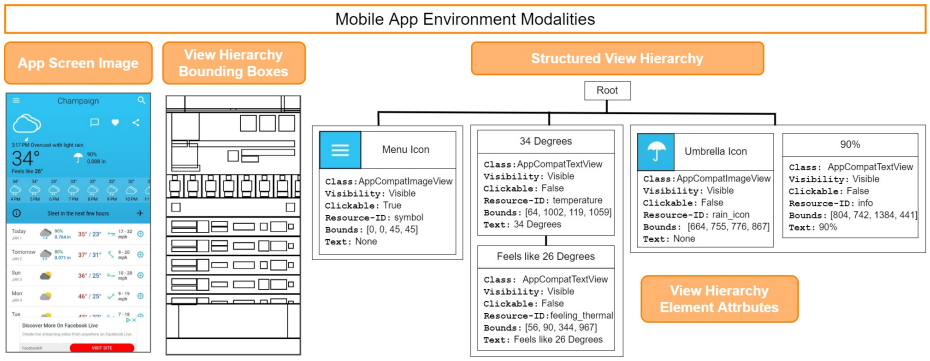

Fig. 2. We illustrate captured app modalities: the rendered screen and view hierarchy, which contains element metadata such as the Android class, resource ID, and text

3 MoTIF Dataset

For a mobile app task dataset, we need natural language tasks for apps and their resulting action sequence. Figure 1 illustrates MoTIF tasks like "*open settings and change temperature unit to C.*" For each command, we collect expert demonstrations of attempts to complete the request. At each time step we capture the app screen, the app backend view hierarchy, what type of action is taken, and where the action occurred. We show the modalities captured at each time step in greater detail in Figure 2. The Android app backend, *i.e.*, view hierarchy, is a tree-like structure akin to the Document Object Model (DOM) used for HTML. It organizes each screen element hierarchically, and contains additional metadata like the Android class of an element (*e.g.*, a text view or image view), its resource identifier, the text it contains, whether it is clickable, and other attributes.

3.1 Data Collection

We provide a general framework for others to collect natural language data with unknown feasibility; Figure 3 illustrates the collection pipeline. We select 125 apps for MoTIF over 15 app categories (the complete app list can be found in the Supplementary). Ten apps with (1) at least 50k downloads and (2) a rating higher than or equal to 4/5 were chosen for each category. Next, a first set of annotators writes commands. A list of (app, task) pairs are then provided to a second set of annotators in an interactive session, where they attempt the task, specify if it is not feasible, and can ask a clarifying question if not. The Supplementary includes annotator demographics, payment, and collection interface details.

Natural Language Commands To collect natural language tasks, we instruct workers to write commands as if they are asking the app to perform the task for them. Annotators can explore the app before deciding on their list of tasks. We ask them to write functional or navigational tasks, and not commands requiring

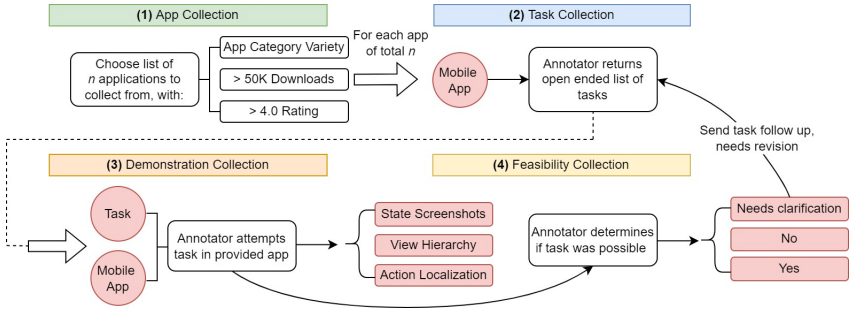

Fig. 3. The data collection pipeline (see Section 3.1). Colored boxes (app, task, demonstration, and feasibility collection) are stages of curating the dataset

text comprehension like summarizing an article. We neither structure the written tasks nor prescribe a specific number of tasks to be written for each app.

Task-Application Pairing When collecting natural language tasks, annotators can first explore the app. Once we have tasks for every app, we introduce additional feasibility uncertainty for the demonstration stage by collecting demos for both the original (app, task) list, as well as tasks paired with apps they were not originally written for. We create these additional (app, task) pairs by clustering tasks within each Android category (for example, clustering all tasks for Music and Audio Android apps) and selecting representatives from each cluster. These representative tasks are then collected for all apps of that category, which we coin "*category-clustered*." Specifically, we cluster the mean FastText embedding [7] of the language commands using K-Means [28].

Clusters are visualized with T-SNE [29] (see Supplementary). If a particular app's tasks are isolated from other clusters, we retain "*app-specific*" pairings, *i.e.*, the (app, task) pairs for tasks specifically written for the given app. This resulted in 40 apps having only app-specific tasks. If two apps' tasks are closely clustered, we group them; 17 apps' tasks were gathered this way. Figure 1 (bottom) shows a category-clustered task which was deemed infeasible by annotators. The command "*open settings and clear search history*" was paired with the music app Spotify even though it was not written for it. This is a sensible request given that Spotify is a music streaming app. Yet, no search history is found under settings, only the option to "delete cache," and follow-up questions are asked.

Task Demonstration and Feasibility Annotations Once the language commands are paired with apps, we instruct new annotators to demonstrate the task in the given app. We provide a website interface connected to physical Android phones for crowd workers to interact with, as well as anonymized login credentials so that no personally identifiable information is collected. They are instructed to record their demonstration after they have logged in (we consider logging in to be a separate task). After attempting to complete the task, they are brought to a post-survey where they provide details on whether or not the task was successfully completed. We therefore have demonstrations of actions taken both

Table 2. Task feasibility and follow-up question breakdown. Annotators can state the action: can't be completed (impossible), is under-specified (unclear), may be possible, but are unsure how or other tasks need to be completed first (premature)

#	Feasible	Infeasible			Total
		Impossible	Unclear	Premature	
Task Demonstrations	3,337	911	159	300	4,707
Follow-Up Questions	93	253	136	164	646

in successful and unsuccessful episodes, which may provide interesting insight toward how to reason about whether a task is or is not feasible, and why.

3.2 Dataset Analysis

Natural Language Commands We collected over 6.1k natural language tasks across 125 Android apps. The vocabulary size was 3,763 after removing non-alphanumeric characters. The average number of tasks submitted per app is 56, with average length being 5.6 words. The minimum task length is one, consisting of single action tasks like 'refresh' or 'login,' with the longest at 44 words. Word cloud visualizations, additional examples and statistics are in the Supplementary.

Feasibility Annotations We collect at least five expert demonstrations per (app, task) pair for two purposes: to reach a majority feasibility label and to capture different attempts of the same task, as some tasks can be completed in multiple ways. See the Supplementary for an annotator agreement histogram.

Of the resulting tasks, 29.2% are deemed infeasible by at least five crowd workers. However, the tasks considered infeasible do not always correlate to mismatched (app, task) pairs, *i.e.*, some *app-specific* tasks are deemed infeasible during demonstration. This confirms the need to study commands with unknown feasibility, as someone familiar with an app can still pose requests that are either not possible, ambiguous, or state dependent. Of the infeasible tasks, 16.8% are from app-specific pairs. *E.g.*, the request "*click shuttle and station*" originally written for the NASA app was labeled infeasible because the app has changing interactive features. Thus app changes and dynamic features also motivate studying infeasible requests, as a task that was once feasible may not always be.

Table 2 provides statistics on the number of task demonstrations and follow-up questions per feasibility category. There are three options for annotators to choose from: (1) the action cannot be completed in the app, (2) the action is unclear or under-specified, or (3) the task seems to be possible, but they cannot figure out how or other tasks need to be completed first. These map to Table 2's impossible, unclear, and premature columns. If a crowd worker cannot complete the task, they are prompted to ask a follow-up question. We instruct them to write the question(s) such that if they had the answer, they may now be able to complete the original action or perform an alternative task for the user.

4 Task Feasibility Experiments

We first perform experiments with MoTIF for task feasibility. Given a natural language command and the app states visited during its demonstration, the purpose of task feasibility prediction is to classify if the command can be completed. To determine feasibility, we expect a model to learn the most relevant state for the requested task and if the functionality needed to complete it is present. Our results provide an initial upper bound on performance, as the input action sequences can be considered the ground truth exploration needed to determine feasibility, as opposed to a learned agent's exploration. MoTIF has 4.7k demonstrations and we reserve 10% for testing. Note that our test set only includes (app, task) pairs for which all annotators agreed on their feasibility annotation.

4.1 Models

We propose a Multi-Layer Perceptron (MLP) baseline with two hidden layers that outputs a binary feasibility prediction. Each MLP is trained for 50 epochs with cross entropy using Stochastic Gradient Descent with a learning rate of 1e-2. The natural language command is always input to the classifier, and we ablate which app environment features are additional input. In addition to the feature ablations, we ablate how the demonstration sequence is aggregated (averaging or concatenating over time steps or using the last hidden state of an LSTM [16]).

Features We encode the task command and view hierarchy elements per step with mean pooled features. Specifically, we try both FastText [6] and CLIP [35] (trained with a Transformer backbone for its image and text encoders [11,41]). As seen in Figure 2, the view hierarchy captures all rendered app elements and their attributes: the element's text (ET), resource-identifier (ID) and class labels (CLS) which provide content and type information. We use the best combination of these attributes in Table 3 and have more ablations in the Supplementary. We also include Screen2Vec [24] in our view hierarchy representations. Screen2Vec is a semantic embedding of the view hierarchy, representing the view hierarchy with a GUI, text, and layout embedder. The GUI and text encoders make use of BERT features while the layout features are learned with an autoencoder. Thus, it tries to encode both textual and structural features, but no visual information.

For visual features, we extract ResNet152 [15] features for ten crops of each app image and CLIP features of each whole app image. We also include icon features by cropping all icon images per screen (*e.g.*, the menu and umbrella icons shown in Figure 2). We embed each icon image using the embedding layer of a CNN trained for the downstream task of icon classification by Liu *et al.* [27].

Metrics We report the average F1 score over ten runs with different random initialization. "Infeasible" is defined as the positive class, as we care more about correctly classifying tasks that are infeasible, than misclassifying feasible tasks.

Table 3. Task feasibility F1 score using our MLP. We ablate input features and action sequence aggregation. The random baseline predicts a feasibility label given the train distribution. On the right is a confusion matrix for the predictions of our best classifier

$\mathbf{C}_{feas}$ Input Features	Demo Aggregation		
	Avg	Cat	LSTM
Random	20.1		
(a) View Hierarchy			
FastText [6] (ET, ID)	16.7	43.6	34.1
CLIP [35] (ET, ID)	28.0	50.9	36.2
Screen2Vec [24]	25.9	33.7	36.0
(b) App Screen Image			
ResNet [15]	31.3	41.9	35.9
Icons [27]	0.4	40.0	15.2
CLIP [35]	44.7	58.2	<u>42.8</u>
(c) Best Combination			
CLIP [35] (Screen, ET, ID)	<u>44.8</u>	<u>61.1</u>	40.9

(c) Best Combination

		Ground Truth	
		Feasible	Infeasible
Prediction	Feasible	76.4%	8.6%
	Infeasible	4.0%	11.0%

4.2 Results

Our best task feasibility classifier (Table 3(c) left) achieves an F1 score of 61.1 when CLIP embeds the task, view hierarchy, and app screen image. This is still fairly low, and feature ablations demonstrate room to improve both the language and visual representations. While CLIP has shown significant performance gains in other vision-language tasks, it is somewhat surprising that domain-specific embeddings (*e.g.*, Screen2Vec, Icons) are not as competitive. The combination of view hierarchy and app screen features does not largely outperform the app screen image CLIP results (and does worse with LSTM aggregation), suggesting a need for better vision-language encodings which can pull features together from different modalities such as the view hierarchy.

We include the confusion matrix on the right of Table 3 for our best model. In downstream use, the classifier would result in 5% of tasks being missed out on; *i.e.*, 5% of tasks were incorrectly classified as infeasible. This reduces the utility of assistive applications, where we'd like all possible commands to correctly be completed. However, the 44% of infeasible tasks that were incorrectly classified as feasible can have more negative consequences. In application, this means a vision-language model would attempt to complete an unsatisfiable request, resulting in unknown behavior. We need downstream models to behave in reliable ways, especially for users that cannot verify the task was reasonably completed.

Table 3(a) left compares methods of encoding the view hierarchy. Using CLIP for view hierarchy elements results in notably better performance than FastText, albeit less significant when input demos are aggregated with an LSTM. Our final view hierarchy embedding is Screen2Vec which performs worse than CLIP and on par with FastText, despite being trained on mobile app data. Screen2Vec may not capture enough low level view hierarchy information to predict feasibility, and methods trained on huge data, even if from another domain, are more powerful.

In Table 3(b) left we ablate over the visual representations of the app screen. While icon representations are trained on images from the same domain as MoTIF, they are significantly less effective than ResNet and CLIP. The F1 score nearly drops to zero when the average icon feature is used, illustrating that the average icon does not carry useful information for feasibility classification. Icon features may be too low-level or require improved aggregation methods.

Comparing demonstration aggregation methods (averaging, concatenating, or LSTM), there is a trend that concatenating time steps is the best method, suggesting a sequential representation of the action sequence is needed. However, when the best representations for the view hierarchy and app screen are combined in Table 3(c), averaging manages to outperform the LSTM performance.

In future work we hope to learn hierarchical representations in order to encode global information such as that of Screen2Vec as well as local information from icon embeddings. Taking advantage of the tree structure from the view hierarchy via Transformers or Graph Neural Networks may help learn structured app features. Additionally, all current approaches do not take into account any notion of app "affordance," *i.e.*, which app elements are most actionable.

5 Task Automation Experiments

In app task automation, we are given an app environment (with all of its modalities) and a language command. The goal is to interact with the app and output a sequence of app actions that complete the task, akin to interactive VLN. At each time step there are two predictions: an action (clicking, typing, or swiping) and a localization (grounding visually on the app screen or classifying over the app elements). We benchmark several methods and analyze performance below.

5.1 Models

We adapt three models for the mobile app domain with as few changes as possible. The VLN approaches described below (Seq2Seq and MOCA) take both the high-level goal and low level instructions as input while Seq2Act only supports low level instruction. In the supplementary we include input language ablations to consider what performance with real downstream use would look like.

Seq2Seq is a VLN model for the ALFRED, a dataset of actionable commands for tasks in household environments. It originally predicts an action and binary mask at each time step. The mask isolates the household object on which the action is performed. The features at each time step include the attended language instruction, the current step's visual features, the last predicted action, and the hidden state of a BiLSTM which takes the former as input. The previous step's BiLSTM hidden state attends to the language input. These features are passed to a fully connected layer with Softmax for action prediction and a deconvolutional network for mask prediction. We replace the mask prediction network for three fully connected layers that predict a point in the app screen and minimize the mean squared error. Action prediction is trained via cross entropy.

MOCA [39], also proposed for ALFRED, decouples the action and grounding predictions of each step in a VLN sequence. One model stream is for the action prediction policy, and another for interaction grounding. Both streams first use a BiLSTM language encoder, which take the high-level goal or low level instruction as input, respectively. The encoded tokens are attended to using ResNet visual features via dynamic attention filters. Then, two LSTM decoders are used: one for the action policy stream and another for the interaction grounding.

At test time MOCA makes use of an off-the-shelf object segmentation model to perform grounding given the predicted object class. To adapt the object class prediction to mobile apps, we instead perform app element type prediction (prediction is over twelve classes, including button, checkbox, edit text, image view, and more). As no such segmentation model exists for mobile apps yet, we also predict bounding box localization directly using the LSTM decoder output, but use the app element type prediction to narrow grounding options at evaluation.

Seq2Act [25] models mobile app task automation in two stages: action phrase extraction and action grounding. Both stages are modeled with Transformers. The first model predicts a span (*i.e.*, substring) of the original input command that corresponds to the action type, action location, and action input. It has an encoder-decoder architecture: the encoder embeds the instruction's text tokens and the decoder computes a query vector for the action type, location, and input phrases given the previously decoded spans. A text span is selected for each decoder query (action type, action location, action input) via cosine similarity.

The action grounding model takes each extracted phrase as input to predict an action type and location (which app element it is performed on). Actions are predicted given the encoder embedding of the predicted action type span via a Multi-Layer Perception. To localize the action, a Transformer is trained to embed app elements using the view hierarchy attributes as shown in Figure 2. A Softmax is applied to the similarities of the predicted app location span embeddings and the latent app element representations. The max scoring app element becomes the grounding prediction for that time step.

Datasets We evaluate task automation on two MoTIF test splits: an app seen and an app unseen split to study generalization to new environments; generalization of tasks across apps is provided in the Supplementary. We jointly train VLN models on MoTIF and RicoSCA for additional data (see the Supplementary for additional experiments trained solely on MoTIF). Seq2Act was originally trained on RicoSCA and we adapt its training data split to be able to evaluate seen versus unseen apps at test time.

Features Visual features for Seq2Seq and MOCA are from the last convolutional layer of a ResNet18, as done for the original models; these features are needed for meaningful localization on the mobile app screen. We also include CLIP features of the screen at each time step. Note that VLN methods require a test-time environment; we build an offline version of each Android app to approximate a complete state-action space graph. Details on the creation of these graphs can be found in the Supplementary. Seq2Act does not use off-the-shelf features as input; all text and app element embeddings are learned from scratch.

Table 4. Mobile app task accuracy on MoTIF. We evaluate the Seq2Seq and MOCA navigation models and the Transformer grounding model Seq2Act

Model	App Seen			App Unseen		
	Action	Ground	Action + Ground	Action	Ground	Action + Ground
(a) Seq2Seq [38]						
Complete Sequence	68.5	22.5	22.5	54.3	18.0	17.7
Partial Sequence	89.5	40.4	40.1	81.7	31.3	30.6
(b) MOCA [39]						
Complete Sequence	51.1	21.3	20.7	44.8	17.0	15.1
Partial Sequence	78.5	40.0	38.6	72.2	32.7	30.0
(c) Seq2Act [25]						
Complete Sequence	98.8	27.6	27.6	94.9	23.5	23.5
Partial Sequence	99.7	64.4	64.3	98.9	62.2	61.7

Metrics We report complete and partial sequence accuracy per [25]: the complete score for an action sequence is 1 if the predicted and ground truth sequences have the same length and the same predictions at each step, else 0. The partial sequence score is the fraction of predicted steps that match the ground-truth. These are reported for action prediction (Action), action grounding (Ground), or both jointly. Seq2Seq localization is correct if the predicted point falls within the bounding box of the ground truth app element. MOCA localization is correct if the predicted bounding box and ground truth have an IoU greater than 0.5.

5.2 Results

Despite MOCA being a more recent model for interactive vision-language navigation, it generally does not outperform Seq2Seq. The app element type prediction MOCA uses may be responsible for the similar or lower accuracy, as the original intention of object class prediction was to narrow down grounding interaction to very few options. *E.g.*, in the home environments of ALFRED, which MOCA evaluated on, the object class predicted may be apple. If there is only a single apple in the scene, the object segmentation model would be highly effective for grounding. The mobile app domain differs in that there are many app elements per time step of the same type, *e.g.*, there are many app icons or app buttons, and this prediction may not significantly reduce the grounding prediction space.

The Seq2Seq and MOCA models perform worse than Seq2Act. While additional model ablations may improve performance, it is clear that action localization on the continuous app screen is more challenging. Seq2Act achieves the highest performance for all metrics. Seq2Act was originally evaluated with PIXELHELP [25] and achieved 70.6% complete Action + Ground accuracy on it, much higher than the accuracy reported on MoTIF. This may be due to PIXELHELP containing click-only tasks for four test environments, which does not reflect the model's performance on a greater variety of apps or tasks. MoTIF's step-by-

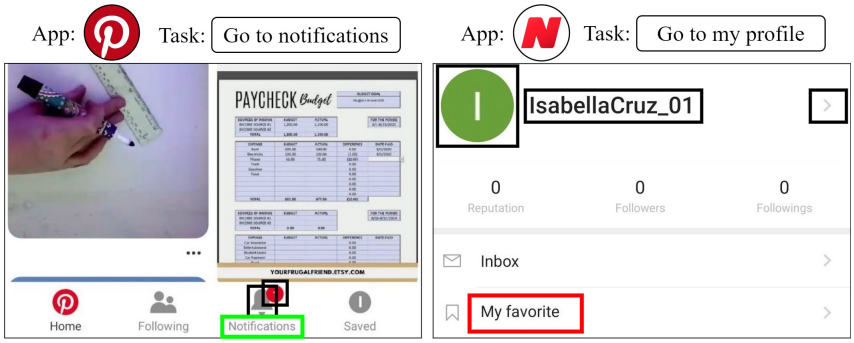

Fig. 4. Seq2Act text matching. Green and red boxes are valid and invalid predictions, respectively; black are additional valid ground truth. The left shows valid text matching, identifying "notifications" in the app Pinterest. The right shows Seq2Act incorrectly matching "my" in the input task to the app element "My favorite" in the Opera news app

step instructions also contain location descriptions for app elements which don't contain text, differing from the Seq2Act training data distribution.

Qualitatively inspecting misclassifications, we find one culprit to be Seq2Act overly relying on matching input task text to view hierarchy text. In Figure 4, we show Seq2Act's text matching tendency, which can result in failure. For example, Seq2Act predicts the app element with the word "my" in it for the input command "*go to my profile*." These results, in addition to the high visual performance from the feasibility classifier, verifies the need for visual input to correct model bias to match input task text directly to the view hierarchy.

Performance is unsurprisingly worse for unseen app environments. We suspect that current model formulations do not learn enough about app elements outside of the ground truth action sequences during training. None of the benchmark models include exploration, and as a result, may be biased to the small subset of elements seen in expert demonstration. In future work, using pretrained generic app features or incorporating exploration into the training process through reinforcement learning approaches may alleviate this.

6 Discussion

We find our best task feasibility prediction results to be low at a 61.1 F1 score, given that the input demonstrations serve as the oracle exploration needed to determine feasibility. In addition to improving vision-language feasibility reasoning, a necessary next step is to instead use learned explorations during training. Our ablations demonstrate that visual inputs are useful for feasibility prediction, and research toward better mobile app features that actually use the rendered screen could increase performance. Building hierarchical visual and textual features may provide stronger context clues for determining command feasibility

in the app environment. We also hope to perform experiments classifying why tasks are not feasible and automating question generation in response, making use of MoTIF's subclass annotations for infeasible tasks and follow up questions.

By evaluating action and grounding performance independently, we found that models for completing mobile app tasks can have more difficulty grounding and consistently perform more poorly in new app environments. Better representations of app elements are needed; specifically, incorporating pretraining tasks for improved app features or allowing for exploration outside of ground truth action sequences may be necessary to diversify test time predictions.

Limitations The MoTIF dataset is not on the scale of pretraining datasets used in other VL tasks (*e.g.*, Alt-Text [18], JFT-300M [40]), as it is very expensive and time costly to collect natural language commands and feasibility labels. MoTIF is nonetheless useful to the research community as it can be used to evaluate how existing methods would solve language-based digital tasks in realistic settings.

Societal Impact Methods for automating language commands and predicting command feasibility can be used to assist people who are not able to interact with apps, either situationally (*e.g.*, while driving) or physically (*e.g.*, users who are low-vision or blind). Improving mobile app task automation could better balance the capabilities of current assistive technologies, which typically lack agency or flexibility [42]. *E.g.*, screen readers are primarily used for web browsing and information consumption (lacking agency), while interactive virtual assistants (*e.g.*, Siri, Alexa) have limited, structured commands (lacking flexibility).

MoTIF's collection was designed to ensure no personally identifiable information is captured. But, in downstream use of app task automation, user privacy is of concern. People who use assistive tools (*e.g.*, people who are blind) already expose sensitive information to other humans to receive help [1,2]. To mitigate potential harm, deployment of our research can be limited to apps which do not require log in information; these are less likely to include name, address, or payment data. MoTIF does not have tasks which require payment, and we can deny payment related tasks to prevent fraud and other undesired outcomes.

7 Conclusion

We introduced Mobile app Tasks with Iterative Feedback (MoTIF), a new VLN dataset that contains natural language commands for tasks in mobile apps which may not be feasible. MoTIF is the first dataset to capture task uncertainty for interactive visual environments and contains greater linguistic and visual diversity than prior work, allowing for more research toward robust vision-language methods. We introduced the task of feasibility prediction and evaluate prior methods for automating mobile app tasks. Results verify that MoTIF poses new vision-language challenges, and that the vision-language community can make use of more realistic data to evaluate and improve upon current methods.

Acknowledgements This work is funded in part by Boston University, the Google Ph.D. Fellowship program, the MIT-IBM Watson AI Lab, the Google Faculty Research Award and NSF Grant IIS-1750563.

References

1. Ahmed, T., Hoyle, R., Connelly, K., Crandall, D., Kapadia, A.: Privacy concerns and behaviors of people with visual impairments. In: Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems. p. 3523–3532. CHI '15, Association for Computing Machinery, New York, NY, USA (2015). https://doi.org/10.1145/2702123.2702334, https://doi.org/10.1145/2702123.2702334
2. Akter, T., Dosono, B., Ahmed, T., Kapadia, A., Semaan, B.C.: "I am uncomfortable sharing what I can't see": Privacy concerns of the visually impaired with camera based assistive applications. In: USENIX Security Symposium (2020)
3. Anderson, P., Wu, Q., Teney, D., Bruce, J., Johnson, M., Sünderhauf, N., Reid, I.D., Gould, S., van den Hengel, A.: Vision-and-language navigation: Interpreting visually-grounded navigation instructions in real environments. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2018)
4. Appalaraju, S., Jasani, B., Kota, B.U., Xie, Y., Manmatha, R.: Docformer: End-to-end transformer for document understanding. In: 2021 IEEE/CVF International Conference on Computer Vision (ICCV) (2021)
5. Blukis, V., Paxton, C., Fox, D., Garg, A., Artzi, Y.: A persistent spatial semantic representation for high-level natural language instruction execution (2021)
6. Bojanowski, P., Grave, E., Joulin, A., Mikolov, T.: Enriching word vectors with subword information. Transactions of the Association for Computational Linguistics **5** (2017)
7. Conneau, A., Lample, G., Ranzato, M., Denoyer, L., Jégou, H.: Word translation without parallel data. In: International Conference on Learning Representations (ICLR) (2018)
8. Das, A., Datta, S., Gkioxari, G., Lee, S., Parikh, D., Batra, D.: Embodied Question Answering. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2018)
9. Deka, B., Huang, Z., Franzen, C., Hibschman, J., Afergan, D., Li, Y., Nichols, J., Kumar, R.: Rico: A mobile app dataset for building data-driven design applications. In: 30th Annual Symposium on User Interface Software and Technology (UIST) (2017)
10. Deka, B., Huang, Z., Kumar, R.: Erica: Interaction mining mobile apps. In: 29th Annual Symposium on User Interface Software and Technology (UIST) (2016)
11. Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., Houlsby, N.: An image is worth 16x16 words: Transformers for image recognition at scale (2021)
12. Gardner, R., Varma, M., Zhu, C., Krishna, R.: Determining question-answer plausibility in crowdsourced datasets using multi-task learning. In: W-NUT@EMNLP (2020)
13. Gordon, D., Kembhavi, A., Rastegari, M., Redmon, J., Fox, D., Farhadi, A.: IQA: Visual question answering in interactive environments. In: 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). pp. 4089–4098 (2018). https://doi.org/10.1109/CVPR.2018.00430
14. Gurari, D., Li, Q., Stangl, A.J., Guo, A., Lin, C., Grauman, K., Luo, J., Bigham, J.P.: Vizwiz grand challenge: Answering visual questions from blind people. In: Conference on Computer Vision and Pattern Recognition (CVPR) (2018)
15. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). pp. 770–778 (2016). https://doi.org/10.1109/CVPR.2016.90

16. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural Comput. **9**(8), 1735–1780 (Nov 1997). https://doi.org/10.1162/neco.1997.9.8.1735, https://doi.org/10.1162/neco.1997.9.8.1735
17. Irshad, M.Z., Ma, C.Y., Kira, Z.: Hierarchical cross-modal agent for robotics vision-and-language navigation. In: Proceedings of the IEEE International Conference on Robotics and Automation (ICRA) (2021), https://arxiv.org/abs/2104.10674
18. Jia, C., Yang, Y., Xia, Y., Chen, Y.T., Parekh, Z., Pham, H., Le, Q.V., Sung, Y., Li, Z., Duerig, T.: Scaling up visual and vision-language representation learning with noisy text supervision (2021)
19. Ku, A., Anderson, P., Patel, R., Ie, E., Baldridge, J.: Room-across-room: Multilingual vision-and-language navigation with dense spatiotemporal grounding. In: Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP). pp. 4392–4412. Association for Computational Linguistics, Online (Nov 2020). https://doi.org/10.18653/v1/2020.emnlp-main.356, https://aclanthology.org/2020.emnlp-main.356
20. Li, P., Gu, J., Kuen, J., Morariu, V.I., Zhao, H., Jain, R., Manjunatha, V., Liu, H.: Selfdoc: Self-supervised document representation learning. In: 2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) (2021)
21. Li, T.J.J., Azaria, A., Myers, B.A.: Sugilite: Creating multimodal smartphone automation by demonstration. In: Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems. p. 6038–6049. CHI '17, Association for Computing Machinery, New York, NY, USA (2017)
22. Li, T.J.J., Chen, J., Xia, H., Mitchell, T.M., Myers, B.A.: Multi-Modal Repairs of Conversational Breakdowns in Task-Oriented Dialogs, p. 1094–1107. Association for Computing Machinery, New York, NY, USA (2020)
23. Li, T.J.J., Mitchell, T.M., Myers, B.A.: Demonstration + Natural Language: Multimodal Interfaces for GUI-Based Interactive Task Learning Agents, pp. 495–537. Springer International Publishing, Cham (2021)
24. Li, T.J.J., Popowski, L., Mitchell, T.M., Myers, B.A.: Screen2vec: Semantic embedding of gui screens and gui components. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. CHI '21 (2021)
25. Li, Y., He, J., Zhou, X., Zhang, Y., Baldridge, J.: Mapping natural language instructions to mobile UI action sequences. In: Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics. pp. 8198–8210. Association for Computational Linguistics, Online (Jul 2020). https://doi.org/10.18653/v1/2020.acl-main.729, https://www.aclweb.org/anthology/2020.acl-main.729
26. Li, Y., Li, G., Zhou, X., Dehghani, M., Gritsenko, A.A.: VUT: versatile UI transformer for multi-modal multi-task user interface modeling. CoRR **abs/2112.05692** (2021), https://arxiv.org/abs/2112.05692
27. Liu, T.F., Craft, M., Situ, J., Yumer, E., Mech, R., Kumar, R.: Learning design semantics for mobile apps. In: 31st Annual Symposium on User Interface Software and Technology (UIST) (2018)
28. Lloyd, S.: Least squares quantization in pcm. In: IEEE Transactions on Information Theory (1982)
29. van der Maaten, L., Hinton, G.: Visualizing data using t-SNE. Journal of Machine Learning Research **9**, 2579–2605 (2008), http://www.jmlr.org/papers/v9/vandermaaten08a.html
30. Massiceti, D., Dokania, P.K., Siddharth, N., Torr, P.H.S.: Visual dialogue without vision or dialogue. CoRR **abs/1812.06417** (2018), http://arxiv.org/abs/1812.06417

31. Min, S.Y., Chaplot, D.S., Ravikumar, P., Bisk, Y., Salakhutdinov, R.: Film: Following instructions in language with modular methods (2021)
32. Nguyen, K., Daumé III, H.: Help, anna! visual navigation with natural multimodal assistance via retrospective curiosity-encouraging imitation learning. In: Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP) (November 2019)
33. Pasupat, P., Jiang, T.S., Liu, E.Z., Guu, K., Liang, P.: Mapping natural language commands to web elements. In: Empirical Methods in Natural Language Processing (EMNLP) (2018)
34. Puig, X., Ra, K., Boben, M., Li, J., Wang, T., Fidler, S., Torralba, A.: Virtualhome: Simulating household activities via programs. In: 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). pp. 8494–8502. IEEE Computer Society, Los Alamitos, CA, USA (jun 2018). https://doi.org/10.1109/CVPR.2018.00886, https://doi.ieeecomputersociety.org/10.1109/CVPR.2018.00886
35. Radford, A., Kim, J.W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., Sastry, G., Askell, A., Mishkin, P., Clark, J., Krueger, G., Sutskever, I.: Learning transferable visual models from natural language supervision. CoRR **abs/2103.00020** (2021), https://arxiv.org/abs/2103.00020
36. Ray, A., Christie, G., Bansal, M., Batra, D., Parikh, D.: Question relevance in vqa: Identifying non-visual and false-premise questions (2016)
37. Shi, T., Karpathy, A., Fan, L., Hernandez, J., Liang, P.: World of bits: An open-domain platform for web-based agents. In: 34th International Conference on Machine Learning (ICML) (2015)
38. Shridhar, M., Thomason, J., Gordon, D., Bisk, Y., Han, W., Mottaghi, R., Zettlemoyer, L., Fox, D.: ALFRED: A Benchmark for Interpreting Grounded Instructions for Everyday Tasks. In: The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2020), https://arxiv.org/abs/1912.01734
39. Singh, K.P., Bhambri, S., Kim, B., Mottaghi, R., Choi, J.: Factorizing perception and policy for interactive instruction following. In: Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV) (2021)
40. Sun, C., Shrivastava, A., Singh, S., Gupta, A.: Revisiting unreasonable effectiveness of data in deep learning era. CoRR **abs/1707.02968** (2017), http://arxiv.org/abs/1707.02968
41. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L., Polosukhin, I.: Attention is all you need. In: Conference on Neural Information Processing Systems (NeurIPS) (2017)
42. Vtyurina, A., Fourney, A., Morris, M.R., Findlater, L., White, R.W.: Bridging screen readers and voice assistants for enhanced eyes-free web search. In: International ACM SIGACCESS Conference on Computers and Accessibility (ASSETS) (2019)
43. Yamaguchi, K.: Canvasvae: Learning to generate vector graphic documents. In: Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV) (2021)
44. Zhu, F., Zhu, Y., Chang, X., Liang, X.: Vision-language navigation with self-supervised auxiliary reasoning tasks. In: 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). pp. 10009–10019 (2020). https://doi.org/10.1109/CVPR42600.2020.01003
45. Zhu, Y., Gordon, D., Kolve, E., Fox, D., Fei-Fei, L., Gupta, A.K., Mottaghi, R., Farhadi, A.: Visual semantic planning using deep successor representations. In: 2017 IEEE International Conference on Computer Vision (ICCV) (2017)