LlamaTouch: A Faithful and Scalable Testbed for Mobile UI Task Automation

Li Zhang
State Key Laboratory of Networking and Switching Technology,
Beijing University of Posts and Telecommunications

Shihe Wang
State Key Laboratory of Networking and Switching Technology,
Beijing University of Posts and Telecommunications

Xianqing Jia
State Key Laboratory of Networking and Switching Technology,
Beijing University of Posts and Telecommunications

Zhihan Zheng
State Key Laboratory of Networking and Switching Technology,
Beijing University of Posts and Telecommunications

Yunhe Yan
State Key Laboratory of Networking and Switching Technology,
Beijing University of Posts and Telecommunications

Longxi Gao
State Key Laboratory of Networking and Switching Technology,
Beijing University of Posts and Telecommunications

Yuanchun Li
Institute for AI Industry Research (AIR), Tsinghua University

Mengwei Xu
State Key Laboratory of Networking and Switching Technology,
Beijing University of Posts and Telecommunications

ABSTRACT

The emergent large language/multimodal models facilitate the evolution of mobile agents, especially in mobile UI task automation. However, existing evaluation approaches, which rely on human validation or established datasets to compare agent-predicted actions with predefined action sequences, are unscalable and unfaithful. To overcome these limitations, this paper presents LlamaTouch, a testbed for on-device mobile UI task execution and faithful, scalable task evaluation. By observing that the task execution process only transfers UI states, LlamaTouch employs a novel evaluation approach that only assesses whether an agent traverses all manually annotated, essential application/system states. LlamaTouch comprises three key techniques: (1) *On-device task execution* that enables mobile agents to interact with realistic mobile environments for task execution. (2) *Fine-grained UI component annotation* that merges pixel-level screenshots and textual screen hierarchies to explicitly identify and precisely annotate essential UI components with a rich set of designed annotation primitives. (3) *A multi-level application state matching algorithm* that utilizes exact and fuzzy matching to accurately detect critical information in each screen, even with unpredictable UI layout/content dynamics. LlamaTouch currently incorporates four mobile agents and 496 tasks, encompassing both tasks in the widely-used datasets and our self-constructed ones to cover more diverse mobile applications. Evaluation results demonstrate LlamaTouch's high faithfulness of evaluation in real-world mobile environments and its better scalability than human validation. LlamaTouch also enables easy task annotation and integration of new mobile agents. Code and dataset are publicly available at https://github.com/LlamaTouch/LlamaTouch.

CCS CONCEPTS

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

KEYWORDS

mobile agent, UI task automation, evaluation, testbed

ACM Reference Format:
Li Zhang, Shihe Wang, Xianqing Jia, Zhihan Zheng, Yunhe Yan, Longxi Gao, Yuanchun Li, and Mengwei Xu. 2024. LlamaTouch: A Faithful and Scalable Testbed for Mobile UI Task Automation. In *The 37th Annual ACM Symposium on User Interface Software and Technology (UIST '24), October 13–16, 2024, Pittsburgh, PA, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3654777.3676382

1 INTRODUCTION

Mobile intelligent agents empower users to interact with their smartphones using natural languages, alleviating them from tedious and cumbersome smartphone operations. These agents are particularly beneficial for individuals with visual or hand impairments, or in situations where using a screen is not practical (e.g., driving). Notable mobile agents, such as Apple Siri [6] and Google Assistant [14], have become indispensable services on smartphones. The recent advent of large language models (LLMs) and multimodal LLMs has facilitated researchers in building more powerful mobile agents [16, 19, 29, 30]. The key capability of these agents is to comprehend user instructions in natural language and execute corresponding actions on the mobile interface, as called *mobile UI task automation*, e.g., "forward the last email from Bob to Alice".

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

UIST '24, October 13–16, 2024, Pittsburgh, PA, USA

© 2024 Copyright held by the owner/author(s). Publication rights licensed to ACM.

ACM ISBN 979-8-4007-0628-8/24/10
https://doi.org/10.1145/3654777.3676382

Despite claims of powerful task automation capabilities achieved by recent LLM-powered mobile agents, their evaluation methods are somewhat flawed. Unlike traditional machine learning models evaluated on well-established static datasets, mobile agents need to interact with the dynamic and indeterministic states of a smartphone (e.g., network connectivity, dynamic content) as inputs. Additionally, mobile devices use touch and gesture-based interactions (e.g., swipe, pinch), leading to diverse and ambiguous inputs. This variability complicates static action-based matching algorithms. Therefore, simply evaluating mobile agents using deterministic smartphone states from datasets cannot uncover their true capabilities [18, 25, 26].

In general, there are two methods to evaluate mobile UI automation tasks, but neither achieves both high faithfulness and scalability. (1) The most intuitive approach is to request humans to verify the completion of tasks. However, human evaluation is difficult to reproduce [9], and the requisite human effort increases with the number of agents, tasks, and evaluation platforms. (2) The most popular approach used in most prior work [15, 25, 26, 30, 35, 37] is *exact action match on established datasets*, akin to traditional machine learning evaluations. The key idea is to ask annotators to generate a correct sequence of actions that succeed on the task as *data labels*, and then compare agent-generated actions to these labels. Although this approach allows for some error tolerance, e.g., the variations in click positions on the screen [25], it cannot cover all possible and "infinite" paths to complete a UI automation task. Consequently, it leads to a significantly higher false negative rate. For example, for the task *"Reserve a rental car in Los Angeles from June 1st-7th, with a budget of up to $60 per day on Expedia"*, the sequence of three filtering actions can be interchanged. Only taking one of these execution paths as the reference may incorrectly verify a task that is essentially completed. Moreover, LLM-powered agents are known to be able to self-correct their wrong actions [23], which is critical to enhancing UI task automation capabilities, yet is impossible to evaluate in a static dataset. These limitations are further demonstrated in §2.2.

This paper presents `LlamaTouch`, the first testbed for evaluating mobile agents in real-world mobile environments without compromising faithfulness and scalability. The key idea of `LlamaTouch` is to check the task execution trace against a few "essential states" identified by the annotators, rather than matching them against predefined action sequences in static traces. For instance, the essential states for the task *"open app Microsoft Excel (install if not already installed), go to login, ..."* should include (1) the application "Microsoft Excel" is opened, and (2) the application is on the login page. Other operations, like app installation, are considered non-essential and should be ignored. During task execution, `LlamaTouch` enables mobile agents to retrieve only task descriptions from static datasets, while device states are directly acquired from realistic mobile devices. Actions produced by mobile agents are directly operated on those devices, and all UI interaction data are recorded as task execution traces. In the evaluation phase, `LlamaTouch` compares task execution traces with annotated essential states to determine whether a task has been completed.

To ensure faithful and scalable evaluation, `LlamaTouch` integrates two effective methods. (1) `LlamaTouch` adopts a fine-grained labeling mechanism for essential state annotation at both the screen level and single UI component level. It combines pixel-level screenshots and textual screen hierarchies to explicitly highlight important UI components. With a rich set of annotation primitives provided by `LlamaTouch`, it reduces human effort to heuristically identify and annotate the attributes of essential states for evaluation, e.g., the text inside a textbox should be exactly matched. These annotated UI states are subsequently used for faithful evaluation. (2) During evaluation, `LlamaTouch` employs a multi-level state matching algorithm that combines fuzzy and exact matches on diverse annotated UI states. It uses (i) approximate screen matching, which enables `LlamaTouch` to adapt to dynamic mobile environments and varying screen contents, and (ii) mixed UI state matching, which detects and matches critical on-screen information.

Dataset and testbed. We present a large-scale dataset with pre-annotated essential states for evaluating mobile UI automation tasks in real-world mobile environments. This dataset includes 496 distinct tasks encompassing a wide array of popular Android applications. We complement this dataset with an easy-to-use testbed that enables mobile agents to interact seamlessly with realistic Android environments. This testbed provides a collection of concise, widely used APIs, ensuring compatibility with most mobile agents. Mobile agents can be easily integrated into `LlamaTouch` and use our dataset to test their capabilities in mobile UI task automation in real-world scenarios.

Evaluation. We implemented `LlamaTouch` by utilizing Google Android emulator [12] and one Google Pixel 5 smartphone as realistic Android environments. Currently, `LlamaTouch` has four built-in agents, including AutoUI [37], AppAgent [36], AutoDroid [30], and CoCo-Agent [21], along with 496 diverse tasks. With human validation results as the ground truth for task completion, `LlamaTouch` achieves nearly 80% evaluation accuracy in detecting completed tasks in real-world environments, while prior action-based evaluation methods fail to do so. We also reveal the limitations of current mobile agents in handling tasks practically in real-world environments.

Contributions are summarized as follows.

- We observed the weakness of high false negative rates in evaluating mobile UI task automation agents using static datasets. To address this, we proposed an evaluation design that only compares essential states rather than concrete action sequences.
- We devised a method for annotating essential states using a variety of annotation primitives. This approach combines visually intuitive screenshots with semantically precise view hierarchies to enable fine-grained and accurate UI component localization and annotation.
- We designed a novel task evaluation approach that employs both exact and fuzzy matching at various UI state levels. It enables faithful evaluation of mobile agents and adapts well to dynamic execution environments.
- We proposed `LlamaTouch`, the first testbed to faithfully and salably evaluate mobile UI task automation agents in real-world mobile environments. It comprises 496 tasks with human-annotated essential states. Four agents integrated in `LlamaTouch` demonstrate its faithfulness and scalability in UI automation task evaluation.

Table 1: The comparison between mobile agent benchmarks. LlamaTouch is the first testbed designed for mobile agents driven by essential state matching. LlamaTouch also supports fine-grained UI-guided essential state annotation with a rich set of primitives covering a wide array of matching implementations.

Benchmark	Platform	Real-world Tasks	Real-env Task Exec	Fine-grained UI Annotation	Essential State Match
Rico [10]	Mobile	✓	✗	✗	✗
PixelHelp [18]		✓	✗	✗	✗
AndroidEnv [28]		✗	✓	✗	✗
META-GUI [26]		✓	✗	✗	✗
MoTIF [8]		✓	✗	✗	✗
AITW [25]		✓	✗	✗	✗
Mobile-Env [38]		✓	✓	✗	✗
AndroidArena [34]		✓	✓	✗	✗
WebArena [39]	Web	✓	✓	✗	✓
LlamaTouch	**Mobile**	✓	✓	✓	✓

2 BACKGROUND AND MOTIVATION

2.1 Agents for Mobile UI Task Automation

Mobile agents have simplified the cumbersome and dull operations on smartphones for users. The progression of mobile agents for mobile UI task automation can be categorized into three phases. (1) API-based agents like Google Assistant [14] and Apple Siri [6] interact with applications through predefined application programming interfaces. This approach is reliable while limited in structured and predictable tasks. (2) Learning-based agents [18, 25, 26, 37] utilize deep learning techniques to learn from previous mobile interaction traces, but their capabilities are still confined by their training data. (3) Recently, LLMs and multi-modality LLMs have revolutionized the capabilities of mobile agents [15, 22, 29, 30]. These models, owing to their vast knowledge base, can understand complex, real-world mobile screens. Mobile agents powered by these models can accurately interpret natural language instructions and translate them into actionable tasks on smartphone screens. This evolution marks a significant leap in the flexibility and adaptability of mobile agents.

Mobile UI task automation agents typically operate with the following components.

Controller is the brain of mobile agents. It interprets task instructions and UI contexts, and then generates actions to be executed on the current UI context. Widely-used controllers include deep learning models tailored for specific applications [18, 25, 26, 37], LLMs (e.g., GPT-4, Llama) [16, 29, 30], and multi-modality LLMs (e.g., GPT-4V) [15, 35, 36].

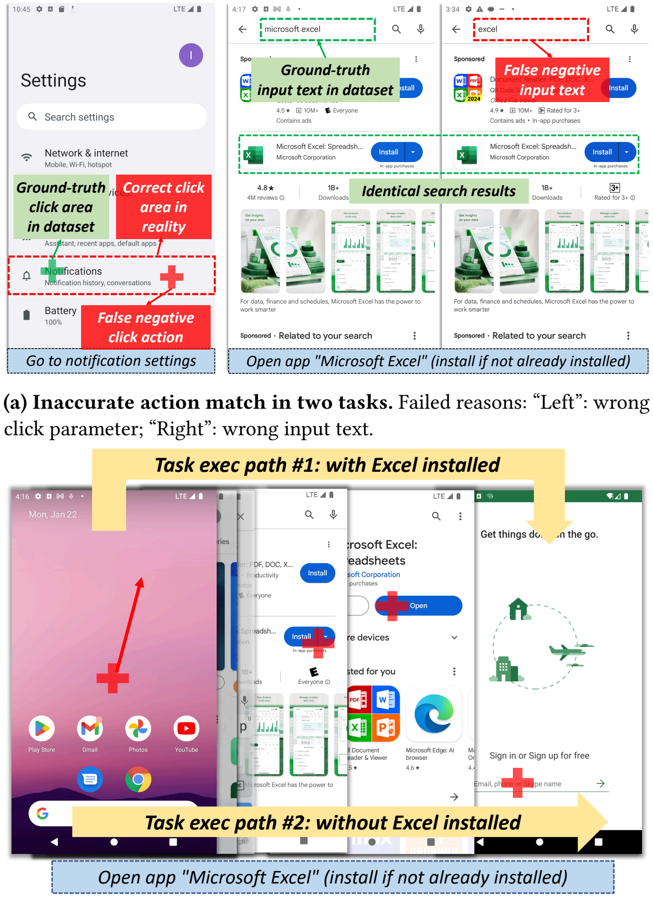
Input: UI Representation. Existing mobile agents take a task description and UI representations as the input of their controller. There are two basic types of UI representations: screenshot and view hierarchy (VH). A screenshot is a visual capture of the current screen. A VH provides a textual tree-like structure of the UI elements present on a screen, including their properties such as type, position, and text contents. On top of screenshots and VHs, some controllers further extract UI semantics to enhance UI understanding. For example, Yan et al. [35] overlay numeric tags on top of each text and icon detected by OCR tools; AXNav [27] converts screenshots to bounding boxes and labels, making them comprehensible to LLMs. Further processing based on VH, such as converting it to simple HTML representations, is also widely utilized [29, 30].

Output: Action. The output of controllers consists of actions to be executed on the current screen, such as click, swipe, and input text. Action parameters can be abstracted at different levels depending on the agent's design and input format. (1) Concrete coordinates on the screen [15, 25, 26, 37]: This operates as a direct interaction with the screen, similar to human operations. (2) Icon marker [35, 36]: The output target will specify a specific icon or graphical element within the UI representation. (3) HTML index [16, 29, 30]: By ingesting HTML representations, controllers will give a concrete HTML index as the action target, which matches specific elements (icons or text) on the screen.

2.2 Mobile UI Task Automation Benchmarks

As shown in Table 1, a variety of datasets and environments are proposed to evaluate mobile agents in UI task automation, but none of them achieve both faithfulness and scalability.

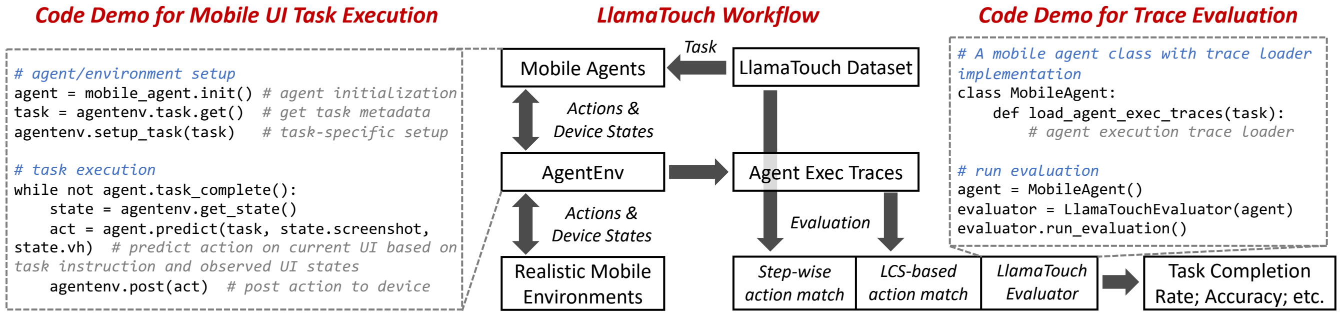
Some work such as Rico [10], PixelHelp [18], and AITW [25], provides static datasets with task descriptions, UI representations, and actions sequences. Mobile agents predict concrete actions on static UI representations, which are compared with the ground-truth actions in the datasets. While this approach is straightforward, it is


Settings
Search settings
Network & internet
Ground-truth click area in dataset
Correct click area in reality
Notifications
Battery
False negative click action
Go to notification settings
microsoft excel
Ground-truth input text in dataset
excel
False negative input text
Identical search results
Microsoft Excel: Spreadsh...
Install
Open app "Microsoft Excel" (install if not already installed)
Task exec path #1: with Excel installed
Open
Get things done on the go.
Play Store
Gmail
Photos
YouTube
Task exec path #2: without Excel installed
Open app "Microsoft Excel" (install if not already installed)

(a) Inaccurate action match in two tasks. Failed reasons: "Left": wrong click parameter; "Right": wrong input text.

(b) Different task execution paths lead to the same screen.

Figure 1: Two major limitations of evaluating mobile UI task automation on static datasets.



```
# agent/environment setup
agent = mobile_agent.init() # agent initialization
task = agentenv.task.get()  # get task metadata
agentenv.setup_task(task)   # task-specific setup

# task execution
while not agent.task_complete():
    state = agentenv.get_state()
    act = agent.predict(task, state.screenshot,
state.vh)  # predict action on current UI based on
task instruction and observed UI states
    agentenv.post(act)  # post action to device
```

```
# A mobile agent class with trace loader
implementation
class MobileAgent:
    def load_agent_exec_traces(task):
        # agent execution trace loader

# run evaluation
agent = MobileAgent()
evaluator = LlamaTouchEvaluator(agent)
evaluator.run_evaluation()
```

Figure 2: `LlamaTouch` workflow and code demonstrations for mobile UI task execution and trace evaluation. `LlamaTouch` enables mobile UI task automation agents to integrate easily with `AgentEnv` for on-device task execution with minimal programming effort. Agent execution traces recorded by `AgentEnv` are used in conjunction with the `LlamaTouch` dataset in a separate evaluation process.

insufficient to reveal the performance of mobile agents for two reasons. *(1) Inaccurate exact action match.* Functionally correct actions may be deemed incorrect due to different action parameters. Figure 1a illustrates two cases. First, for click actions, the clickable area defined in the dataset may be narrow, whereas in reality, the area might encompass the entire UI component (marked with the red bounding box). Second, non-identical text inputs can lead to the same correct search result (marked with green bounding boxes) in most search tasks. This issue has been observed in previous literature but remains unsolved [25, 35]. *(2) Lack of tolerance for different execution paths.* In real-world environments, a task can usually be completed in various paths based on different device/application states, as shown in Figure 1b. However, predefined datasets might only provide one deterministic path for reference, leading to inaccurate evaluation.

There is also other work that enables agent execution in real-world environments, such as AndroidEnv [28] and Mobile-Env [38]. However, they do not inherently support essential state match during end-to-end task execution, therefore compromising evaluation accuracy. AndroidArena [34] observed the weakness of step-wise action match on static datasets: it does not fully tolerate redundant actions in task execution paths. They proposed a subsequence-based action match, where a task is treated as completed if it contains the ground-truth action sequence as its subsequence. We take AndroidArena as a baseline in §5.3 to compare its evaluation accuracy with `LlamaTouch`. WebArena [39] provides a realistic playground for web agents. It uses essential states to evaluate task completion (e.g., the final result should be or should include some key information). `LlamaTouch` differs from WebArena on the mobile platform in both essential state annotation and evaluation process: (i) `LlamaTouch` combines visual screenshots with textual VHs of the same screens for fine-grained and precise UI component identification. (ii) `LlamaTouch` uses a richer set of primitives to comprehensively annotate essential UI states and faithfully evaluate them even with high screen content dynamics.

Human validation is usually used to validate whether a UI automation task is completed [25, 27]. However, the cost of human validation is too high, scaling poorly to multiple tasks, agents, and mobile devices. `LlamaTouch` ensures high scalability as with evaluating on static datasets while preserving faithfulness similar to human validation.

3 LLAMATOUCH DESIGN

Figure 2 shows the workflow of using `LlamaTouch` for on-device task execution and trace evaluation, using the well-constructed `LlamaTouch` dataset. Compared to previous evaluation approaches, `LlamaTouch` exhibits the following benefits.

- Practical on-device mobile UI task execution (§3.1). Most previous evaluation methods are simulated on mobile UI interaction datasets. `LlamaTouch`, on the other hand, enables mobile agents to operate in realistic mobile environments for UI task automation, revealing their true capabilities in real-world scenarios.
- Fine-grained essential application state annotation (§3.2). By observing UI automation task execution transfers states of essential UI elements within an application, `LlamaTouch` enables annotators to explicitly annotate essential application states that should be detected and matched for task completion. This approach avoids the previous static evaluation methods' focus on the determinism of traces, reducing the probability of false negatives in evaluation.
- Faithful and scalable task evaluation (§3.3). `LlamaTouch` evaluates the performance of mobile agents by comparing their task execution traces captured in real-world mobile environments with annotated essential states. By combining exact and fuzzy matching algorithms on different application states, `LlamaTouch` achieves faithful task evaluation without losing scalability.

3.1 On-device Mobile UI Task Execution

Evaluating mobile agents on predefined, deterministic traces, as discussed in Section 2.2, depicts significant inaccuracy. `LlamaTouch` empowers on-device mobile UI task execution to reveal the real capabilities of mobile agents. To simplify this process, we propose `AgentEnv`, a bridge between existing mobile agents and realistic mobile environments (e.g., smartphones, Android emulators, and cloud device farms). With `AgentEnv`, mobile agents execute tasks on real-world mobile environments by following the processes shown in Figure 3. ① Mobile agents request a task instruction from `AgentEnv`. The instruction comes from the `LlamaTouch` dataset. ② With the task instruction, mobile agents request UI representations (e.g., screenshot, VH) from `AgentEnv`, which are then ③ forwarded to mobile devices. ④ Taking the task instruction and UI representations as inputs, mobile agents predict an action to be performed on the current UI and dispatch the predicted action to `AgentEnv`. ⑤

Table 2: A set of primitives used in essential state annotation and trace evaluation.

Match Type	State Type	Primitive	Keyword	Use Case
Fuzzy match	UI state	Screen info	`fuzzy<-1>`	Verify if the contents on two screens are approximately identical.
		Textbox	`fuzzy<n>`	Verify if the content of the target textbox is semantically similar to the content of the original textbox<n> in the ground-truth UI.
Exact match		Activity	`activity`	A coarse-grained approach to determine whether two UIs represent the same functional screen in an application.
		UI component	`exact<n>`, `exclude<n>`	Verify if the UI component is exactly identical to the UI component<n>, or if it does not occur, in the ground-truth UI.
	System state	(Un)installation	`installed<app>`, `uninstalled<app>`	Verify if the target application named "app" has been successfully installed or uninstalled.
	Action	Action	`click<n>`, `type<input_text>`	Verify if two actions and their parameters are identical.

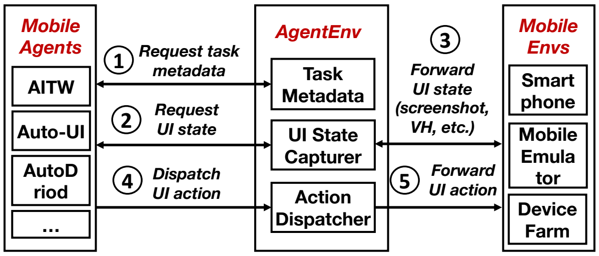


Figure 3: Interaction between mobile agents, `AgentEnv`, and real-world mobile environment in the on-device mobile UI task execution process.

`AgentEnv` forwards and executes the agent-predicted action to mobile environments. The processes from ② to ⑤ are repeated until mobile agents consider the task completed. During task execution, all UI representations and corresponding actions are captured as task execution traces for further evaluation in §3.3. These UI representations include (1) pixel-level screenshots; (2) textual screen VHs; (3) activity names of the application in the foreground of each screen; and (4) actions performed on each screen. Essential system states, such as the list of installed applications, will also be recorded for faithful mobile agent evaluation. The left-hand side of Figure 2 demonstrates the code implementation of the interaction between mobile agents and `AgentEnv`. Appendix A.1 presents details of the APIs provided by `LlamaTouch` for mobile agent integration.

3.2 Essential Application State Annotation

Two major cases demonstrated in §2.2 highlight why exact action match on predefined action sequences is unfaithful. First, it demands that the agent-generated actions and their parameters match exactly with those in the dataset. Second, it takes a fixed UI interaction sequence provided by the dataset as the reference, making it unable to evaluate alternative task execution paths.

Insight: The task execution process transfers identifiable application states. During the execution of a UI automation task, the application states change, and some of these states can be explicitly represented by UI components. Even if the task execution paths differ, there are overlaps in the essential application states. We can utilize these overlapping application states to determine whether a task achieves some milestones or is completed. As the example shown in Figure 1b, by identifying the intent of the task "open app Microsoft Excel (install if not already installed), go to login, ...", it contains two potential essential states for evaluation: (1) the application "Microsoft Excel" is opened; and (2) the application is located at the login page. Other actions, like detecting whether the application is installed, can be omitted during evaluation.

To achieve this, essential states should be accurately identified and annotated. However, simply annotating application states at the whole UI representation level (e.g., an entire screenshot) and comparing screen-level similarity [11] is too coarse-grained and may lead to inaccurate matching. For example, the screen contents of a web-shopping application could be subtly different due to nondeterministic swiping gestures or dynamically loaded contents across different executions. For accurate and efficient essential state annotation, `LlamaTouch` breaks the whole pixel-level UI representation into separate UI components. This is achieved by simplifying the textual VH of each screen, which precisely expresses the attributes of every UI element, to extract important, visible UI components. These extracted UI components are combined with visually intuitive screenshots to provide precise overlayed bounding boxes and unique identifiers to annotators. Figure 4 shows an example of VH-enhanced screenshots in a task provided to annotators.

Annotation primitives. `LlamaTouch` incorporates a list of primitives for essential state annotation. These primitives represent essential information on the screen and indicate how this information should be matched. Annotators are responsible for clearly identifying and using primitives to represent application states that are informative and deterministic for validating task execution results. Table 2 comprehensively shows these primitives and their use cases. Currently, `LlamaTouch` incorporates six types of primitives and nine keywords for annotation. These primitives can be divided into three types according to the application state type they represent.

- *UI state.* The purpose of UI state annotation is to extract and compare whether two screens contain identical or similar information. As a screen may contain different types of components, `LlamaTouch` uses four major primitives to cover the comparison logic of all these UI components. (1) *Screen info* and (2) *textbox* are used to compare whether the whole screen or a dedicated textbox has similar contents. *Screen info* is primarily used for checking whether two UI representations are within the same screen (i.e., the same page of an application). *Textbox* primitive is used to annotate textboxes that may contain dynamic contents, such as a search box in a web-shopping application or the URL of a website. (3) The functionality of *activity* is like *screen info*: it can be used to approximately detect whether two UI representations are on the same screen of an application. (4) *UI component* is used to deterministically annotate the state of a UI component, including textbox,

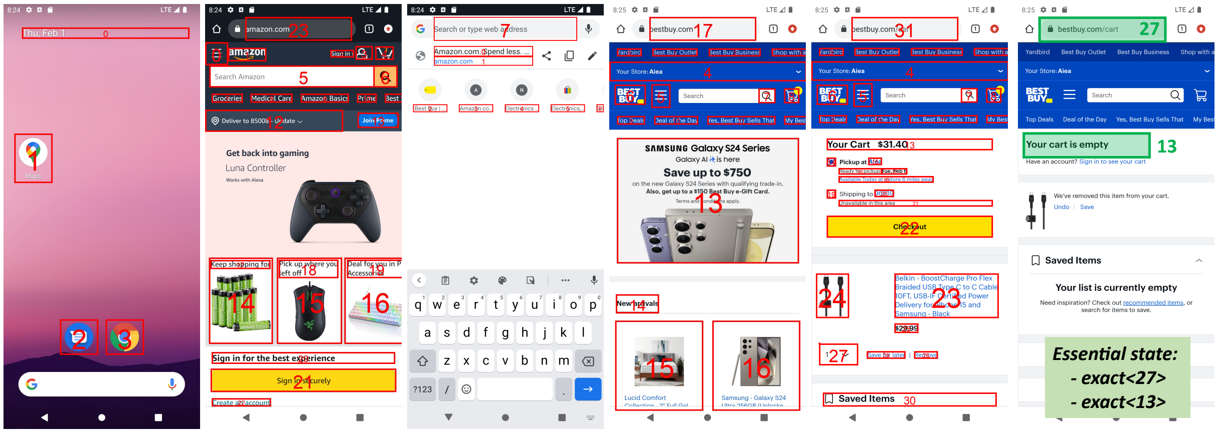


Figure 4: Annotated essential states for the task "empty the shopping cart on bestbuy" in the last UI representation: two textboxes with the exact keyword. The essential states represent the application state after task execution: "the shopping cart on bestbuy is empty".

button, image, etc. For example, if the content of a textbox should be matched, this textbox should be annotated with the exact keyword. States that require certain UI components not to occur on the screen can be annotated with the exclude keyword.

- *System state.* System state annotation in LlamaTouch is primarily inspired by tasks from the previously built dataset: AITW [25]. For example, a task like "install app YouTube Kids" can be detected directly using shell commands to access system states (e.g., pm list packages), without involving complex UI states. Currently, LlamaTouch supports programmatically checking application installation status. The keywords can be easily extended and customized to detect other system or application states.
- *Action.* Although the principle of annotation is to detect application state transfer during task execution. Sometimes, action on a specific UI representation is necessary to validate agent behavior, especially when there are not enough state identifiers shown in the UI. LlamaTouch provides click and type keywords that cover the most common actions.

Case study. Essential state annotation can be divided into the following processes. First, for each UI representation, LlamaTouch overlays all functional UI components with numeric markers on the screenshot as shown in Figure 4. This is done by extracting the precise metadata of each UI component from textual VH. Annotators will then identify the essential states that should be checked during evaluation to ensure the task is completed. LlamaTouch simplifies this process by only requiring annotators to explicitly identify what UI element should be matched and what annotation primitive should be used. For example, after emptying the shopping cart on bestbuy, the screen will display a textbox with the content "Your cart is empty". This textbox should be treated as an essential state, which is highlighted in a bounding box with numeric ID 13 in the last screen of Figure 4, as it represents the state after task completion. As we anticipate the content of the textbox should be exactly matched, the annotated keyword is *exact<13>*. The keyword *exact<27>* is used to validate whether two screens are both in the shopping cart of bestbuy. All annotations, along with the UI representations and task descriptions, construct the essential state-powered dataset. These essential states are used by LlamaTouch for faithful task evaluation.

3.3 Faithful and Scalable Task Evaluation

Annotated essential states (§3.2) and captured task execution traces (§3.1) are jointly used for evaluation in LlamaTouch. LlamaTouch iterates through essential states to determine whether a task execution trace sequentially matches all annotated states. If so, the task is deemed completed. To achieve faithful evaluation, the most significant challenge lies in how to ensure the task execution trace matches the essential states. Task execution traces captured from real-world mobile environments may contain dynamic screen content; it is vital to adapt the annotated essential states from a static dataset to varying screens captured from the real world, while achieving precise matches on only critical information. To address this problem, LlamaTouch employs a multi-level state matching algorithm, which combines fuzzy match and exact match on both the entire screen and separated UI components to ensure faithful evaluation. The algorithm first approximately matches two screens according to their UI representations and activities. Then, within two matched screens, it compares each annotated essential state with iterated UI components in the target screen based on their annotated primitives. A task is considered completed only when all annotated essential states have matching counterparts. The matching philosophy of all annotation primitives is as follows.

Approximate screen match is used to ensure two screens are on the same functionality page of an application, even under high screen content dynamics. This is the preliminary process for in-screen exact/fuzzy content match. To achieve this, LlamaTouch utilizes two annotation primitives proposed in §3.2: *activity* and *screen info*.

1. Application activity match. Activity represents an entry point for users to interact with the application [3]. This is a clear identifier to indicate whether two screens are in the same application. Screens with distinct functionalities typically have unique activity names,

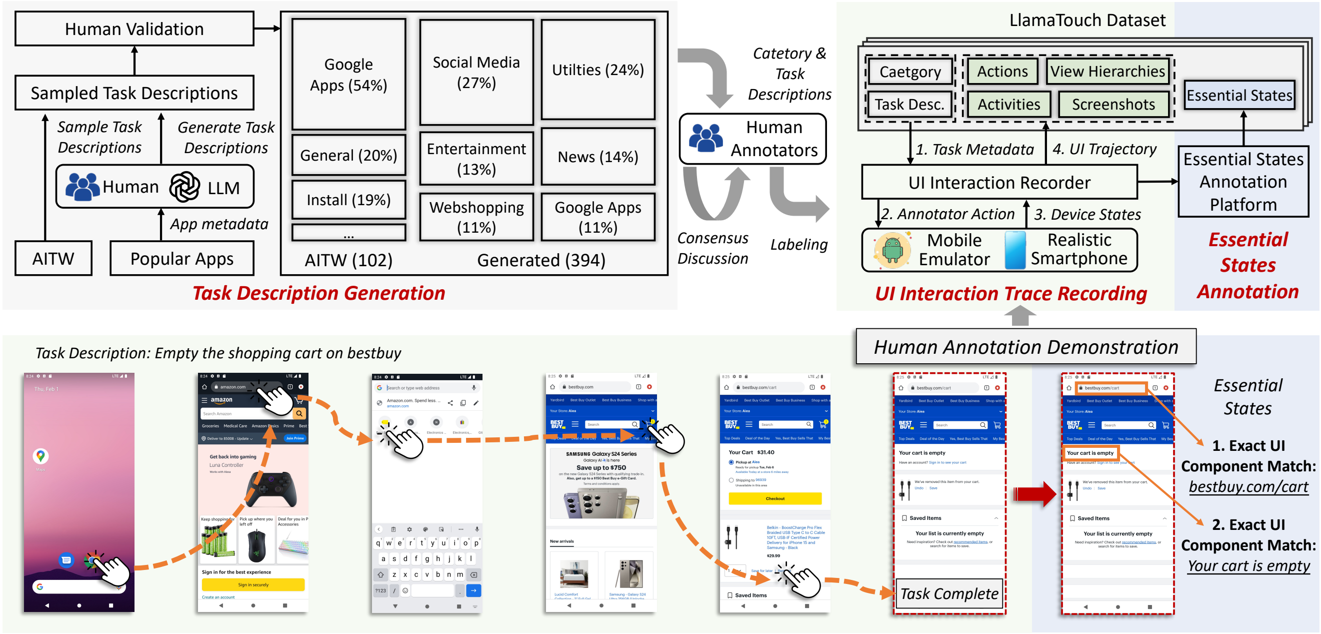


Figure 5: Workflow for constructing the LlamaTouch dataset. Generally, the workflow has three independent processes. (1) Generate task descriptions by sampling from previous datasets or constructing new ones through humans or LLMs. (2) Record UI interaction traces according to task descriptions. (3) Annotate essential states atop UI interaction traces.

even if they are within the same application. For example, the activity of the main settings page is "com.android.settings.Settings", while the Wi-Fi settings page has the activity "com.wifiadmin.settings.WifiSettingsActivity". Typically, exact application activity match acts as a foundational filtering process to identify whether two screens are on the same page. However, some specific application design philosophies may cause different functional pages of an application to contain the same activity name [2]. Under these circumstances, LlamaTouch involves *screen info* for accurate screen match.

2. Fuzzy screen info match is utilized when the activity name cannot differentiate between functional pages of an application. To better compare two screens, it is crucial to extract only the critical information from the screen. In summary, LlamaTouch simplifies the textual screen VH to a simple HTML representation, as in prior work [29, 30], while preserving the types of every UI component. LlamaTouch compares two simplified HTML representations of the screens using the cosine similarity of their sentence embeddings. Two screens are deemed similar when their cosine similarity exceeds a predefined threshold, e.g., 0.85 in our experiments. The fuzzy screen info match design helps LlamaTouch maintain faithfulness during evaluation when dealing with dynamic screen contents.

Mixed UI state match will be applied to matched screens after the approximate screen match process. A mix of annotated UI states will be checked in this phase, including both fuzzy match and exact match on UI components, actions, and system states.

1. Exact UI component match requires that an annotated UI component be identical in two screens being compared, including all their attributes such as *class*, *text*, and *selected* in VH. This is especially useful for evaluating the content of a textbox, the status of a button (e.g., checked or not), or a selectable icon on the screen. Given a target UI component to be exactly matched, LlamaTouch will iterate through nodes in the VH of the matched screen until a matching UI component is found. Exact UI component match fails if no matching node is found on available screens. In our dataset, exact UI component match accounts for 51% (698 out of 1,379) of annotated essential states.

2. Fuzzy textbox match is crucial for comparing the content inside a textbox on the screen, especially when the content may be slightly different. Semantically similar search keywords with the same intent that comply with a specific task description should be matched, as they will lead to the same results. For example, Figure 1b shows searching for "Microsoft Excel" and "Excel" in the Google Play Store both display the target application. LlamaTouch extracts the content of the annotated textbox and then compares the text with nodes in the matched screen using the same approach as in *fuzzy screen info match*.

3. Action match. Although the initiative of LlamaTouch is to detect state transfer during task execution, there are still cases for evaluating concrete actions performed on the screen, such as clicking a specific UI component or typing the correct captcha. LlamaTouch directly compares the actions and their parameters of the annotated actions with their counterparts on the matched screens. Different from prior studies [25], LlamaTouch uses XPath [1] of target UI components extracted from screen VHs as parameters for click actions. This approach provides more tolerance in the action space compared to using precise coordinates.

4. System state match is usually more efficient and accurate than merely comparing UI states for specific tasks that involve deterministic system states, e.g., installed applications. LlamaTouch currently supports checking whether an application is installed or not. Such system states are recorded during task execution on real-world mobile devices (§3.1). LlamaTouch will check whether the annotated system state is identical to that of the last screen in the task execution trace, which is also recorded by AgentEnv.

Through the multi-level state matching algorithm, LlamaTouch achieves high evaluation accuracy on real-world task execution traces, while preserving the scalability of evaluating on static datasets. The above evaluation logic is well encapsulated into the LlamaTouch evaluator. To use it for evaluation, an agent only needs to define how to load the task execution traces for each task, as shown in the right side of Figure 2, The evaluator will automatically conduct the evaluation and report metrics such as task completion rate. Experiments in §5 show the faithful evaluation of LlamaTouch.

4 LLAMATOUCH DATASET

4.1 Dataset Construction

The LlamaTouch dataset consists of a combination of tasks from the previous AITW dataset [25] and self-generated tasks that involve diverse categories and popular applications. The inclusion of new tasks in currently popular applications, which have not been covered in previous literature, aims to properly assess the generality and real capabilities of mobile agents. Figure 5 demonstrates the workflow for constructing the dataset, where each data sample undergoes the following processes.

Generate task descriptions. Task descriptions are generated by both humans and LLMs using app metadata (e.g., app names, categories, descriptions) from popular apps in the Google Play Store. For tasks in the AITW dataset, we sample parts of them after deduplicating similar task descriptions. The sampled task descriptions are then validated by humans to avoid duplication, infeasibility, and high complexities that may far exceed the capabilities of human and mobile agents. After this process, we sampled 102 tasks from the AITW dataset among 26 unique apps and 394 newly generated tasks among 46 unique apps. The generated descriptions cover a variety of application categories such as utilities (e.g., Zoom, Expedia), social media (e.g., Discord, Instagram), and web shopping (e.g., Walmart, Amazon). The left-hand side of Figure 5 shows the proportion of these categories.

We then employ six human annotators, all of whom are authors of this study and experts in smartphone usage, to generate data samples in LlamaTouch through the following two independent annotation stages. The image on the lower side of Figure 5 illustrates the outputs of the human annotation process.

Record UI interaction traces. The validated task descriptions are used as guidance for recording UI interaction traces. Tasks sampled from the AITW dataset are also required to go through this process as they lack view hierarchies. Given a task description, human annotators interact with mobile apps through our developed *UI Interaction Recorder*. This tool is built on top of mobile emulators or realistic smartphones and displays the graphical user interface to users, allowing them to operate it like normal smartphones. Specifically, human annotators are asked to complete a task in the simplest manner and to avoid redundant operations, as in [25]. As shown in Figure 5, the task "Empty the shopping cart on bestbuy" requires five continuous click actions to complete. The recorder captures actions, VHs, activities, and screenshots, which collectively form the UI trajectory.

Annotate essential states. The recorded UI interaction traces, along with task descriptions, are then processed by human annotators to identify essential states. To help annotators better understand the application state transformation and simplify the annotation process, we developed an essential state annotation system. This system displays the entire UI interaction trace, with potentially significant UI components shown with numeric indices in each screenshot. Annotators are asked to identify the most significant, identifiable states that represent key milestones during task completion. They annotate these states (their numeric indices) with the proper essential state primitives we proposed in §3.2. These essential states, together with recorded UI interaction traces, form the final LlamaTouch dataset.

During the annotation process, to ensure data reliability, anything that a single human annotator cannot decide on is annotated based on a consensus reached by three or more annotators. Overall, this dataset includes 496 tasks, covering 57 unique Android applications with diverse task complexities.

4.2 Dataset Statistics

In this section, we present statistics of the LlamaTouch dataset. Table 3 quantifies task complexities by showing the average steps (actions) required to complete a task. Tasks from AITW [25] are slightly more complex than those generated by LlamaTouch, with average steps of 7.35 versus 5.67, respectively. Overall, the average number of steps to complete a task is nearly 7, with task complexities ranging from 2 to 42 steps.

Table 3: LlamaTouch dataset statistics and task complexities measured by the average steps (actions) to complete a task.

Category	# Task	# Apps	Avg. Steps
AITW [25]	102	26	7.35 (2-19)
Generated	394	46	5.67 (3-42)
Total	496	57	7.01 (2-42)

We further analyze action types in the ground-truth dataset. Table 4 shows the statistics of actions contributing to the dataset. In summary, all 496 tasks involve *click* actions. Out of these, 292 tasks involve *swipe* actions, and 147 tasks involve *input* actions. A few tasks also use press *home* and *back* for navigation. By dividing the total number of actions by the number of tasks involving these actions, we observe an average of more than 4 *click* actions per data sample. The average number of all other actions is less than 2.

Statistics in Table 5 detail the annotated essential states. We find that the essential states of 441 tasks can be presented on only one UI representation, indicating that most tasks check only the final states after task completion. There are 53 tasks and 2 tasks with essential states presented on two and three UI representations, respectively. Among the annotated essential states, *exact UI component match* accounts for 51% of all essential states, followed by *exact activity match* at 36%. More than 84% of tasks (418 out of 496) have at

Table 4: Action statistics in LlamaTouch dataset.

Action	# Action	# Tasks W/ Action	Mean/Stddev
Click	2,192	496	4.42/2.51
Swipe	376	292	1.29/0.9
Input Text	173	147	1.18/0.55
Press Home	44	43	1.02/0.15
Press Back	6	5	1.2/0.45

Table 5: Statistics of annotated essential states.

Type	Essential State (ES)	# ES	# Tasks W/ ES	Mean/Stddev
Exact Match	UI Component	698	418	1.67/0.92
	Activity	490	442	1.11/0.32
	Action: Click	98	93	1.05/0.23
	Action: Type	1	1	1/0
	System: Install	7	7	1/0
	System: Uninstall	3	3	1/0
Fuzzy Match	UI Component	51	44	1.16/0.43
	Screen Info	31	31	1/0

least one of these two types of essential states. Fuzzy match is also important for the evaluation process: 44 tasks have 51 *fuzzy UI component match* in total; *fuzzy screen info match* occur in 31 tasks. In summary, all types of essential states construct the dataset, which significantly contributes to the faithful evaluation of mobile UI task automation, as we will show in the following section.

5 EVALUATION

5.1 Experiment Setup

Mobile environments. LlamaTouch primarily utilizes an x86-64 Android emulator [12] as the mobile environment for task execution. The Android emulator is configured with Android 12 (API level 31) with Google Play Services deploying on a Linux server running Ubuntu 18.04 OS. Tasks involving applications that are not compatible with the Android emulator (e.g., Snapchat) are tested on a real Google Pixel 5 device with Android 14 OS.

Task setup. In this study, tasks are set up in two stages. First, we construct an Android system image for the Android emulator, which provides a concrete environment where partial task states are prepared. This ensures a controlled and replicable starting point for the tasks. Second, we develop setup scripts using the Android UIAutomator2 library [4] to manage complex setup processes with specific constraints (e.g., require manual login or run on real-world smartphones). These scripts automate the initialization of the environment, ensuring tasks are ready for execution with minimal manual intervention.

Mobile agents. We selected four mobile agents that cover diverse types of brains, including supervised learning models, LLM, and large multi-modality models.

- Auto-UI [37] employs a multimodal transformer model based on BLIP-2 [17] and T5 [24] variants as the brain for decision-making. It takes pixel-level screenshots and task descriptions as input.
- AutoDroid [30] uses LLMs for device control. It first takes textual screen VH and simplifies it to an explicit, readable HTML representation. The simplified HTML representation is then processed by LLMs to generate the corresponding action. We selected GPT-4 (gpt-4-0125-preview) [5] as the backend of AutoDroid.
- AppAgent [36] utilizes GPT-4V [22] to comprehend the screenshots of mobile devices and then dispatch controlling commands to LlamaTouch. In our evaluation, we used the AppAgent version without document pre-exploration for simplicity.
- CoCo-Agent [21] uses LLaVA [20] as the brain. It is trained on a subset of the original AITW dataset.

Evaluation methodologies. We used the following methodologies for evaluating whether an agent completes a task.

- *Step-wise action match* is widely-utilized in evaluating mobile agents on well-established datasets [25, 26, 30, 37]. It compares the agent-generated action sequences in real-world environments with ground-truth action sequences in the dataset. When two action sequences are identical, the task is treated as completed. We utilize the action match algorithm in AITW [25]: two actions are matched only if they have identical action types and parameters.
- *Longest common subsequence (LCS)-based action match* is proposed in AndroidArena [34]. It is built upon step-wise action match by tolerating redundant actions between ground-truth ones. Two action sequences are matched when the execution action sequence contains the ground-truth sequence as the subsequence.
- LlamaTouch that compares agent execution traces with our annotated essential states to check whether a task is completed.
- *Human* validates whether a task is completed based on the agent execution traces. Humans are instructed not only to focus on the action (and its parameter) on each UI representation, but also the whole application state transfer during task execution. The results of human validation are treated as ground truth for comparing the accuracy of the three evaluation approaches.

5.2 Metrics

We primarily compare (1) the end-to-end task completion rate (TCR) of different mobile agents and (2) the accuracy of different evaluation approaches, using human validation results as the ground truth.

End-to-end TCR. Executing tasks in realistic mobile environments results in two outcomes: task completion and task non-completion. In step-wise action match, a task is considered completed only when the two action sequences are identical. LCS-based action match assesses whether a ground-truth action sequence is a subsequence of the agent-generated action sequence; if so, the task is evaluated as completed. In LlamaTouch, a task is considered completed when the agent execution trace passes through all essential states in the ground-truth dataset, using the proposed algorithm in §3.3. The end-to-end TCR of a certain evaluation method is calculated as the number of tasks evaluated completed divided by the total number of tasks in the dataset.

Evaluation accuracy. In this study, we use human validation results on agent execution traces as the ground truth for task completion. Assume N denotes the ground-truth dataset encompassing all tasks. For task n in N, we use H_n to represent the human validation result based on the agent execution trace, where H_n can be either "True" or "False". When evaluating the same agent execution trace using a specific evaluation method, the outcome is represented by E_n, which can also be "True" or "False". Therefore, the accuracy formula is defined as:

$$\text{accuracy} = \frac{\sum_{n \in N} \left(\delta_{E_n, H_n}\right)}{|N|} \tag{1}$$

The formulation indicates that a task is accurately evaluated only when the evaluation result E_n matches the human validation result H_n. Consequently, the accuracy of an evaluation approach is defined as the proportion of correctly evaluated tasks to the total number of tasks in the dataset $|N|$.

5.3 Task Completion Rate and Accuracy

Table 6 presents the end-to-end TCR and accuracy of different evaluation designs. All three approaches achieve more than 90% accuracy on average. However, the step-wise action match and LCS-based action match fall short in recognizing tasks correctly executed by mobile agents, resulting in nearly 0% TCR. The high accuracy rates of these two approaches are attributed to the large portion of incomplete tasks; only 6% of tasks are deemed completed upon human validation. Compared to action match on static datasets, the TCR evaluated by LlamaTouch is 8.67%, which closely aligns with the results of human validation.

To demonstrate LlamaTouch's effectiveness in addressing the issue of false negative results pervasive in previous evaluation methods, we focused on tasks considered successfully completed by human evaluation, excluding all incomplete tasks. Table 7 displays the number of tasks completed by agents and the evaluation accuracy for these tasks. Overall, agents successfully completed 30 tasks on average. Among these tasks, both step-wise action match and LCS-based action match achieve no more than 0.1% evaluation accuracy. This indicates they are unable to faithfully evaluate tasks executed in real-world environments using static datasets, primarily because a single static action sequence is difficult to match with multiple possible paths to task completion in real-world environments. Notably, LlamaTouch exhibits an average accuracy of 79% in validating these task execution traces, significantly reducing the percentage of false negative cases observed with other evaluation methods. The findings uncover LlamaTouch's proficiency in evaluating UI automation tasks under real-world settings.

5.4 Ablation Study

LlamaTouch achieves high accuracy when evaluating agent execution traces in real-world environments, attributed to the annotation and implementation of various essential state primitives. In this section, we evaluate the effectiveness of two types of match designs in LlamaTouch: fuzzy match and exact match. We aggregate the execution traces of all mobile agents evaluated in §5.3. We evaluate the system by first disabling exact match and fuzzy match separately, and then enabling different primitives one at a time. We present the results for (1) all tasks, (2) tasks in AITW [25], and (3) new tasks generated in LlamaTouch. The results for task completion rate and accuracy are shown in Table 8.

Table 6: End-to-end task completion rate (TCR %) and accuracy (Acc. %) of different evaluation approaches of all tasks.

Mobile Agent	Step-wise action match		LCS action match		LlamaTouch		Human
	TCR	Acc.	TCR	Acc.	TCR	Acc.	TCR
AutoUI	0.00	98.18	0.00	98.18	4.44	96.57	1.82
AutoDroid	0.00	85.98	0.00	85.98	14.84	91.87	14.02
AppAgent	0.00	93.33	0.61	93.13	10.91	94.95	6.67
CoCo-Agent	0.00	97.97	0.00	97.97	4.47	96.34	2.03
Average	0.00	93.86	0.15	93.81	8.67	94.93	6.14

Table 7: Accuracy (Acc. %) of different evaluation approaches among all successful tasks in human validation.

Mobile Agent	Step-wise action match	LCS action match	LlamaTouch	Human
	Acc.	Acc.	Acc.	# success
AutoUI	0.00	0.00	77.78	9
AutoDroid	0.00	0.00	73.91	69
AppAgent	0.00	3.03	93.94	33
CoCo-Agent	0.00	0.00	70.00	10
Average	0.00	0.76	78.91	30

Exact match ensures key information on two screens is identical. As shown in the results, exact match significantly contributes to improving evaluation accuracy. For example, without all exact match primitives, LlamaTouch's evaluation accuracy on all tasks drops from 95% to 19%. Among all exact match primitives, *activity* match greatly improves evaluation accuracy. This is due to its simple yet efficient ability to locate functional application screens and the large proportion of annotated activity primitives, comprising 35% of all primitives in our dataset. *UI component match* is also necessary for faithful evaluation in LlamaTouch, as it is typically with screen location primitives such as *activity* to detect critical information within the matched screen. Exact match for *action* and *system state* only improves accuracy slightly because only a few tasks are annotated with these primitives.

Fuzzy match does not show a notable improvement in evaluation accuracy, despite the presence of 30 screen-level fuzzy match primitives and 52 textbox fuzzy match primitives in the dataset. Few tasks with these primitives can be completed by the four agents we evaluated, leading to only a slight improvement. Although primitives for fuzzy match are not well explored, they play an important role in dealing with screen content or UI layout dynamics in the real world. With more performant mobile agents in the future, they will be further explored and evaluated.

5.5 Absolute Capabilities of Mobile Agents

In this section, we present the absolute capabilities of different mobile agents in mobile UI automation tasks. First, we categorize tasks according to their sources: AITW and our self-generated dataset. This aims to determine whether an agent that has previously learned on a well-established dataset can adapt to new

Table 8: Ablation study on different essential state primitives.

Evaluation design	All tasks		AITW		Generated	
	TCR	Acc.	TCR	Acc.	TCR	Acc.
Complete LlamaTouch	8.67	94.93	17.46	89.43	6.38	96.36
LlamaTouch W/O exact match	86.77	18.85	82.55	29.75	87.87	16.02
+ activity exact match	23.21	81.60	41.05	68.29	18.58	85.06
+ action exact match	86.62	19.01	81.81	30.49	87.87	16.02
+ UI component exact match	15.86	88.15	40.30	68.55	9.51	93.24
+ system state exact match	85.41	20.22	75.91	36.40	87.87	16.02
LlamaTouch W/O fuzzy match	10.54	93.26	23.36	84.02	7.21	95.66
+ screen-level fuzzy match	10.24	93.36	22.13	84.76	7.15	95.60
+ textbox fuzzy match	8.97	94.83	18.69	88.69	6.45	96.43

Table 9: The comparison of task completion rate categorized by task sources (i.e., from AITW and LlamaTouch generated tasks).

Agent	End-to-end TCR		
	Overall	AITW	Generated
Auto-UI	4.44	12.75	2.29
AutoDroid	14.84	22.77	12.79
AppAgent	10.91	21.57	8.14
CoCo-Agent	4.47	12.75	2.31

applications/tasks. Second, we categorize tasks according to their complexities, measured by the number of steps required to complete a task in the ground-truth dataset. Tasks are further divided into three difficulty levels: easy (steps ≤ 4), medium (4 < steps ≤ 8), and high (step > 8).

Performance across datasets. Among the four evaluated agents, AutoUI and CoCo-Agent were previously trained on AITW. AutoDroid and AppAgent directly invoke GPT-4 and GPT-4V, respectively. We separately show the end-to-end TCR of all agents on tasks in AITW and LlamaTouch's generated tasks to demonstrate their performance generalization to new scenarios (i.e., new tasks and new applications). Results in Table 9 indicate that AutoDroid and AppAgent also achieve higher TCR in AITW tasks compared to LlamaTouch's generated tasks. This is mainly attributed to the task complexity gap between the two datasets. AutoUI and CoCo-Agent achieve a 12.75% TCR on tasks in AITW. However, for generated tasks in LlamaTouch, both perform poorly, with only 2.29% and 2.31% TCR, respectively. The TCR gap between datasets reveals their lack of capability to adapt to previously unseen tasks. Considering the vast number of real-world applications, mobile agents with strong generalization capabilities to unseen scenarios are more competitive.

Table 10: The comparison of task completion rate between mobile agents. Tasks are categorized into different difficulty levels according to the number of steps required to finish it in the ground-truth dataset.

Agent	End-to-end TCR			
	Overall	Steps≤4	4<Steps≤8	Steps>8
Auto-UI	4.44	4.95	4.19	4.82
AutoDroid	14.84	27.00	12.94	7.32
AppAgent	10.91	16.83	10.97	3.61
CoCo-Agent	4.47	7.92	3.91	2.41

Performance under different task complexity. We categorize tasks into three difficulty levels based on the number of steps required to complete them in the datasets. The results are shown in Table 10. Generally, mobile agents can better complete simple tasks that require fewer steps. There is a significant drop in TCR when task complexity increases. AutoDroid outperforms the other three agents for tasks at all difficulty levels. AppAgent achieves a TCR similar to AutoDroid across all tasks. We believe that the knowledgeable GPT-4/4V can better interpret task descriptions in natural language and pixel-level or textual UI representations. Therefore, agents based on GPT-4/4V achieve high TCR. However, there is still considerable room for mobile agents to improve their capabilities in mobile UI task automation.

6 LIMITATIONS OF LLAMATOUCH

Supporting WebView-based apps. One limitation of LlamaTouch is that its evaluation process requires Android's VH to empower approximate UI layout matching and accurate UI component matching (§3.2). Some Android applications built with WebView [13] are unable to access their VH, preventing LlamaTouch from evaluating agent performance on these applications. However, only a minor number of applications are built with WebView, inspiring LlamaTouch's solution in utilizing VH. When evaluating WebView-based apps are required, more advantaged techniques, such as screen similarity detection [31, 32], OCR/model-powered screen element recognition [7, 33], should be incorporated to retrofit LlamaTouch. We will consider this in future work.

Biases in identifying essential states. LlamaTouch requires human or other autonomous agents, e.g., GPT-4V, to identify and annotate essential states on predefined UI interaction traces. This process, however, could involve biases and leads to inaccuracy due to potentially limited knowledge in application execution and unseen circumstances. For example, given a task "Delete YouTube in the Google Play Store", the predefined traces might explicitly navigate to the app page of "YouTube" and click the "Uninstall" button, which might be annotated as one essential state during task execution. However, they may ignore that this application might be not installed at all. In this case, there will be no explicit action of "clicking the uninstall button" while this task is still completed. Such biases might result in low accuracy in checking task completion rates. However, we think this limitation could be eliminated and refined by involving expert reviewing. Furthermore, for every single task description, involving diverse essential states on different potential task execution paths could better enhance the robustness of LlamaTouch's evaluation design.

7 CONCLUSION

In this work, we proposed LlamaTouch, the first testbed for evaluating mobile agents with both faithfulness and scalability in mobile UI task automation. LlamaTouch enables mobile agents to be tested on realistic mobile environments. At the evaluation stage, it matches the task execution traces with annotated essential states. LlamaTouch tolerates different task execution paths and dynamic execution environments, significantly reducing false negative results that occurred in previous evaluation approaches. It achieves high evaluation accuracy, which is comparable to human validation, while preserving the scalability of evaluating on static datasets. By conducting task execution to real-world mobile devices, we also reveal the limited capabilities of current mobile agents in mobile UI automation tasks.

ACKNOWLEDGMENTS

This work was supported by National Key R&D Program of China (No.2021ZD0113001), NSFC (No.62102045 and No.62272261), and China Institute of IoT (Wuxi).

REFERENCES

[1] 2017. XML Path Language (XPath) 3.1. https://www.w3.org/TR/xpath-31/.

[2] 2018. Single activity: Why, when, and how (Android Dev Summit '18). https://www.youtube.com/watch?v=2k8x8V77CrU.

[3] 2024. Activity | Android Developers. https://developer.android.com/reference/android/app/Activity.

[4] 2024. Android UIAutomator2. https://github.com/appium/appium-uiautomator2-driver.

[5] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774* (2023).

[6] Apple. 2024. Siri - Apple. https://www.apple.com/siri/.

[7] Sara Bunian, Kai Li, Chaima Jemmali, Casper Harteveld, Yun Fu, and Magy Seif Seif El-Nasr. 2021. Vins: Visual search for mobile user interface design. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*. 1–14.

[8] Andrea Burns, Deniz Arsan, Sanjna Agrawal, Ranjitha Kumar, Kate Saenko, and Bryan A Plummer. 2022. A dataset for interactive vision-language navigation with unknown command feasibility. In *European Conference on Computer Vision*. Springer, 312–328.

[9] Cheng-Han Chiang and Hung-yi Lee. 2023. Can Large Language Models Be an Alternative to Human Evaluations? *arXiv preprint arXiv:2305.01937* (2023).

[10] Biplab Deka, Zifeng Huang, Chad Franzen, Joshua Hibschman, Daniel Afergan, Yang Li, Jeffrey Nichols, and Ranjitha Kumar. 2017. Rico: A mobile app dataset for building data-driven design applications. In *Proceedings of the 30th annual ACM symposium on user interface software and technology*. 845–854.

[11] Shirin Feiz, Jason Wu, Xiaoyi Zhang, Amanda Swearngin, Titus Barik, and Jeffrey Nichols. 2022. Understanding screen relationships from screenshots of smartphone applications. In *27th International Conference on Intelligent User Interfaces*. 447–458.

[12] Google. 2023. Run apps on the Android Emulator | Android Developers. https://developer.android.com/studio/run/emulator.

[13] Google. 2024. Build web apps in WebView. https://developer.android.com/develop/ui/views/layout/webapps/webview.

[14] Google. 2024. Google Assistant, your own personal Google. https://www.apple.com/siri/.

[15] Wenyi Hong, Weihan Wang, Qingsong Lv, Jiazheng Xu, Wenmeng Yu, Junhui Ji, Yan Wang, Zihan Wang, Yuxiao Dong, Ming Ding, et al. 2023. CogAgent: A Visual Language Model for GUI Agents. *arXiv preprint arXiv:2312.08914* (2023).

[16] Sunjae Lee, Junyoung Choi, Jungjae Lee, Hojun Choi, Steven Y Ko, Sangeun Oh, and Insik Shin. 2023. Explore, Select, Derive, and Recall: Augmenting LLM with Human-like Memory for Mobile Task Automation. *arXiv preprint arXiv:2312.03003* (2023).

[17] Junnan Li, Dongxu Li, Silvio Savarese, and Steven Hoi. 2023. Blip-2: Bootstrapping language-image pre-training with frozen image encoders and large language models. *arXiv preprint arXiv:2301.12597* (2023).

[18] Yang Li, Jiacong He, Xin Zhou, Yuan Zhang, and Jason Baldridge. 2020. Mapping natural language instructions to mobile UI action sequences. *arXiv preprint arXiv:2005.03776* (2020).

[19] Yuanchun Li, Hao Wen, Weijun Wang, Xiangyu Li, Yizhen Yuan, Guohong Liu, Jiacheng Liu, Wenxing Xu, Xiang Wang, Yi Sun, et al. 2024. Personal LLM Agents: Insights and Survey about the Capability, Efficiency and Security. *arXiv preprint arXiv:2401.05459* (2024).

[20] Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. 2023. Visual Instruction Tuning. arXiv:2304.08485 [cs.CV]

[21] Xinbei Ma, Zhuosheng Zhang, and Hai Zhao. 2024. Comprehensive Cognitive LLM Agent for Smartphone GUI Automation. *arXiv preprint arXiv:2402.11941* (2024).

[22] OpenAI. 2023. GPT-4V(ision) system card. https://openai.com/research/gpt-4v-system-card.

[23] Lihang Pan, Bowen Wang, Chun Yu, Yuxuan Chen, Xiangyu Zhang, and Yuanchun Shi. 2023. AutoTask: Executing Arbitrary Voice Commands by Exploring and Learning from Mobile GUI. *arXiv preprint arXiv:2312.16062* (2023).

[24] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *The Journal of Machine Learning Research* 21, 1 (2020), 5485–5551.

[25] Christopher Rawles, Alice Li, Daniel Rodriguez, Oriana Riva, and Timothy Lillicrap. 2023. Android in the wild: A large-scale dataset for android device control. *arXiv preprint arXiv:2307.10088* (2023).

[26] Liangtai Sun, Xingyu Chen, Lu Chen, Tianle Dai, Zichen Zhu, and Kai Yu. 2022. META-GUI: Towards Multi-modal Conversational Agents on Mobile GUI. *arXiv preprint arXiv:2205.11029* (2022).

[27] Maryam Taeb, Amanda Swearngin, Eldon School, Ruijia Cheng, Yue Jiang, and Jeffrey Nichols. 2023. AXNav: Replaying Accessibility Tests from Natural Language. *arXiv preprint arXiv:2310.02424* (2023).

[28] 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* (2021).

[29] Bryan Wang, Gang Li, and Yang Li. 2023. Enabling conversational interaction with mobile ui using large language models. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*. 1–17.

[30] Hao Wen, Yuanchun Li, Guohong Liu, Shanhui Zhao, Tao Yu, Toby Jia-Jun Li, Shiqi Jiang, Yunhao Liu, Yaqin Zhang, and Yunxin Liu. 2024. Autodroid: Llm-powered task automation in android. (2024), 543–557.

[31] Jason Wu, Rebecca Krosnick, Eldon Schoop, Amanda Swearngin, Jeffrey P Bigham, and Jeffrey Nichols. 2023. Never-ending Learning of User Interfaces. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*. 1–13.

[32] Jason Wu, Siyan Wang, Siman Shen, Yi-Hao Peng, Jeffrey Nichols, and Jeffrey P Bigham. 2023. WebUI: A Dataset for Enhancing Visual UI Understanding with Web Semantics. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*. 1–14.

[33] Mulong Xie, Sidong Feng, Zhenchang Xing, Jieshan Chen, and Chunyang Chen. 2020. UIED: a hybrid tool for GUI element detection. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 1655–1659.

[34] Mingzhe Xing, Rongkai Zhang, Hui Xue, Qi Chen, Fan Yang, and Zhen Xiao. 2024. Understanding the Weakness of Large Language Model Agents within a Complex Android Environment. *arXiv preprint arXiv:2402.06596* (2024).

[35] An Yan, Zhengyuan Yang, Wanrong Zhu, Kevin Lin, Linjie Li, Jianfeng Wang, Jianwei Yang, Yiwu Zhong, Julian McAuley, Jianfeng Gao, et al. 2023. GPT-4V in Wonderland: Large Multimodal Models for Zero-Shot Smartphone GUI Navigation. *arXiv preprint arXiv:2311.07562* (2023).

[36] Zhao Yang, Jiaxuan Liu, Yucheng Han, Xin Chen, Zebiao Huang, Bin Fu, and Gang Yu. 2023. AppAgent: Multimodal Agents as Smartphone Users. *arXiv preprint arXiv:2312.13771* (2023).

[37] Zhuosheng Zhan and Aston Zhang. 2023. You only look at screens: Multimodal chain-of-action agents. *arXiv preprint arXiv:2309.11436* (2023).

[38] Danyang Zhang, Lu Chen, and Kai Yu. 2023. Mobile-env: A universal platform for training and evaluation of mobile interaction. *arXiv preprint arXiv:2305.08144* (2023).

[39] Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Yonatan Bisk, Daniel Fried, Uri Alon, et al. 2023. Webarena: A realistic web environment for building autonomous agents. *arXiv preprint arXiv:2307.13854* (2023).

A APPENDIX

A.1 APIs Provided by AgentEnv

LlamaTouch provides three categories of APIs for mobile agent integration and UI automation task execution. APIs, parameters, and their return values are listed in Table 11.

Table 11: APIs provided by LlamaTouch to mobile agents for UI automation task execution on real mobile devices.

API Category	API	Parameter	Return Value
Metadata query	get_task_instruction	None	str: A task description in natural language.
UI state query	get_screenshot	None	str: A base64 encoded string representing the current screenshot.
	get_view_hierarchy	None	str: A string representing the textual view hierarchy in XML.
Action space	post_task_complete	None	None
	post_task_impossible	None	None
	post_press_home	None	None
	post_press_back	None	None
	post_click	x, y: Normalized x/y coordinates for a targeted screen position.	None
	post_type	str: The text to be input.	None
	post_swipe	touch_x, touch_y: Normalized x/y coordinates where the swipe begins. lift_x, lift_y: Normalized x and y coordinates where the swipe ends. duration: The interval of the swipe gesture.	None