AutoEval: A Practical Framework for Autonomous Evaluation of Mobile Agents

Jiahui Sun, Zhichao Hua*
Shanghai Jiao Tong University, Shanghai, China
{jason_2001, zchua}@sjtu.edu.cn

***Abstract*—Accurate and systematic evaluation of mobile agents can significantly advance their development and real-world applicability. However, existing benchmarks for mobile agents lack practicality and scalability due to the extensive manual effort required to define task reward signals and implement corresponding evaluation codes. To this end, we propose AutoEval, an autonomous agent evaluation framework that tests a mobile agent without any manual effort. First, we design a Structured Substate Representation to describe the UI state changes during agent execution, such that task reward signals can be automatically generated. Second, we utilize a Judge System that can autonomously evaluate agents' performance given the automatically generated task reward signals. By providing only a task description, our framework evaluates agents with fine-grained performance feedback to that task without any extra manual effort. We implement a prototype of our framework and validate the automatically generated task reward signals, finding over 93% coverage to human-annotated reward signals. Moreover, to prove the effectiveness of our autonomous Judge System, we manually verify its judge results and demonstrate that it achieves 94% accuracy. Finally, we evaluate the state-of-the-art mobile agents using our framework, providing detailed insights into their performance characteristics and limitations.**

***Index Terms*—Automatic Evaluation, Benchmarking, Mobile Agent**

I. Introduction

The advancement of Large Language Models (LLMs) has empowered mobile agents to understand and interact effectively with mobile platforms like Android, potentially freeing humans from tedious daily tasks [1]–[7]. However, these agents still struggle to perform reliably even on simple tasks like "setting a 9AM alarm labeled meeting". Therefore, benchmarking mobile agent performance is critical. An effective benchmark with fine-grained feedback reveals the agent's bottlenecks and guides targeted improvements.

However, existing mobile agent benchmarks are not practical enough to evaluate. To provide fine-grained agent performance measurement beyond binary feedback (i.e. success or failure), most benchmarks evaluate task completion through a series of milestone states. For example, some benchmarks [8]–[13] compare agent trajectories with human demonstrations, providing step-wise performance evaluation. Nevertheless, these benchmarks face the problem of misleading because agents can take multiple paths to solve tasks while human's demonstration cover only limited ways. For this reason, recent benchmarks [14], [15] choose to evaluate agents leveraging reward signals like expected UI state or operating system state. However, such benchmarks are impractical for two reasons. First, they still require significant manual effort to define task-specific reward signals. Second, checking these reward signals requires extensive code development for pattern matching or operating system state checking, introducing high engineering overhead that scales poorly with the number of tasks. For example, evaluating 138 tasks in AndroidLab [14] requires an additional 3,000 lines of evaluation code, highlighting the substantial development overhead.

To address the manual effort issue, previous work Agent-Eval-Refine [16] designs a LLM-based evaluator for agent evaluation. However, it only provides a binary answer of whether the agent completes the task or not, which is too coarse-grained for detailed diagnosis. Our work builds on top of these advancements, seeking a fine-grained autonomous and reliable evaluation of mobile agents.

In this paper, we propose AutoEval, a novel autonomous evaluation framework for mobile agents. The core idea and comparison with existing benchmarks is illustrated in Figure 1. The key technical challenge in our work lies in how to design a LLM-based pipeline to generate task reward signals and evaluate agent performance autonomously while keeping accuracy simultaneously. To tackle this challenge, we design a Structured Substate Representation model to describe task reward signal and implement the State Decomposer which can automatically generate substates accordingly. Moreover, we develop a Judge System that autonomously evaluates agent performance through a three-stage pipeline: (1) capturing structured textual description from agent execution screenshots (2) evaluating agent performance by reasoning about screenshot descriptions against generated substates (3) checking evaluation results through consistency constraints to ensure the reliability of the evaluation results. Based on the above design, AutoEval can evaluate mobile agents without any manual effort.

We collect tasks from existing mobile agent benchmarks and augment them with automatically generated substates. Comparing these generated substates with human-annotated ones, we find that they cover 93.28% of human-annotated substates without additional knowledge. We then evaluate our Judge System's reliability, achieving 94.35% accuracy compared with oracle human evaluation. Finally, we conduct comprehensive evaluation of state-of-the-art mobile agents using our benchmark, providing a detailed analysis of their performance characteristics and failure patterns. We will make

*Corresponding author: Zhichao Hua (zchua@sjtu.edu.cn).

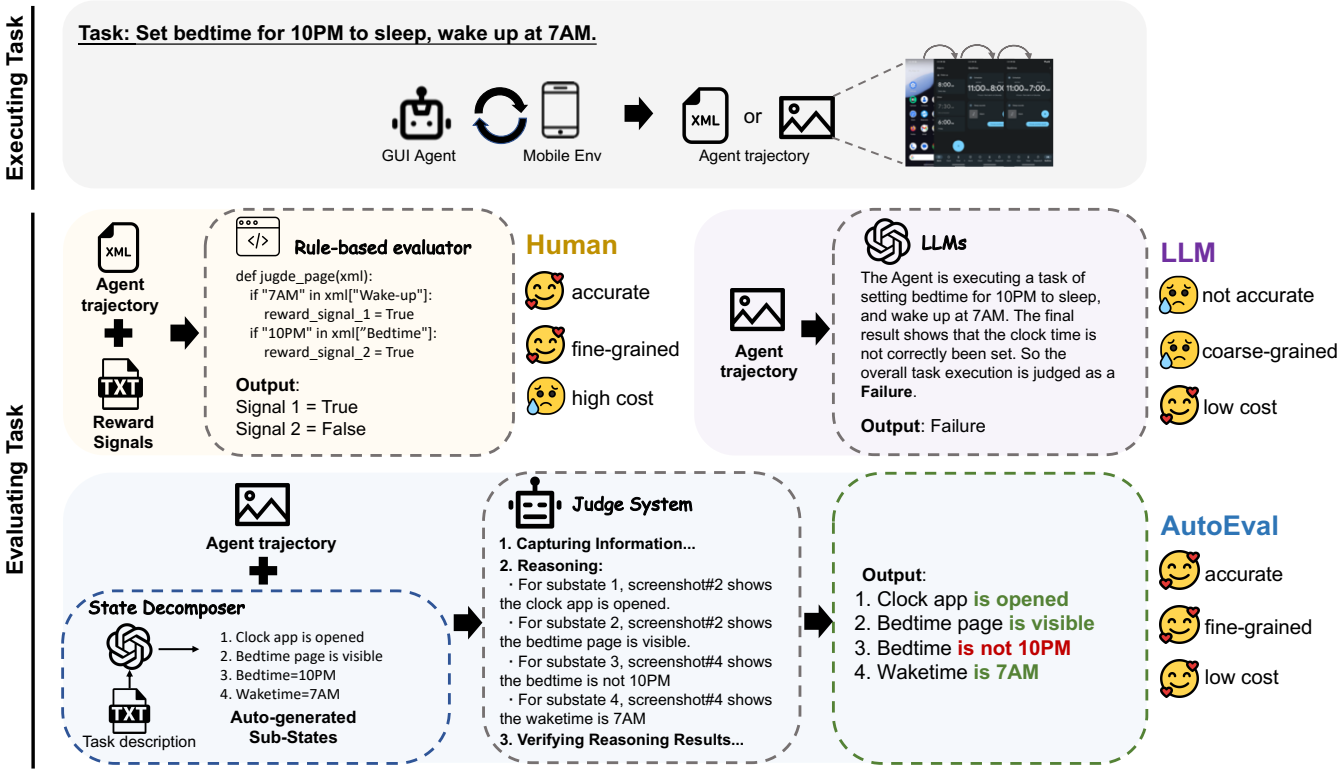

Fig. 1. Comparison of **AutoEval** (ours) with existing agent benchmarks. **Human**-based-benchmark [14], [15]: requires manual reward signal definition and extensive code development for agent evaluation, leading to high accuracy but high cost. **LLM**-based-benchmark [16]: inputs the agent's execution trajectory and utilizes LLM to generate a binary answer of whether the agent completes the task or not. It reduces the costs significantly but sacrifices the accuracy and fine-grained agent performance feedback. **AutoEval**: introduces automatically generated substates along with an autonomous Judge System to provide fine-grained feedback on agent's performance, achieving both high accuracy and low cost.

our data and code publicly available upon acceptance.

Our Contributions can be summarized as follows:

- We propose Structured Substate Representation with State Decomposer for autonomous substate generation, and create a benchmark with 93 tasks across 9 applications.
- We design an autonomous Judge System that replaces human-written evaluation code and provides fine-grained and reliable performance feedback.
- We implement a prototype system based on the proposed design and evaluate its effectiveness.
- We test existing mobile agents in our framework, comparing and analyzing their performance in a fine-grained manner.

II. System Design

A. Overview

We propose AutoEval, an autonomous evaluation framework for mobile agents that aims to fulfill two design goals (DG).

DG1. Autonomous: Minimize human effort in evaluating mobile agents. AutoEval seeks to automate the entire evaluation process by eliminating manual definition of task reward signals and paired evaluation code.

DG2. Reliable: Comparable evaluation accuracy to human evaluation.

To achieve DG1, AutoEval uses a State Decomposer that autonomously generates a specific task's substates. Then, Judge System (Section II-D) autonomously evaluates the agent's performance by checking substate completion given screenshots-trajectory during the agent's task execution.

To achieve DG2, we design Structured Substate Representation (Section II-B) that captures UI states as reward signals during task execution. We guide the Judge System to evaluate each substate using a reasoning-then-checking approach.

The architecture of AutoEval is illustrated in Figure 1. To evaluate how an agent performs on a given task, our framework operates in two phases: Offline Phase and Online Phase.

- *Offline Phase*: The State Decomposer (II-C) automatically generates substates in Structured Substate Representation (II-B).
- *Online Phase*: AutoEval launches the agent with a specific task and captures its screenshots-trajectory. Then, the Judge System (II-D) is responsible for autonomously evaluating the agent's performance by iterating over the screenshots-trajectory and checking the completion of each substate.

B. Structured Substate Representation Model

In this section, we introduce the **S**tructured **S**ubstate **R**epresentation Model (SSR) which describes the UI state during the agent's task execution.

Substates are defined as some inescapable UI states like "Firefox's main page is visible", indicating the correctness of agent task execution, serving the same purpose as reward signals [15]. Substates can be further represented as StateNodes; each StateNode contains a natural language description of a specific UI state and a pointer to its parent StateNode. Based on the observation that the agent's interaction with mobile apps primarily consists of navigating between pages and performing operations (e.g., typing and clicking) within each page, StateNodes are then categorized into two types: PageNodes and UnitNodes.

- **PageNode**: represents page state (e.g., "The app's search page is visible"). PageNode's parent MUST be another PageNode, which represents the previous page from which the current page was entered.
- **UnitNode**: represents unit state (e.g., "The search input field contains 'Large Language Model'"). UnitNode's parent MUST be a PageNode, which represents the page containing this unit.

C. State Decomposer

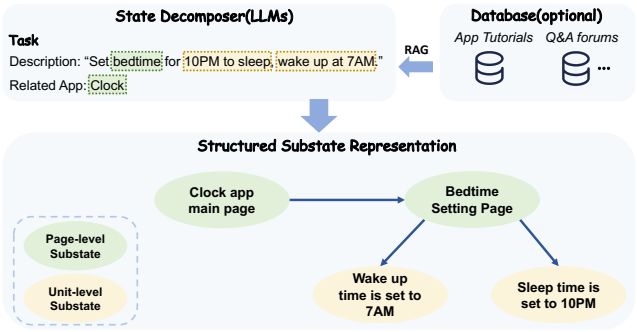

Fig. 2. The process of automatically generating task-specific Structured Substate Representation with State Decomposer.

Figure 2 illustrates the process of generating substates with State Decomposer. It incorporates a pre-defined prompt that describes the SSR model and guides LLM to output substates corresponding to a specific task. These substates are structured as a tree, where the app's home screen typically serves as the root PageNode.

Our key insight is leveraging LLM's knowledge of common apps gained from pre/post-training to generate substates for specific tasks. For a new app that LLM has never been trained on, it can still generate roughly accurate substates through transfer of knowledge from past experiences with similar apps, supplemented by app-specific RAG when needed.

D. Judge System

The architecture of Judge System is shown in Figure 3. Given the substates of a task and the screenshots-trajectory during the agent's task execution, Judge System autonomously evaluates the agent's functionality correctness with substate-level feedback. It has three key components: Capturer, Reasoner and Checker. The overall judging process is summarized in Algorithm 1. We provide a detailed description of each component in the following.

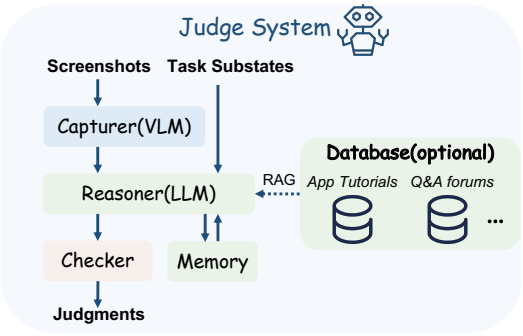

Fig. 3. The architecture of Judge System.

Algorithm 1 Judging Process

Require: Task description t, initial substates $\mathcal{S}_t$, screenshots trajectory $\mathcal{SC}_t$
Ensure: Final evaluated substates $\mathcal{S}_t$

```
Initialize memory M ← ∅
for each screenshot sc_i ∈ SC_t do
    d_i ← CAPTURER(sc_i) {Convert screenshot to text}
    r_i, info ← REASONER(d_i, S_t, M)
    M ← M ∪ {info} {Update memory}
    S_t ← CHECKER(r_i)
end for
return S_t
```

Capturer. The Capturer is built upon a Vision Language Model (VLM) that converts screenshots into detailed textual descriptions of layout, content, and app identification.

Reasoner. The Reasoner is based on a Large Language Model (LLM) and plays a crucial role in judging an agent's performance. Given the description of the ith screenshot d_i, task t, task substates representation $\mathcal{S}_t$, Reasoner generates an analysis a_k and a judge result j_k for each substate $s_k \in \mathcal{S}_t$. The judge result j_k for substate s_k is either Success if d_i matches s_k or Uncertain otherwise.

Integrated with the structured substates representation, we can prompt the Reasoner to reason about each of the substates following Algorithm 2. We enhance Reasoner with a memory module storing critical information the Reasoner proactively reported and successful substates, and optionally use Retrieval-Augmented Generation (RAG) to incorporate app-specific knowledge for more accurate results.

Checker. During our empirical evaluation, we find that the Reasoner occasionally generates unexpected outputs, failing to strictly follow the reasoning process in Algorithm 2. Consequently, we incorporate a Checker module to guarantee the Reasoner's output is acceptable and consistent with the

Algorithm 2 Reasoning Process

Require: Screenshot description d_i, Substates $\mathcal{S}_t$ (sorted by hierarchy), Memory $\mathcal{M}$
Ensure: Judgment results $\mathcal{J}$

```
Initialize judgments J ← ∅
for each substate s_k ∈ S_t do
    info, a_k, j_k ← MATCH(d_i, s_k, M)
    if s_k is a UnitNode then
        Let p be the parent of s_k
        if J[p] ≠ Success then
            j_k ← Failure {Parent constraint}
        end if
    end if
    Append j_k to J
end for
return J, info
```

reasoning process. There are two rules that Checker guarantees: 1) Each substate judge result must be either Success or Uncertain. 2) UnitNodes can only be judged as Success if their parent PageNode is also Success. If violated, Checker retries or skips the current judgment.

III. Evaluation

We evaluate AutoEval on a variety of popular Large Language Models (LLMs) configurations and mobile agents. Our evaluation seeks to answer the following questions:

Q1. What is the accuracy and completeness of the substates generated by State Decomposer (Section II-C) compared to human-annotated substates?

Q2. What is the accuracy of the Judge System (Section II-D)?

Q3. Can our framework effectively demonstrate the fine-grained and comprehensive performance of mobile agents?

Models and Environments. We evaluate AutoEval across multiple LLM configurations, including GPT-4o [17], DeepSeek V3 [18], Gemini-2.0-flash-thinking [19]. The Capturer in Judge System utilizes Gemini-2.0-flash as its backend VLM. We conduct our experiments on an Android Virtual Device (AVD) emulator as the mobile environment for agent execution.

Mobile Agent. We evaluate a prompt-based agent and a training-based agent using our framework. Each agent starts performing tasks in an identical manually initialized environment.

- Mobile-Agent-E [7]: Hierarchical multi-agent framework. We select Qwen-VL-Plus [20] as its caption model following the original paper and use gemini-2.0-flash-thinking as its reasoning model.
- CogAgent [6]: Training-based agent. In our evaluation, we test on the latest version of the CogAgent-9B-20241220.

A. State Decomposer Evaluation

To answer Q1, we collect 93 tasks from the following benchmark and augment these tasks with substates using State Decomposer. We then manually evaluate the quality of substates generated by State Decomposer.

AndroidLab [14]: An open-source Android agent benchmark emphasizing generalizability. We evaluate all Operation Tasks across 9 applications.

Metrics. To evaluate the quality of automatically generated task substates ($\mathcal{S}_{\text{Auto}}$), we compare them with manually defined substates ($\mathcal{S}_{\text{Human}}$) using Cover Rate, Redundant Rate, Optional Rate, and Incorrect Rate metrics.

- Cover Rate: measures how many substates in $\mathcal{S}_{\text{Human}}$ are covered by $\mathcal{S}_{\text{Auto}}$.
- Redundant Rate: indicates the proportion of duplicate and redundant substates in $\mathcal{S}_{\text{Auto}}$.
- Optional Rate: agents can take multiple paths to achieve the same task goal, resulting in some substates being path-dependent. We define these path-dependent substates in $\mathcal{S}_{\text{Auto}}$ as optional substates.
- Incorrect Rate: represents the percentage of invalid substates in $\mathcal{S}_{\text{Auto}}$ due to hallucinated content.

Results. Table I compares State Decomposer performance across different LLM configurations. Both GPT-4o and DeepSeek V3 achieve a high Cover Rate with low Incorrect Rate, indicating strong capability in identifying essential task substates. Experimental results confirm the 10.49% redundancy doesn't impact accuracy. Since redundant substates are semantic duplicates of the same UI state, they yield identical judgments without introducing any additional false positives or negatives. Both models exhibit similar Optional Rate, likely because their knowledge is constrained to substates on all seen task completion paths, limiting their ability to identify some substates that could be non-existent in other unseen but valid task completion paths.

B. Judge System Performance Evaluation

To answer Q2, we collect 93 screenshots-trajectory traces from both human participants and Mobile-Agent-E during task execution. These traces are then autonomously judged by the Judge System with task-specific substates generated in Section III-A. We evaluate Judge System reliability through human verification, comparing its substate-level judgments against the actual screenshots-trajectory traces using three metrics: Judge Success Rate (SR), False Positive Rate (FP), and False Negative Rate (FN).

Results. Table II shows the accuracy of Judge System in judging both human and agent traces, while Reasoner is configured with different models. Contributing to our design of the Judge System and Structured Substates Representation, we find that Judge System achieves high accuracy that aligns with human judgments. The Gemini-2.0-flash-thinking based Judge System achieves the highest accuracy (94.35%) among all LLM configurations, demonstrating the strong capability of reasoning models.

Error Analysis. We further analyze error cases in autonomous judgments and identify two main causes:

1) *Limited Capturer model capabilities* (30.8% of errors): Though Capturer performs well in most cases, it still faces the problem of information loss when converting

TABLE I

COMPARISON OF STATE DECOMPOSER PERFORMANCE ACROSS DIFFERENT APPLICATIONS. WE COMPARE GPT-4O AND DEEPSEEK V3 ON FOUR KEY METRICS. THE BEST RESULTS FOR EACH APPLICATION ARE HIGHLIGHTED IN **BOLD**. ARROWS INDICATE WHETHER LOWER (↓) OR HIGHER (↑) VALUES ARE BETTER.

Application	↑ Cover Rate (%)		↓ Redundant Rate (%)		↓ Optional Rate (%)		↓ Incorrect Rate (%)	
	GPT-4o	DeepSeek	GPT-4o	DeepSeek	GPT-4o	DeepSeek	GPT-4o	DeepSeek
Bluecoins	97.83	**100.00**	**14.71**	26.03	16.18	**10.96**	2.94	**0.00**
Calendar	94.23	**98.08**	**11.76**	33.77	14.71	**0.00**	1.47	**0.00**
Cantook	87.88	**93.94**	**11.43**	15.00	**5.71**	7.50	**0.00**	**0.00**
Clock	**92.47**	**92.47**	**6.93**	16.82	1.98	**0.00**	2.97	**2.80**
Contacts	**98.04**	92.16	**6.90**	13.51	**6.90**	17.57	**0.00**	5.41
Map	73.08	**88.46**	**17.65**	18.92	23.53	**18.92**	2.94	**0.00**
PiMusic	**100.00**	83.33	**10.34**	25.71	**6.90**	17.14	**0.00**	**0.00**
Setting	**98.04**	96.08	**10.94**	13.64	10.94	**9.09**	**0.00**	3.03
Zoom	**95.00**	**95.00**	**5.00**	**5.00**	**0.00**	**0.00**	**0.00**	**0.00**
Average	93.28	**93.94**	**10.49**	19.85	9.82	**8.13**	**1.56**	1.70

TABLE II

JUDGE SYSTEM RELIABILITY ACROSS DIFFERENT REASONER BASE MODELS ON HUMAN AND AGENT TRACES. (DEEPSEEK: DEEPSEEK-V3; GEMINI: GEMINI-2.0-FLASH-THINKING)

Model	Human Trace			Agent Trace		
	↑ SR (%)	↓ FP (%)	↓ FN (%)	↑ SR (%)	↓ FP (%)	↓ FN (%)
GPT-4o	87.76	1.90	10.34	87.70	2.05	10.25
DeepSeek	82.91	3.59	13.50	90.33	**1.61**	8.06
Gemini	**94.31**	**0.42**	**5.27**	**94.35**	2.02	**3.63**

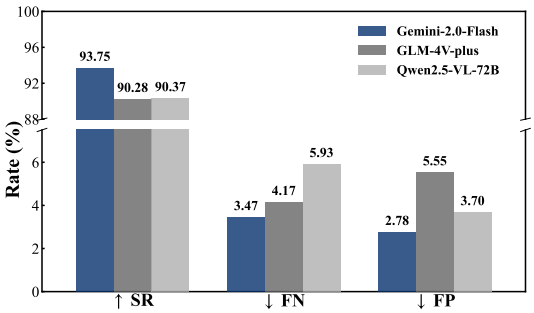

Fig. 4. Comparison of Judge System accuracy with different Vision-Language Models.

screenshots into textual descriptions. Missing important information leads to reasoning errors. For example, in the case of judging the substate "Sort button is visible", the Capturer may fail to identify the Sort button in its output even though the button is actually present in the screenshot, which leads to false negative judgments.

2) *Limited Reasoner model capabilities* (69.2% of errors): The Judge System can make incorrect judgments even given all necessary information due to the limited capabilities of reasoning and instruction following.

VLM Backend Ablation. To justify our choice of Gemini-2.0-Flash as the Capturer backend, we evaluate the Judge System accuracy with different Vision-Language Models on 26 tasks from AndroidLab. Figure 4 shows the comparison results. Gemini-2.0-Flash achieves the highest accuracy (93.75%), outperforming Qwen2.5-VL-72B and GLM-4V-plus. The superior performance of Gemini-2.0-Flash in screenshot understanding enables more accurate textual descriptions, reducing information loss during the captioning process.

Cost Analysis. Table III presents the per-round API costs for the two LM-based components in the Judge System. Checker overhead is negligible compared to Reasoner and Capturer. On average, the Judge System requires 8 rounds to complete the evaluation of a single agent task execution, taking about 132.7s (Capturer: 52.4s, Reasoner: 80.3s). The total cost for evaluating one task execution amounts to approximately 0.0224 USD. Using multiple test phones/emulators can reduce evaluation time, and lower-cost models can further reduce evaluation costs.

TABLE III

TOKEN CONSUMPTION AND COST ANALYSIS FOR JUDGE SYSTEM COMPONENTS. REASONER USES GEMINI-2.0-FLASH-THINKING WHILE CAPTURER USES GEMINI-2.0-FLASH AS BACKEND VLM. ON AVERAGE, 8 ROUNDS ARE REQUIRED PER TASK EXECUTION.

Component	Input		Output		Total ($)
	Tokens	Cost ($)	Tokens	Cost ($)	
Reasoner	2,668	0.0011	750	0.0003	0.0014
Capturer	2,981	0.0012	466	0.0002	0.0014
Per Round	**5,649**	**0.0023**	**1,216**	**0.0005**	**0.0028**
Per Task (8x)	**45,192**	**0.0184**	**9,728**	**0.0040**	**0.0224**

Generalizability Analysis. To further validate the generalizability of the Judge System beyond AndroidLab, we randomly sample 53 additional tasks from AndroidArena [10] and AitW [9], covering 8 new application categories not present in our primary evaluation. Human annotators manually execute these tasks and record their interaction trajectories, which are then submitted to AutoEval for autonomous evaluation. The Judge System achieves 91.38% accuracy on these unseen tasks, demonstrating strong cross-benchmark generalization.

C. Mobile Agent Performance Evaluation

To answer Q3, we evaluate the absolute capability of different mobile agents using our framework on tasks with substates generated in Section III-A. We choose Gemini-2.0-flash-thinking as a backend model for Reasoner in Judge System.

Metrics. AutoEval provides substate-level evaluation for each task execution, enabling us to evaluate mobile agent performance using Substate Completion Rate (SCR) and Task Completion Rate (TCR).

- Substate Completion Rate (SCR): The average percentage of successfully completed substates across all tasks.
- Task Completion Rate (TCR): The percentage of tasks where all substates are successfully completed.

TABLE IV
PERFORMANCE OF DIFFERENT MOBILE AGENTS EVALUATED BY AUTOEVAL.

Agent	SCR (%)	TCR (%)
CogAgent	62.59	22.45
Mobile-Agent-E	77.84	32.65

Results. Table IV shows the performance of different mobile agents. We find a notable gap between SCR and TCR for both agents, indicating that our evaluation framework can give a more realistic and fine-grained evaluation of agent performance. The substate completion feedback from AutoEval reveals mobile agent weaknesses, including insufficient app-specific knowledge and action space limitations (e.g., inability to handle certain UI interactions or gestures).

IV. CONCLUSION

In this study, we introduced AutoEval, a practical evaluation framework that enables autonomous fine-grained evaluation of mobile agents. By proposing a Structured Substate Representation model, AutoEval can automatically generate reward signals for a given task. Moreover, AutoEval designs a three-stage Judge System to autonomously evaluate the performance of mobile agents. Our results show that AutoEval produces reward signals aligned with human annotations and reaches human-level autonomous evaluation accuracy.

V. ACKNOWLEDGEMENTS

We thank the reviewers for their insightful comments. This work is supported in part by China National Natural Science Foundation (No.62202289) and a research grant from Huawei Technologies.

REFERENCES

[1] Chi Zhang, Zhao Yang, Jiaxuan Liu, Yanda Li, Yucheng Han, Xin Chen, Zebiao Huang, Bin Fu, and Gang Yu, "Appagent: Multimodal agents as smartphone users," in *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems*, 2025, pp. 1–20.

[2] Zhuosheng Zhang and Aston Zhang, "You only look at screens: Multimodal chain-of-action agents," in *Findings of the Association for Computational Linguistics: ACL 2024*, 2024, pp. 3132–3149.

[3] Songqin Nong, Jiali Zhu, Rui Wu, Jiongchao Jin, Shuo Shan, Xiutian Huang, and Wenhao Xu, "Mobileflow: A multimodal llm for mobile gui agent," *arXiv preprint arXiv:2407.04346*, 2024.

[4] Sunjae Lee, Junyoung Choi, Jungjae Lee, Munim Hasan Wasi, Hojun Choi, Steven Y Ko, Sangeun Oh, and Insik Shin, "Explore, select, derive, and recall: Augmenting llm with human-like memory for mobile task automation," *arXiv preprint arXiv:2312.03003*, 2023.

[5] Hao Wen, Yuanchun Li, Guohong Liu, Shanhui Zhao, Tao Yu, Toby Jia-Jun Li, Shiqi Jiang, Yunhao Liu, Yaqin Zhang, and Yunxin Liu, "Autodroid: Llm-powered task automation in android," in *Proceedings of the 30th Annual International Conference on Mobile Computing and Networking*, 2024, pp. 543–557.

[6] Wenyi Hong, Weihan Wang, Qingsong Lv, Jiazheng Xu, Wenmeng Yu, Junhui Ji, Yan Wang, Zihan Wang, Yuxiao Dong, Ming Ding, et al., "Cogagent: A visual language model for gui agents," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024, pp. 14281–14290.

[7] Zhenhailong Wang, Haiyang Xu, Junyang Wang, Xi Zhang, Ming Yan, Ji Zhang, Fei Huang, and Heng Ji, "Mobile-agent-e: Self-evolving mobile assistant for complex tasks," *arXiv preprint arXiv:2501.11733*, 2025.

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

[9] Christopher Rawles, Alice Li, Daniel Rodriguez, Oriana Riva, and Timothy Lillicrap, "Androidinthewild: A large-scale dataset for android device control," *Advances in Neural Information Processing Systems*, vol. 36, pp. 59708–59728, 2023.

[10] Mingzhe Xing, Rongkai Zhang, Hui Xue, Qi Chen, Fan Yang, and Zhen Xiao, "Understanding the weakness of large language model agents within a complex android environment," in *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2024, pp. 6061–6072.

[11] Li Zhang, Shihe Wang, Xianqing Jia, Zhihan Zheng, Yunhe Yan, Longxi Gao, Yuanchun Li, and Mengwei Xu, "Llamatouch: A faithful and scalable testbed for mobile ui task automation," in *Proceedings of the 37th Annual ACM Symposium on User Interface Software and Technology*, 2024, pp. 1–13.

[12] Sagar Gubbi Venkatesh, Partha Talukdar, and Srini Narayanan, "Ugif: Ui grounded instruction following," *arXiv preprint arXiv:2211.07615*, 2022.

[13] Martin Klissarov, Pierluca D'Oro, Shagun Sodhani, Roberta Raileanu, Pierre-Luc Bacon, Pascal Vincent, Amy Zhang, and Mikael Henaff, "Motif: Intrinsic motivation from artificial intelligence feedback," *arXiv preprint arXiv:2310.00166*, 2023.

[14] Yifan Xu, Xiao Liu, Xueqiao Sun, Siyi Cheng, Hao Yu, Hanyu Lai, Shudan Zhang, Dan Zhang, Jie Tang, and Yuxiao Dong, "Androidlab: Training and systematic benchmarking of android autonomous agents," in *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2025, pp. 2144–2166.

[15] Christopher Rawles, Sarah Clinckemaillie, Yifan Chang, Jonathan Waltz, Gabrielle Lau, Marybeth Fair, Alice Li, William Bishop, Wei Li, Folawiyo Campbell-Ajala, et al., "Androidworld: A dynamic benchmarking environment for autonomous agents," *arXiv preprint arXiv:2405.14573*, 2024.

[16] Jiayi Pan, Yichi Zhang, Nicholas Tomlin, Yifei Zhou, Sergey Levine, and Alane Suhr, "Autonomous evaluation and refinement of digital agents," *arXiv preprint arXiv:2404.06474*, 2024.

[17] Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, et al., "Gpt-4o system card," *arXiv preprint arXiv:2410.21276*, 2024.

[18] Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al., "Deepseek-v3 technical report," *arXiv preprint arXiv:2412.19437*, 2024.

[19] Gemini Team, Rohan Anil, Sebastian Borgeaud, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, Katie Millican, et al., "Gemini: a family of highly capable multimodal models," *arXiv preprint arXiv:2312.11805*, 2023.

[20] Jinze Bai, Shuai Bai, Shusheng Yang, Shijie Wang, Sinan Tan, Peng Wang, Junyang Lin, Chang Zhou, and Jingren Zhou, "Qwen-vl: A versatile vision-language model for understanding, localization, text reading, and beyond," *arXiv preprint arXiv:2308.12966*, 2023.