GUITAR: Unmasking the Competence Illusion in GUI Agent Evaluation via State-Transition Representation

Anonymous ACL submission

Abstract

Recent advancements in Vision-Language Models have spurred interest in Graphical User Interface (GUI) agents. However, current evaluation relying on step accuracy rate harbors a critical blind spot: it conflates frequency with competence. In long-tailed GUI tasks, repetitive actions and screens dominate trajectories, inducing a Competence Illusion: a frequency-driven evaluation bias in which performance on high-frequency screens disproportionately shapes the measured score. Decoupling performance from frequency is essential, but presents a challenge as raw screens lack structural frequency-agnostic abstract or aggregation. To bridge this gap, we propose a state-centric schema and introduce GUITAR, a diagnostic framework that represents execution trajectories as State Transition Graphs and enables multi-granular diagnosis of failures. Through extensive experiments, we show that execution failures are highly concentrated on a small number of states and transitions, including rare but critical transitions that remain invisible under conventional evaluation. Our findings highlight fundamental limitations of frequency-weighted metrics and demonstrate the necessity of state- and transition-level diagnostics for faithful evaluation of GUI agents.[1]

1 Introduction

Recent advancements in Vision-Language Models (VLMs) have spurred growing interest in Graphical User Interface (GUI) agents that autonomously operate complex applications (Achiam et al., 2023; Anil et al., 2023; Hong et al., 2024; Yang et al., 2023). As agent capabilities improve, reliable evaluation becomes increasingly important, serving not only to quantify progress but also to provide a granular understanding of failure modes.

Current trajectory evaluation methods primarily focus on step accuracy rates (SR) (Zhang and Zhang, 2024; Rawles et al., 2024; Zheng et al., 2025). However, these metrics harbor a critical blind spot: they conflate frequency with competence. In real-world GUI tasks, agents encounter a long-tail distribution of screens. A substantial portion of a trajectory consists of repetitive screens and actions, with a small number of screens being visited frequently. In contrast, many screens are encountered only rarely, yet they are often decision-critical and more prone to execution errors. When performance is aggregated at the step level, this skew introduces a ***Competence Illusion***: a frequency-induced evaluation bias in which step-level metrics overweight high-frequency screenshots and underweight rare but critical interactions, thereby distorting the measured competence of an agent. As a consequence, step-level evaluation suffers from two fundamental limitations: (1) it fails to capture failures in low-frequency but pivotal screenshots, and (2) it cannot localize performance weaknesses to specific failure-prone scenarios. Figure 1 shows how SR fails to identify bottlenecks.

However, correcting this bias poses a significant challenge: **raw GUI trajectories lack intrinsic categorical labels (e.g., state identity).** Minor pixel-level variations often make each screen appear as a new state. To decouple visitation frequency, visually and functionally similar screens must map to unique states. Without such state abstraction, measuring frequency-independent competence is mathematically ill-defined.

To bridge this gap, we propose a state-centric schema and introduce **GUITAR** (GUI Transition Analysis and Representation) a diagnostic framework. Instead of treating evaluation at the step-level, GUITAR unifies fragmented execution trajectories into a structured State Transition Graph (STG). By employing a semantically-guided abstraction process to map visual screens to unique states, GUITAR provides the necessary foundation to define and measure state-level competence. This

[1] All codes and dataset will be released upon acceptance.

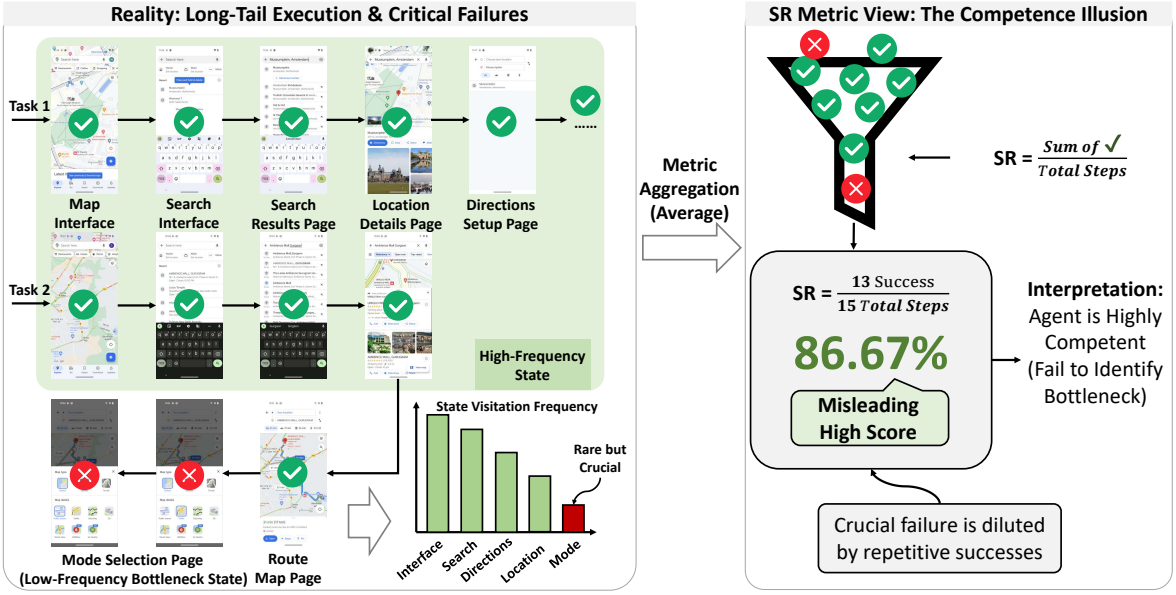

Figure 1: The Competence Illusion of the step-level metric (SR).

transformation allows us to access agents' competence independent of visitation frequency. Leveraging the STG, we conduct a set of hypothesis-driven analyses. First, we demonstrate that agent competence is unevenly distributed across the frequency spectrum, indicating that frequency-weighted metrics are unreliable proxies for true competence. Building on this observation, we find that the step-level metric systematically obscures low-competence bottleneck states, limiting its effectiveness for failure localization. Finally, we investigate failures at the level of state transitions and describe two failure patterns. This reveals diagnostic insights that cannot be inferred from step-level or state-level summaries alone.

The contributions of this work are summarized as follows:

(1) **Conceptual Insight.** We identify a frequency-induced competence illusion in step-level evaluation of GUI agents and show that visitation frequency is not a reliable indicator of execution competence.

(2) **Diagnostic Framework.** We propose GUI-TAR, a state–transition–based framework that enables multi-granular diagnostic analysis of agent behavior beyond step-level metrics.

(3) **Empirical Analysis.** We conduct a hypothesis-driven analysis of diverse GUI agents across multiple applications, revealing common state- and transition-level failure concentration. We demonstrate how transition-level diagnostics uncover latent and systemic failure patterns that are invisible to conventional evaluation.

2 Related Work

2.1 GUI Agents

Recently, there has been considerable exploration in the field of GUI Agents, ranging from box prediction based on HTML and OCR parsing to coordinate prediction based on images (Li and Li, 2022; Li et al., 2024b; Wang et al., 2024; Zhang et al., 2024; Yan et al., 2023; Ma et al., 2024; Gou et al., 2024). Wu et al. (2024) employed significant engineering effort to collect multi-platform data and train a more powerful GUI Agent. Qin et al. (2025) trained UI-TARS on large-scale GUI screenshot data, enabling context-aware understanding of UI widgets. Wang et al. (2025a) enhanced the capabilities of GUI agents through multi-round reinforcement learning. Ye et al. (2025) trained GUI-Owl by proposing a self-evolving trajectory data production pipeline for data collection.

2.2 Evaluation For GUI Agents

Prior work has introduced a variety of datasets for evaluating GUI agents, ranging from static UI understanding (Bai et al., 2021; Deka et al., 2017) to query-driven UI element grounding (Cheng et al., 2024; Rawles et al., 2023; Sun et al., 2022; Li et al., 2025). As planning capabilities gained prominence, recent benchmarks began to emphasize fine-grained execution trajectories (Li et al., 2024a; Zheng et al., 2025; Wang et al., 2025b; Zhou et al., 2024; Xie et al., 2024; Rawles et al., 2024; Lu

et al., 2024). However, most existing evaluations remain grounded in step-level metrics. Closest to our work, Qian et al. (2025) aggregates trajectories using graph structures, but focuses on cross-model variance within individual tasks, where the state space is substantially simpler than full-scale applications. Zhang et al. (2025) proposes a state-level evaluation framework, yet operates on static datasets with predefined states and does not account for the dynamic and non-deterministic state transitions characteristic of GUI interaction.

In contrast, GUITAR addresses the structural competence illusion induced by visitation frequency in step-level evaluation. By constructing application-level STGs via semantically-guided abstraction, GUITAR localizes failure-prone bottleneck states and diagnoses transition-level failure patterns that are invisible to step-level aggregation.

3 Problem Formalization

In this section, we first show how visitation frequency induces systematic bias in SR metric (Section 3.1). Second, we demonstrate that even with perfect state abstraction, state-level aggregation remains fundamentally insufficient, motivating the need for transition-level analysis (Section 3.2).

3.1 Frequency-Induced Evaluation Bias

Current GUI agent evaluations predominantly rely on SR, which aggregates performance across all steps without accounting for the impact of visitation frequency. Under skewed distributions, this leads to a competence illusion: the score is heavily biased toward high-frequency states, thereby masking failures in low-frequency but critical bottleneck states that are essential for true competence.

We formalize a task environment as a finite set of latent states $S = \{s_1, \ldots, s_N\}$. Each state s has an intrinsic execution competence $c(s) \in [0, 1]$, defined as the probability of executing the correct action when encountering s. During evaluation, states are visited according to an induced distribution $p(s)$. Under this formulation, SR estimates:

$$SR = \frac{\text{successful steps}}{\text{total steps}} = \mathbb{E}_{s \sim p(s)}[c(s)], \quad (1)$$

which weights competence by visitation frequency.

However, frequency-independent competence is better captured by the uniform state-level average:

$$C_{\text{uniform}} = \mathbb{E}_{s \sim \text{Uniform}(s)}[c(s)]. \quad (2)$$

When $p(s)$ is highly skewed and negatively correlated with $c(s)$, SR systematically overestimates true state-level competence. Importantly, this bias arises even when agent behavior is fixed; it is purely induced by the evaluation procedure. We refer to this structural mismatch as the ***Competence Illusion***. This frequency-induced bias is a structural property of step-level evaluation and arises even in the absence of environmental complexity or abstraction error. A minimal synthetic demonstration isolating this effect is provided in Appendix B.

Therefore, addressing this evaluation bias necessitates principled state identification. Without principled state abstraction, notions such as low-frequency states or uniform state-level aggregation are ill-defined in raw trajectories. This motivates the need for an explicit state-based representation.

3.2 Non-Equivalence to Scalar Aggregation

A natural question is whether frequency-induced bias can be resolved by any aggregation or reweighting. We show that this is fundamentally insufficient: there exist failures that are indistinguishable under any scalar or state-level aggregation, yet are revealed by transition-level analysis.

Proposition 1 (Non-Equivalence). There exist two GUI agents π_1 and π_2 such that they are indistinguishable under any scalar or state-level aggregation metrics, but exhibit distinct failure bottlenecks in the STG constructed by GUITAR.

Intuition. Even with perfect state abstraction, state-level aggregation collapses transition identity. Failures occurring within an interstate and failures occurring at an intrastate level cannot be distinguished. By explicitly preserving state-transition structure, GUITAR avoids this collapse and enables failure localization that is provably unattainable by aggregation-based evaluation. A formal construction and proof are provided in Appendix C.

4 Method

This section describes the two core processes of GUITAR: (1) constructing the STG from trajectory data, (2) performing multi-level analysis over the resulting graph to characterize state- and transition-level execution behavior. Figure 3 illustrates the schema of GUITAR. We emphasize that GUITAR does not aim to produce a single scalar score for model ranking, nor does it claim to measure absolute task difficulty. Instead, its purpose is to reveal where models fail, rather than how much they fail.

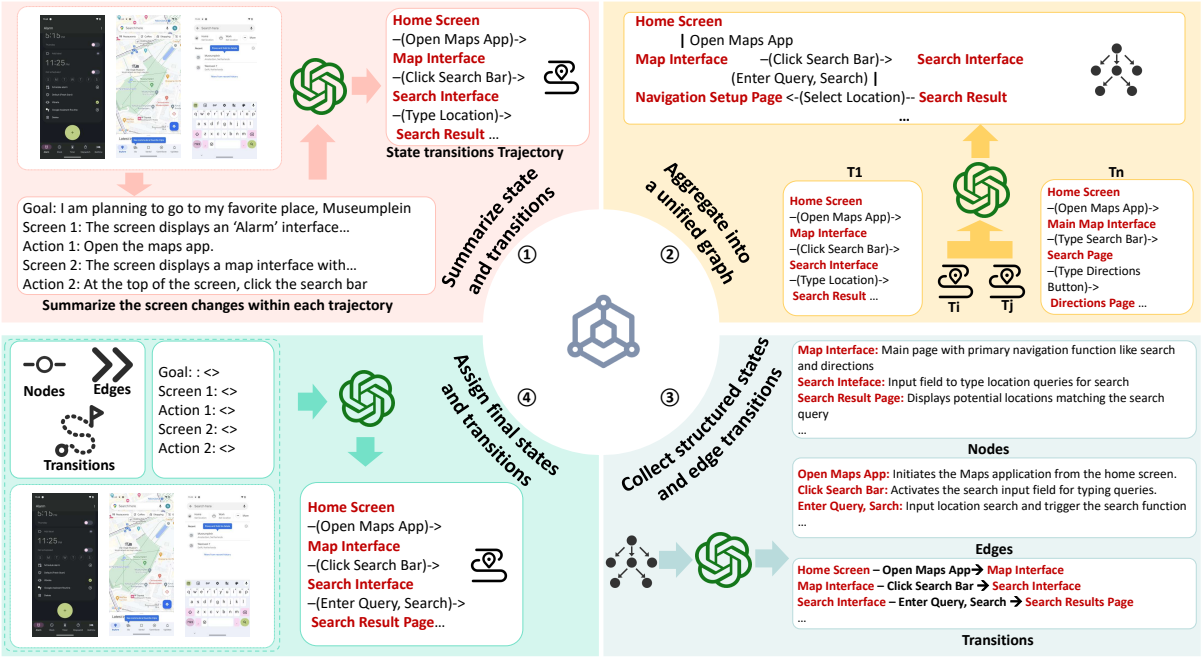

Figure 2: GUITAR automatically executes four stages to construct a unified application STG.

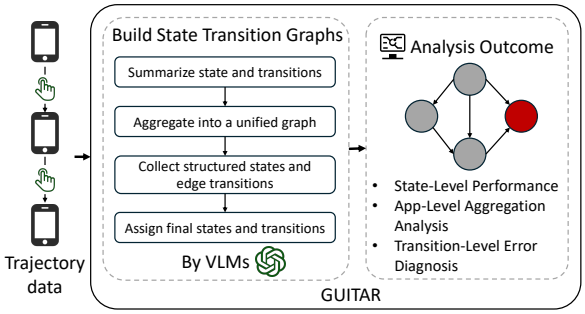

Figure 3: Illustration of the overall schema of GUITAR.

4.1 Building State Transition Graphs

The process consists of four main stages: Summarize states and transitions, Aggregate into a unified graph, Collect structured states and edge transitions, and Assign final states and transitions. Figure 2 illustrates the workflow of GUITAR for the automated construction of STG.

Summarize states and transitions. In the first stage, GUITAR extracts screen changes from each agent trajectory. Given the task goal, screen descriptions, and corresponding actions, VLMs summarize how the interface evolves as the agent executes the task. The outcome is a sequence of abstract states representing the application's dynamic behavior. This process establishes the foundation for identifying state transitions and understanding the agent's interaction flow.

We define a ***state*** as a functional-semantic abstraction of the interface, characterized by the role the current screen plays in the interaction flow, rather than its raw visual appearance. Specifically, a state describes neither the visual appearance of a screen nor the task being executed. Instead, it captures the functional role of the screen within the interaction workflow and the set of actionable affordances available under that context. Under this definition, screens that differ visually but serve the same interface function and afford equivalent interactions are mapped to the same abstract state.

Aggregate into a unified graph. The second stage aggregates and consolidates the state transitions extracted from multiple trajectories, thereby forming a unified state transition graph. By merging individual trajectories, the system identifies overlapping paths and shared transition patterns across different tasks, providing a holistic representation of the application's navigational structure.

Collect structured states and edge transitions. The third stage performs the structural organization of the aggregated graph. It decomposes the overall transition structure into two key components:

- **States**, representing distinct interface states;
- **Edges Transitions**, representing the actions that trigger transitions between states;

This structural reorganization prepares the transition data for the final assignment stage.

Quantity	Aggregation Level	What It Measures	Role
SR	Step (Aggregated)	Frequency-weighted step completion	Coarse outcome
S-Mean	App (Aggregated)	Average state-level accuracy	Overall state-level competence
S-Std	App (Distributional)	Standard deviation of state-level accuracy	Heterogeneity
State-level Accuracy	Individual State	Local competence at a specific state	Bottleneck state detection
Transition-level Error	State Transition	Failure propagation between states	Error attribution

Table 1: Roles of evaluation quantities in GUITAR.

Assign final states and transitions. The final stage refines and assigns standardized representations to all states and edges, thereby generating a unified state transition graph.

4.2 Multi-level Analysis Method

Based on the constructed STG, GUITAR performs complementary analysis at three granularities: State-Level Performance, App-Level Aggregation, Transition Error Diagnosis. Table 1 summarizes how each quantity aligns with the three analysis levels, and clarifies what each signal is (and is not) intended to capture.

State-Level Performance. Traditional step-level evaluation aggregates performance over repeated steps, potentially biasing results toward frequently visited states. To avoid this bias, GUITAR evaluates execution performance at the level of semantic states. For each state in the STG, we compute the state-level accuracy. Specifically, the state-level accuracy is computed as the average step accuracy rate of instructions associated with each state.

App-Level Aggregation Analysis. To summarize state-level performance at the application level, we aggregate the state-level accuracy values across all states in the STG using two statistics:

- **S-Mean.** The average state-level accuracy across all unique states reflects general competence in the abstract state space.
- **S-Std.** The standard deviation of the state-level accuracy captures heterogeneity in state-level performance.

Transition-Level Error Diagnosis. While state-level analysis reveals where execution is unreliable, failures in GUI interaction often arise from specific transitions between states. GUITAR therefore analyzes execution behavior at the transition level. We introduce two transition-level metrics:

- **Transition Frequency (TF).** Measures how frequently each transition is encountered during evaluation, helping identify commonly attempted transitions.
- **Failure Contribution Index (FCI).** Quantifies the proportion of task failures attributable to a specific transition. The formulas for FCI are as follows:

$$FCI_{s_i \to s_j} = \frac{\text{failures on } s_i \to s_j}{\text{all tasks involving } s_i \to s_j} \quad (3)$$

TF captures how commonly a transition is encountered, while FCI quantifies its conditional failure rate. We use these statistics solely for descriptive diagnosis of execution failures.

5 Experiments

This section empirically tests three hypotheses derived from the frequency-induced evaluation bias formalized in Section 3.1.

Hypothesis 1: In real GUI trajectories, execution competence is unevenly distributed across states with different visitation frequencies (Section 5.2).

Hypothesis 2: Step-level metrics fail to expose failure-prone bottleneck states that are critical to task execution (Section 5.3).

Hypothesis 3: Transition-aware analysis enables failure attribution beyond state-level aggregation (Section 5.4).

5.1 Dataset and Setups

For our experiments, we select seven applications comprising a total of 857 examples from AndroidControl (Li et al., 2024a) as the evaluation benchmark. Complete statistics for all seven applications are provided in Appendix D.1. Following the GUITAR framework, we employ GPT-4o to perform semantically guided state abstraction and construct an application-level STG for each application. The

Model	Maps			CNN			Kitchen Stories		
	SR	S-Mean	S-Std	SR	S-Mean	S-Std	SR	S-Mean	S-Std
Aguvis-7B	65.10	58.26↓	35.94	76.62	60.49↓	37.27	76.47	68.95↓	39.24
GUI-Owl-7B	58.33	68.80↑	25.90	55.84	70.23↑	29.58	58.82	47.97↓	37.64
Qwen2.5-VL-72B	65.62	73.61↑	21.22	87.01	77.49↓	27.39	74.79	67.72↓	28.75
UI-TARS-72B-DPO	74.48	80.10↑	28.57	48.05	61.64↑	35.09	57.14	46.04↓	43.07

Table 2: The overall state-level accuracy performance of four GUI agents in the three applications.

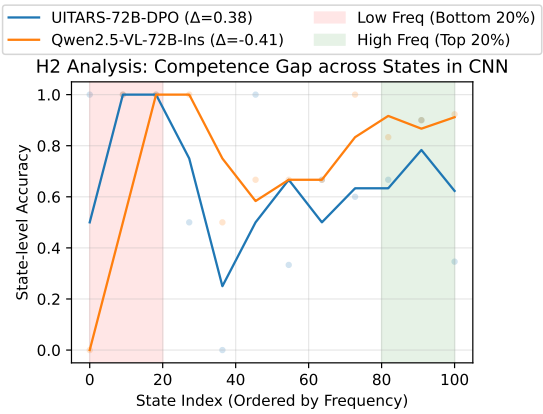

Figure 4: The state-level accuracy across frequency.

prompts used at different stages of STG construction are documented in Appendix I.1. We evaluate a total of eight GUI agents, including Qwen2.5-VL-Instruct (7B, 72B), UITARS (1.5–7B, 72B-DPO), GUI-Owl (7B, 32B), OS-Atlas-Pro-7B, Aguivis-7B. To facilitate focused analysis, we present detailed results for four representative agents in the main text, while full results for all evaluated agents are deferred to the Appendix. Prompts are constructed following each model's card. All experiments are conducted on NVIDIA A100-80G GPUs.

5.2 Competence Distribution Across Visitation Frequencies

Step-level evaluation implicitly weights states by their visitation frequency. Consequently, its reliability depends largely on how competence is distributed between frequently and infrequently accessed states. To address this, we investigate whether competence is evenly distributed across the full frequency range in real GUI trajectories.

For each agent–application pair, we analyze the relationship between state-level execution accuracy and visitation frequency. Without assuming a fixed correlation direction, we observe significant heterogeneity among different agents. Some agents exhibited lower accuracy in low-frequency states, while others demonstrated the opposite trend—performing better in rare states but worse in frequently accessed ones. This diversity indicates that execution competence does not always align with visitation frequency.

To quantify this effect, we divide states into low-frequency and high-frequency groups based on visitation frequency and calculate the accuracy gap between the two groups. Figure 4 presents typical examples of these contrasting patterns. Across all agent–application combinations, both positive and negative gaps are observed, confirming that different agents do not have identical competence in low-frequency and high-frequency regions.

Importantly, this non-uniform distribution of competence implies that step-level metrics are inherently influenced by randomness in the visitation distribution or by environment-specific biases. Regardless of whether this gap is positive or negative, SR cannot serve as a pure measure of competence for comparing agents. **These findings indicate that, in real GUI environments, visitation frequency and competence are fundamentally decoupled**, underscoring the necessity of frequency-aware, state-level evaluation.

5.3 Bottleneck Localization Beyond Step-Level Metrics

Given that execution competence is unevenly distributed across states with different visitation frequencies, we now examine whether a frequency-weighted step-level metric can accurately reflect agent competence and localize failure-prone states.

Discrepancies Between Step-Level and State-Level Metric. Table 2 presents the performance of four GUI agents across three applications. Substantial discrepancies between SR and S-Mean are observed across agent–application pairs. These differences are not random fluctuations; rather, they are systematically rooted in how competence is distributed across states with varying visitation frequencies. Since SR weights states according to their visitation frequency, it amplifies the influence

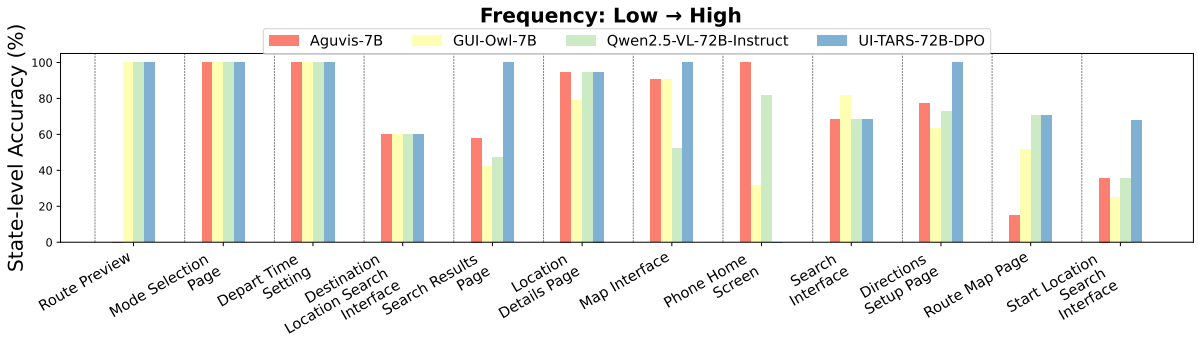

Figure 5: The state-level accuracy performance in Maps App.

State	TN	CN	Accuracy
Mode Selection Page	2	2	100.00
Depart Time Setting	4	4	100.00
Search Results Page	19	8	42.11
Start Location Search Interface	28	7	25.00

Table 3: Representative State-Level Accuracy of GUI-Owl-7B on Maps. TN denotes the total number of cases, and CN denotes the number of correct cases.

State	TN	CN	Accuracy
Home/Top News Page	10	10	100.00
Specific Article Page	26	19	73.08
Search Results Page	3	0	0.00
Confirmation Pop-up	2	0	0.00

Table 4: Representative State-Level Accuracy of Aguvis-7B on CNN. TN denotes the total number of cases, and CN denotes the number of correct cases.

of high-frequency states while suppressing the contribution of low-frequency states.

Why SR and S-Mean Diverge: Representative Case Analysis. To identify these discrepancies, we conduct a representative case analysis on selected agent–application pairs. Specifically, we examine GUI-Owl-7B on Maps and Aguvis-7B on CNN. The representative results are summarized in Table 3 and Table 4. When SR is lower than S-Mean, failures concentrate on high-frequency states, penalizing step-level metrics more heavily. Conversely, when SR exceeds S-Mean, low-competence states are infrequently visited, masking their impact in step-level evaluation while S-Mean fully reflects them. **Consequently, step-level metrics fail to surface such bottleneck states despite high failure rates.** This illustrates that SR prioritizes visitation frequency over failure severity, limiting bottleneck localization. Detailed state-level statistics for these cases are in Appendix E.

State-Level Heterogeneity and the Illusion of Robustness. We further observe substantial state-level performance variation, evidenced by high S-Std values across agents. This heterogeneity is obscured by step-level aggregation; states with extremely low accuracy coexist with highly reliable ones, diluting their impact under SR. Figure 5 illustrates state-level performance for four agents on Maps. While top agents achieve perfect accuracy on some states, others show consistently poor performance. These low-accuracy states represent frequent failure points. Consequently, even with 70–80% SR or S-Mean, these bottleneck states remain hidden, further demonstrating SR's limitations for failure localization.

5.4 Transition-Level Failure Attribution

Average step- and state-level metrics summarize performance but mask failure distributions along execution paths. To determine whether failures are associated with specific transitions, we analyze transition-level error distributions using the STG.

Transition-Level Failure Concentration. At a macro level, Figure 6 visualizes transition-level error distributions across four GUI agents on Maps. We observe that even the SOTA model, UI-TARS-72B-DPO, contains transition edges with a failure rate of 100%. This indicates that failures are not uniformly distributed but are highly concentrated on a few transitions, which is a pattern hidden by average metrics. Crucially, these failure behaviors vary by frequency, motivating finer-grained transition-level characterization.

Descriptive Transition-Level Failure Patterns. Beyond aggregate concentration, transition-level failures exhibit systematic structural patterns. To further characterize transition-level failure concentration, we analyze the transition statistics of

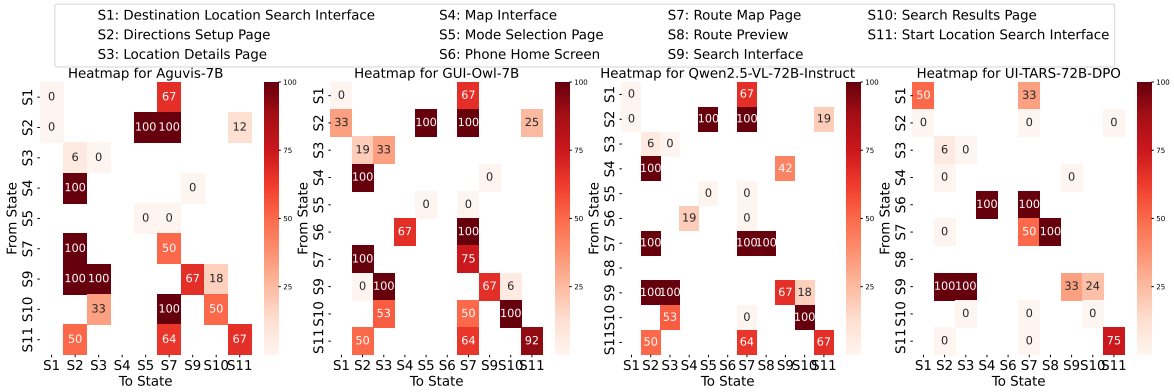

Figure 6: Transition-level bottleneck statistics on Maps, showing that a small number of transitions account for a disproportionate fraction of failures. States with errors are represented in the heatmap, indicating that fewer states depicted correspond to fewer errors at those states. Transitions that do not occur are omitted from the heatmap.

FS	TS	TF(%)	FCI(%)	Failure Modes
Search Results Page	Search Results Page	1.18	100.00	LCTP
Route Map Page	Route Map Page	1.76	100.00	LCTP
Start Location Search Interface	Start Location Search Interface	7.06	75.00	STFP
Start Location Search Interface	Route Map Page	8.24	85.71	STFP

Table 5: Transition-level failure statistics for Aguvis-7B on Maps. FS and TS denote the source and target states.

Aguvis-7B on the Maps application, as summarized in Table 5. We observe two representative transition-level patterns. To facilitate discussion, we introduce terms to describe two recurring transition-level failure patterns observed in our analysis. Importantly, these terms are used purely descriptively to characterize how failures concentrate along transitions.

(1) **Latent Critical Transition Pattern (LCTP).** This pattern refers to rarely traversed transitions with consistently high failure rates. Such transitions are invisible to frequency-weighted evaluation despite being deterministically failure-prone.

(2) **Systemic Transition Failure Pattern (STFP).** This pattern describes frequently executed, persistently failure-prone transitions, resulting in a concentration of failures along common paths.

These patterns reveal qualitatively different failure mechanisms that cannot be distinguished under step-level or state-level aggregation. Such state- and transition-level bottlenecks are observed consistently across different applications. Additional qualitative visualizations for other applications are presented in Appendix H.

5.5 On the Quality of Constructed STGs.

The effectiveness of GUITAR relies on the quality and stability of the constructed STGs. Due to space constraints, we defer a systematic analysis of graph quality to Appendix F. Specifically, the appendix examines (i) the stability of STGs constructed using different VLMs, (ii) the consistency of bottleneck diagnosis across different abstractions, and (iii) a comparison with similarity-threshold-based construction methods. We summarize the key finding here: the STGs are largely stable across VLMs, and the identified bottleneck transitions remain highly consistent, indicating that our diagnostic conclusions are robust to reasonable variations in the abstraction process.

6 Conclusion

In this work, we identify a Competence Illusion induced by step-level: they overfit high-frequency states while neglecting low-frequency ones due to long-tailed bias. To address this, we propose a state-centric schema and introduce GUITAR, a diagnostic framework using application-level STGs for state- and transition-aware analysis. By using semantic guidance to map screenshots to unique states, GUITAR distinguishes between high- and low-frequency scenarios. Our empirical results show GUITAR exposes concentrated failure patterns invisible to conventional metrics, proving the necessity of state- and transition-aware diagnostics for faithful GUI agent assessment.

Limitations

GUITAR relies on semantic abstraction to construct STGs, and its analysis quality depends on the fidelity of the resulting state representations. While our experiments demonstrate that the constructed STGs are sufficiently pure and effective for diagnosis, further improving the robustness and automation of state abstraction remains an open direction. In addition, our evaluation is conducted on a limited set of trajectories. Although GUITAR itself is not inherently restricted to small-scale data, we do not empirically study large-scale STG construction in this work. Efficient abstraction and analysis of STG from large-scale trajectory collections, therefore, remain to be explored. Finally, our analysis is primarily performed offline and focuses on diagnostic insight rather than direct performance improvement. While we include a pilot study on online diagnosis in Appendix G, a systematic investigation of how GUITAR-based diagnostics can be integrated into training or adaptive evaluation loops is left for future work.

Discussion. Our study shows that SR alone is insufficient for evaluating GUI agents in long-tailed environments, as it amplifies performance on high-frequency states while neglecting low-frequency ones. By introducing state- and transition-aware diagnostics, GUITAR complements step-level evaluation and explicitly reveals these hidden failure patterns. GUITAR is not intended to replace existing metrics, but to enhance them with structured diagnostic insights. While SR remains useful for coarse-grained tracking of overall task completion, GUITAR enables localized analysis of where and how execution failures occur, thereby providing a more faithful characterization of agent competence.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, and 1 others. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Rohan Anil, Andrew M Dai, Orhan Firat, Melvin Johnson, Dmitry Lepikhin, Alexandre Passos, Siamak Shakeri, Emanuel Taropa, Paige Bailey, Zhifeng Chen, and 1 others. 2023. Palm 2 technical report. *arXiv preprint arXiv:2305.10403*.

Chongyang Bai, Xiaoxue Zang, Ying Xu, Srinivas Sunkara, Abhinav Rastogi, Jindong Chen, and Blaise Agüera y Arcas. 2021. Uibert: Learning generic multimodal representations for ui understanding. In *IJCAI*.

Kanzhi Cheng, Qiushi Sun, Yougang Chu, Fangzhi Xu, Li YanTao, Jianbing Zhang, and Zhiyong Wu. 2024. Seeclick: Harnessing gui grounding for advanced visual gui agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 9313–9332.

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*, pages 845–854.

Boyu Gou, Ruohan Wang, Boyuan Zheng, Yanan Xie, Cheng Chang, Yiheng Shu, Huan Sun, and Yu Su. 2024. Navigating the digital world as humans do: Universal visual grounding for gui agents. *arXiv preprint arXiv:2410.05243*.

Wenyi Hong, Weihan Wang, Qingsong Lv, Jiazheng Xu, Wenmeng Yu, Junhui Ji, Yan Wang, Zihan Wang, Yuxiao Dong, Ming Ding, and 1 others. 2024. Cogagent: A visual language model for gui agents. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14281–14290.

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

Kaixin Li, Ziyang Meng, Hongzhan Lin, Ziyang Luo, Yuchen Tian, Jing Ma, Zhiyong Huang, and Tat-Seng Chua. 2025. Screenspot-pro: Gui grounding for professional high-resolution computer use. *arXiv preprint arXiv:2504.07981*.

Wei Li, William E Bishop, Alice Li, Christopher Rawles, Folawiyo Campbell-Ajala, Divya Tyamagundlu, and Oriana Riva. 2024a. On the effects of data scale on ui control agents. *Advances in Neural Information Processing Systems*, 37:92130–92154.

Yanda Li, Chi Zhang, Wanqi Yang, Bin Fu, Pei Cheng, Xin Chen, Ling Chen, and Yunchao Wei. 2024b. Appagent v2: Advanced agent for flexible mobile interactions. *arXiv preprint arXiv:2408.11824*.

Quanfeng Lu, Wenqi Shao, Zitao Liu, Fanqing Meng, Boxuan Li, Botong Chen, Siyuan Huang, Kaipeng Zhang, Yu Qiao, and Ping Luo. 2024. Gui odyssey: A comprehensive dataset for cross-app gui navigation on mobile devices. *arXiv preprint arXiv:2406.08451*.

Xinbei Ma, Zhuosheng Zhang, and Hai Zhao. 2024. Coco-agent: A comprehensive cognitive mllm agent for smartphone gui automation. In *Findings of the Association for Computational Linguistics ACL 2024*, pages 9097–9110.

Yaoyao Qian, Yuanli Wang, Jinda Zhang, Yun Zong, Meixu Chen, Hanhan Zhou, Jindan Huang, Yifan Zeng, Xinyu Hu, Chan Hee Song, and 1 others. 2025. Webgrapheval: Multi-turn trajectory evaluation for web agents using graph representation. *arXiv preprint arXiv:2510.19205*.

Yujia Qin, Yining Ye, Junjie Fang, Haoming Wang, Shihao Liang, Shizuo Tian, Junda Zhang, Jiahao Li, Yunxin Li, Shijue Huang, and 1 others. 2025. Ui-tars: Pioneering automated gui interaction with native agents. *arXiv preprint arXiv:2501.12326*.

Christopher Rawles, Sarah Clinckemaillie, Yifan Chang, Jonathan Waltz, Gabrielle Lau, Marybeth Fair, Alice Li, William Bishop, Wei Li, Folawiyo Campbell-Ajala, and 1 others. 2024. Androidworld: A dynamic benchmarking environment for autonomous agents. *arXiv preprint arXiv:2405.14573*.

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

Liangtai Sun, Xingyu Chen, Lu Chen, Tianle Dai, Zichen Zhu, and Kai Yu. 2022. Meta-gui: Towards multi-modal conversational agents on mobile gui. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 6699–6712.

Haoming Wang, Haoyang Zou, Huatong Song, Jiazhan Feng, Junjie Fang, Junting Lu, Longxiang Liu, Qinyu Luo, Shihao Liang, Shijue Huang, and 1 others. 2025a. Ui-tars-2 technical report: Advancing gui agent with multi-turn reinforcement learning. *arXiv preprint arXiv:2509.02544*.

Junyang Wang, Haiyang Xu, Haitao Jia, Xi Zhang, Ming Yan, Weizhou Shen, Ji Zhang, Fei Huang, and Jitao Sang. 2024. Mobile-agent-v2: Mobile device operation assistant with effective navigation via multi-agent collaboration. *Advances in Neural Information Processing Systems*, 37:2686–2710.

Xuehui Wang, Zhenyu Wu, JingJing Xie, Zichen Ding, Bowen Yang, Zehao Li, Zhaoyang Liu, Qingyun Li, Xuan Dong, Zhe Chen, and 1 others. 2025b. Mmbench-gui: Hierarchical multi-platform evaluation framework for gui agents. *arXiv preprint arXiv:2507.19478*.

Zhiyong Wu, Zhenyu Wu, Fangzhi Xu, Yian Wang, Qiushi Sun, Chengyou Jia, Kanzhi Cheng, Zichen Ding, Liheng Chen, Paul Pu Liang, and 1 others. 2024. Os-atlas: A foundation action model for generalist gui agents. *arXiv preprint arXiv:2410.23218*.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh J Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, and 1 others. 2024. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments. *Advances in Neural Information Processing Systems*, 37:52040–52094.

An Yan, Zhengyuan Yang, Wanrong Zhu, Kevin Lin, Linjie Li, Jianfeng Wang, Jianwei Yang, Yiwu Zhong, Julian McAuley, Jianfeng Gao, and 1 others. 2023. Gpt-4v in wonderland: Large multimodal models for zero-shot smartphone gui navigation. *arXiv preprint arXiv:2311.07562*.

Hui Yang, Sifu Yue, and Yunzhong He. 2023. Auto-gpt for online decision making: Benchmarks and additional opinions. *arXiv preprint arXiv:2306.02224*.

Jiabo Ye, Xi Zhang, Haiyang Xu, Haowei Liu, Junyang Wang, Zhaoqing Zhu, Ziwei Zheng, Feiyu Gao, Junjie Cao, Zhengxi Lu, and 1 others. 2025. Mobile-agent-v3: Fundamental agents for gui automation. *arXiv preprint arXiv:2508.15144*.

Shaoqing Zhang, Kehai Chen, Zhuosheng Zhang, Rumei Li, Rongxiang Weng, Yang Xiang, and Min Zhang. 2025. Xbound: Exploring capability boundaries of device-control agents at the state level. *Preprint*, arXiv:2505.21279.

Shaoqing Zhang, Zhuosheng Zhang, Kehai Chen, Xinbei Ma, Muyun Yang, Tiejun Zhao, and Min Zhang. 2024. Dynamic planning for llm-based graphical user interface automation. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 1304–1320.

Zhuosheng Zhang and Aston Zhang. 2024. You only look at screens: Multimodal chain-of-action agents. In *Findings of the Association for Computational Linguistics ACL 2024*, pages 3132–3149.

Zihan Zheng, Tianle Cui, Chuwen Xie, Jiahui Zhang, Jiahui Pan, Lewei He, and Qianglong Chen. 2025. Naturegaia: Pushing the frontiers of gui agents with a challenging benchmark and high-quality trajectory dataset. *arXiv preprint arXiv:2508.01330*.

Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, and 1 others. 2024. Webarena: A realistic web environment for building autonomous agents. In *12th International Conference on Learning Representations, ICLR 2024*.

A Ethical considerations

A.1 AI Disclosure

The authors utilized AI tools solely for linguistic assistance to enhance clarity and readability. The underlying research, including all GUI agent experiments and performance evaluations, is original and reproducible.

A.2 Data Contains Personally Identifying Info Or Offensive Content

The data used in this paper are sourced from open-source repositories and are free of any personally identifiable information or sensitive/offensive content, ensuring the privacy and anonymity of all individuals.

B Synthetic Demonstration of Frequency-Induced Evaluation Bias

This appendix presents a minimal synthetic simulation that isolates the effect of visitation frequency on evaluation metrics. The simulation is independent of any graphical interface, agent architecture, or state abstraction error, and serves to illustrate that frequency-induced bias is an inherent property of step-level evaluation.

Construction. We fix a set of $N = 50$ states, each assigned a latent competence $c(s)$, which linearly decreases from 0.9 to 0.2 as state frequency decreases. This competence distribution remains unchanged across all experimental conditions. We then generate trajectories by sampling states according to different visitation distributions $p(s)$, ranging from a uniform distribution to increasingly skewed Zipf distributions. For each visited state, success or failure is sampled according to its fixed competence $c(s)$. Importantly, agent behavior and environment dynamics are identical across all conditions; the only varying factor is the visitation distribution.

We compare two evaluation quantities: SR and the true uniform state competence S-Mean, which serves as a reference. As shown in Table 6, as the visitation distribution becomes more skewed, SR increases monotonically despite the underlying competence remaining constant, while S-Mean stays unchanged.

Conclusion. These results show that SR can be arbitrarily inflated solely due to visitation frequency skew, thereby confirming the existence of frequency-induced evaluation bias. **By construction, this bias is independent of agent quality or environmental complexity, highlighting the necessity of state-aware evaluation mechanisms for reliable competence assessment.**

Visitation Distributions	SR	S-Mean
Uni	54.58	55.00
Zipf ($\alpha = 1$)	75.72	55.00
Zipf ($\alpha = 2$)	88.18	55.00

Table 6: Synthetic Simulation of Evaluation Bias. While the underlying state-level competencies are fixed, the SR is heavily inflated by skewed visitation frequencies (Zipf distributions), whereas the S-Mean remains invariant and provides an unbiased estimate of true competence.

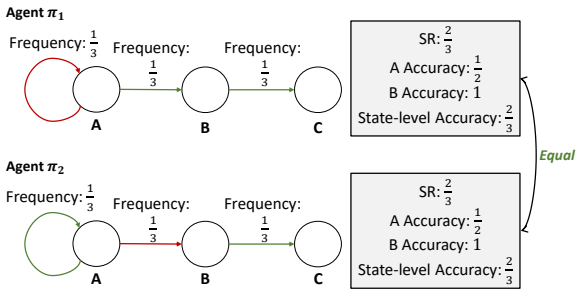

Figure 7: The step-level and state-level fail to distinguish the failure locations of π_1 from π_2.

C Formal Proof of Non-Equivalence to State-Level Aggregation

This appendix provides a formal construction supporting Proposition 1 in Section 3.2, showing that state-transition analysis captures failure behaviors that are invisible to any scalar or state-level aggregation.

Definition (Scalar and State-Level Aggregation). Let $\mathcal{T}$ denote a set of execution trajectories. A *scalar aggregation* is any evaluation function $M : \mathcal{T} \to \mathbb{R}$ that maps trajectories to a single real-valued score. A *state-level aggregation* computes statistics independently for each abstract state (e.g., success rates) and aggregates them without preserving transition identity.

Construction. Consider an environment with abstract states $\{A, B, C\}$ and transitions $A \to A \to B \to C$, where the self-transition $A \to A$ corresponds to repeated interaction within the same UI state.

Define two agents:

- Agent π_1 frequently fails at the self-transition $A \to A$, but reliably executes $A \to B$ and $B \to C$.
- Agent π_2 reliably executes $A \to A$, but frequently fails at the transition $A \to B$.

Both agents visit state A with identical frequency and achieve identical success rates when outcomes are aggregated at the state level. Consequently, no state-level aggregation can distinguish π_1 from π_2 or localize their failures to different states. Figure 7 provides a toy example illustrating this structural information loss.

State-Transition Distinction. In contrast, the STG induced by GUITAR exposes distinct failure bottlenecks. Agent π_1 exhibits concentrated failures on $A \to A$, reflecting difficulty in repeated intra-state execution, whereas agent π_2 fails predominantly on $A \to B$, reflecting difficulty in exiting the state. These failure modes are topologically distinct and cannot be recovered from any aggregation that collapses transition identity.

Conclusion. This construction demonstrates that state-level aggregation inherently collapses transition structure and is fundamentally insufficient for failure localization. **Preserving explicit state-transition structure, as in GUITAR, is therefore necessary for diagnosing agent competence beyond what aggregation-based metrics can provide.**

D Complete Experimental Results and Statistics

This appendix reports the complete state-level accuracy results for all evaluated GUI agents and applications. These results complement the main paper by providing full coverage across models and tasks, while the main text focuses on representative cases for detailed analysis.

D.1 Data Statistics

Figure 8 reports detailed data statistics for three representative applications—Maps, CNN, and Kitchen Stories. Figure 9 presents the data statistics for four applications: Arts & Culture, Flipboard, Guardian, and Pinterest.

D.2 Results on Representative Applications

Table 7 presents the state-level accuracy results of additional GUI agents on the three representative applications: Maps, CNN, and Kitchen Stories,

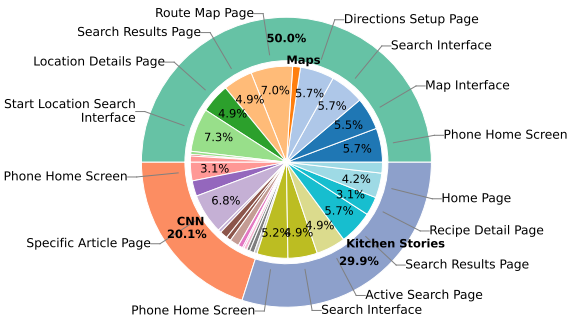

Figure 8: The data statistics for three applications: Maps, CNN, and Kitchen Stories.

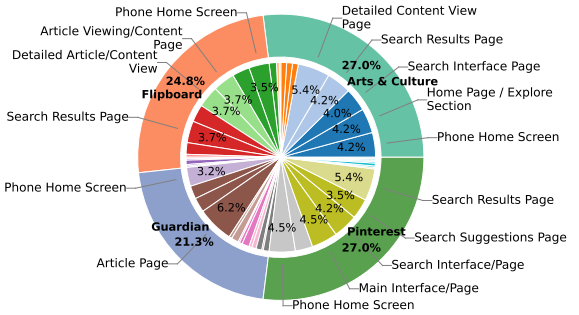

Figure 9: The data statistics for four applications: Arts & Culture, Flipboard, Guardian, and Pinterest.

which used throughout the main paper. Across different models, we consistently observe discrepancies between step-level success rate (SR) and uniform state-level accuracy (S-Mean), indicating that frequency-induced evaluation bias persists beyond the agents discussed in the main text.

D.3 Results on Additional Applications

Table 8 reports state-level accuracy results on four additional applications: Arts & Culture, Flipboard, Guardian, and Pinterest. Despite differences in interface structure and interaction complexity, the same qualitative trend is observed: step-level SR frequently diverges from S-Mean, reflecting uneven competence across states. These results further support the generality of the frequency-induced evaluation bias identified in the main paper.

Overall, these comprehensive results confirm that the evaluation bias observed in the main paper is not restricted to specific agents or applications.

E Detailed Case-Level State Accuracy Analysis

This section provides detailed state-level accuracy statistics for the representative cases discussed in the main text, including GUI-Owl-7B on Maps and Aguvis-7B on CNN. Table 9 and Table 10

Model	Maps			CNN			Kitchen Stories		
	SR	S-Mean	S-Std	SR	S-Mean	S-Std	SR	S-Mean	S-Std
OS-Atlas-7B-Pro	47.39	47.20↓	31.83	67.53	62.54↓	35.30	64.71	59.30↓	39.14
Qwen2.5-VL-7B	69.79	75.15↑	23.81	79.22	83.58↑	28.85	68.91	56.28↓	37.51
GUI-Owl-32B	58.85	57.63↓	27.23	54.55	66.04↑	37.68	63.87	55.49↓	34.54
UI-TARS-1.5-7B	58.33	56.43↓	34.24	31.17	43.53↑	38.59	55.46	47.33↓	36.53

Table 7: Complete state-level accuracy results (SR, S-Mean, S-Std) of additional GUI agents on three representative applications.

Model	Arts & Culture			Flipboard			Guardian			Pinterest		
	SR	S-Mean	S-Std	SR	S-Mean	S-Std	SR	S-Mean	S-Std	SR	S-Mean	S-Std
OS-Atlas-7B-Pro	44.14	39.45↓	32.08	46.53	39.72↓	34.95	57.95	55.88↓	31.88	57.39	61.06↑	35.43
Aguvis-7B	71.17	54.70↓	38.13	62.38	50.32↓	37.34	65.91	53.18↓	33.47	68.70	67.15↓	36.23
GUI-Owl-7B	45.95	45.95-	36.41	45.36	45.70-	35.12	38.64	38.61-	30.96	60.87	64.34↑	35.39
GUI-Owl-32B	49.55	51.25↑	28.04	49.50	57.06↑	36.79	40.91	45.35↑	33.63	59.13	71.98↑	33.62
Qwen2.5-VL-7B	54.05	55.33↑	33.43	59.41	61.33↑	34.56	54.55	44.41↓	34.81	69.57	74.08↑	30.91
Qwen2.5-VL-72B	63.06	63.51-	35.17	64.36	62.05↓	31.72	69.32	59.51↓	32.66	73.04	76.00↑	29.02
UI-TARS-1.5-7B	43.24	43.09-	33.04	40.59	46.85↑	37.76	37.5	49.58↑	38.73	60.00	60.72-	41.88
UI-TARS-72B-DPO	51.35	60.07↑	41.45	60.40	73.33↑	37.49	48.86	60.69↑	35.77	62.61	82.66↑	31.18

Table 8: Complete state-level accuracy results (SR, S-Mean, S-Std) on four additional applications across evaluated GUI agents.

support the case analysis explaining the divergence between SR and S-Mean.

For the GUI-Owl-7B–Maps combination, we observe that several low-frequency states, such as *Mode Selection Page*, *Route Preview*, and *Depart Time Setting*, achieve relatively high accuracy. In contrast, high-frequency states including *Start Location Search Interface*, *Search Results Page*, and *Route Map Page* exhibit noticeably lower accuracy. As a result, after decoupling state frequency from execution capability, the N-Mean score of GUI-Owl-7B on Maps increases compared to its instruction-level success rate (SR).

Conversely, in the Aguvis-7B–CNN setting, the agent performs particularly well on high-frequency states such as *Home/Top News Page*, *Active Search Entry*, and *Specific Article Page*. However, its performance degrades substantially on low-frequency states, including *Search Results Page*, *Settings Menu*, and *Confirmation Pop-up*. This imbalance leads to a decrease in N-Mean relative to SR, indicating that the agent's strong performance is concentrated in frequently visited states while less common states remain fragile.

State	TN	CN	Accuracy
Mode Selection Page	2	2	100.00
Route Preview	1	1	100.00
Depart Time Setting	4	4	100.00
Map Interface	21	19	90.48
Search Interface	22	18	81.82
Location Details Page	19	15	78.95
Directions Setup Page	22	14	63.64
Destination Location Search Interface	5	3	60.00
Route Map Page	27	14	51.85
Search Results Page	19	8	42.11
Start Location Search Interface	28	7	25.00

Table 9: The State-Level Accuracy of GUI-Owl-7B on Maps. TN denotes the total number of cases, and CN denotes the number of correct cases.

F Graph Quality

In this section, we focus on the quality of graphs constructed by GUITAR. We discuss this from three aspects: (1) The stability of STGs constructed using different VLMs, (2) The consistency of bottleneck diagnosis, and (3) A comparison with similarity-threshold-based construction methods.

State	TN	CN	Accuracy
Home/Top News Page	10	10	100.00
Sharing Interface	2	2	100.00
Active Search Entry	6	6	100.00
Gmail Compose Screen	5	4	80.00
Specific Article Page	26	19	73.08
Search Section	3	2	66.67
Category Page	3	2	66.67
Alert Settings Page	2	1	50.00
Notification Settings Page	2	1	50.00
Search Results Page	3	0	0.00
Settings Menu	1	0	0.00
Confirmation Pop-up	2	0	0.00

Table 10: The State-Level Accuracy of Aguvis-7B on CNN. TN denotes the total number of cases, and CN denotes the number of correct cases.

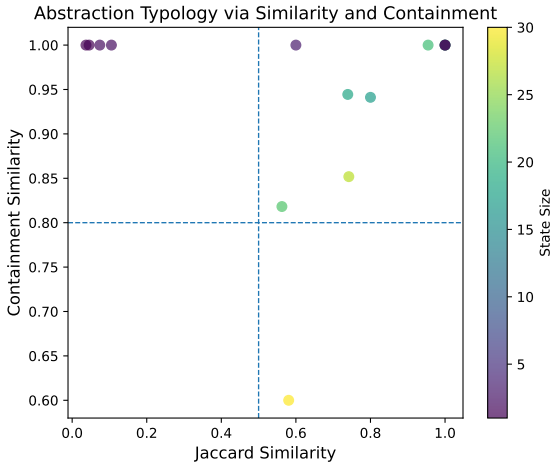

Figure 10: State-level abstraction similarity between STG_2 and STG_1 plotted using symmetric Jaccard similarity (x-axis) and asymmetric containment similarity (y-axis). Each point corresponds to a state in STG_2. Color indicates the number of frames assigned to that state.

F.1 Abstraction Stability Analysis Across Different VLMs

In this section, we analyze the stability of STG abstractions constructed using different VLMs. Our goal is to verify that GUITAR's abstract state transition graphs are not significantly affected by the choice of model, and that strong VLMs are capable of capturing the logical relationships underlying abstraction.

Specifically, we compare STGs independently constructed by two different VLMs (GPT-4o and Qwen2.5-VL-72B), denoted as STG_1 and STG_2, respectively. Both are based on the same set of raw GUI trajectories, and we quantify their similarity at the state level.

F.1.1 State-Level Similarity Measures

Let $S_1 = \{s_1^1, \ldots, s_1^n\}$ and $S_2 = \{s_2^1, \ldots, s_2^m\}$ denote the sets of states in STG_1 and STG_2, respectively. Each state corresponds to a set of screenshots grouped by the abstraction process.

For each pair (s_1, s_2), we define the following similarity metrics:

Jaccard Similarity.

$$\text{Jaccard}(s_1, s_2) = \frac{|\mathcal{F}(s_1) \cap \mathcal{F}(s_2)|}{|\mathcal{F}(s_1) \cup \mathcal{F}(s_2)|}, \quad (4)$$

where $\mathcal{F}(s)$ denotes the set of screenshots assigned to state s.

Containment Similarity.

$$\text{Containment}(s_2 \to s_1) = \frac{|\mathcal{F}(s_1) \cap \mathcal{F}(s_2)|}{|\mathcal{F}(s_2)|}. \quad (5)$$

Containment measures whether a state in STG_2 is largely contained within a single state in STG_1, even if the latter is coarser.

For each $s_2 \in S_2$, we compute its best match in S_1:

$$\text{Sim}(s_2) = \max_{s_1 \in S_1} \text{Jaccard}(s_1, s_2), \quad (6)$$

$$\text{Cont}(s_2) = \max_{s_1 \in S_1} \text{Containment}(s_2 \to s_1) \quad (7)$$

F.1.2 Abstraction Typology

By plotting each state s_2 in the 2D space $(\text{Sim}(s_2), \text{Cont}(s_2))$, we obtain a typology of abstraction behaviors:

- High Sim, High Cont: consistent abstractions;
- Low Sim, High Cont: over-splitting in STG_2;
- High Sim, Low Cont: over-merging in STG_2;
- Low Sim, Low Cont: structurally divergent abstractions.

F.1.3 Visualization and Empirical Results

Figure 10 visualizes this typology for the Maps application.

We observe that most states fall within the high-similarity region, indicating that abstract STGs constructed with different VLMs are generally stable. A small number of states exhibit low Jaccard similarity but high containment, suggesting that STG_2 subdivides certain coarse states from STG_1 into

Model	Top-3	Top-5	Top-10
Aguvis-7B	33.33	40.00	80.00
GUI-Owl-7B	33.33	40.00	70.00
Qwen2.5-VL-72B	66.67	60.00	70.00
UI-TARS-72B-DPO	66.67	40.00	60.00

Table 11: Top-k bottleneck overlap between STG_1 and STG_2.

finer-grained sub-states. Conversely, states with moderate Jaccard similarity and low containment reflect occasional over-merging. These results indicate that GUITAR is applicable to different VLMs, whether open-source or closed-source, and that STGs constructed via VLM-guided semantic abstraction are stable.

F.2 Diagnostic Stability via Top-K Bottleneck Overlap

We next examine whether different STGs lead to consistent diagnostic conclusions. Let $\mathcal{B}_k^{(1)}$ and $\mathcal{B}_k^{(2)}$ denote the sets of top-k bottleneck transitions identified from STG_1 and STG_2 respectively, ranked by bottleneck score bottleneck score $= TF \times FCI$. Considering that the transition representations of different STGs may differ, we compare the images contained in these transitions. We compute their overlap as:

$$\text{Overlap}_k = \frac{|\mathcal{B}_k^{(1)} \cap \mathcal{B}_k^{(2)}|}{k}.$$

Table 11 reports the overlap of the top-k bottleneck transitions identified from STG_1 and STG_2. We observe that the overlap increases with k, reaching 60%–80% at Top-10, indicating that the set of failure-prone transitions is largely consistent across different abstractions. At the same time, the Top-3 overlap is lower, suggesting that fine-grained ranking among the most severe bottlenecks is sensitive to abstraction granularity, while the overall diagnostic signal remains stable.

Together, these results demonstrate that: (1) the abstraction induced by different VLMs is largely stable, with deviations limited to interpretable split/merge variations; and (2) These abstraction differences do not materially affect diagnostic outcomes.

Thus, GUITAR's bottleneck identification is robust to reasonable abstraction variability and does not depend on idiosyncrasies of a specific VLM.

F.3 Compare with Incremental State Graph Construction

In this section, we compare the proposed GUITAR method with the traditional Incremental State Graph Construction (ISGC) method to evaluate the effectiveness of semantic information in state merging. The core mechanism of ISGC lies in merging states based on similarity. When the feature representation of a new screenshot (typically visual features) exceeds a predefined similarity threshold with any existing state in the graph, the screenshot is merged into the most similar state. Otherwise, if no similarity exceeds the threshold, a new state is created.

ISGC Experimental Setup To comprehensively assess the contributions of semantic and visual information, we design four ISGC-based graph construction settings for comparison:

- **Pure Visual Similarity:** Uses CLIP image embeddings as state features, and computes state similarity purely in the visual embedding space.
- **Pure Textual Similarity:** For each screenshot, a vision-language model generates a concise semantic label (e.g., "Login Interface"). Similarity is computed solely based on CLIP embeddings of these textual descriptions.
- **Fused Similarity:** Combines CLIP image embeddings with embeddings of VLM-generated textual descriptions through weighted fusion.
- **ImageHash:** Uses perceptual hashing to compute a binary hash for each screenshot and measures similarity via Hamming distance between hashes.

The CLIP model used is *clip-vit-base-patch16*. Since edge merging is primarily based on textual embeddings, the edge similarity threshold in ISGC is uniformly set to 0.65 across all similarity-based variants. For state merging, the similarity threshold is chosen to yield a comparable number of abstract states to GUITAR, ensuring a fair comparison in abstraction granularity. Specifically, the state similarity thresholds are set to 0.65 for Pure Visual Similarity, 0.85 for Pure Textual Similarity, and 0.75 for Fused Similarity. In the Fused Similarity configuration, the image embedding is assigned a weight of 0.6, and the text embedding is assigned a weight of 0.4. For the ImageHash baseline, we use

a standard perceptual hashing scheme and merge two screenshots into the same state if their Hamming distance is below 35. This threshold is selected to roughly match the abstraction granularity of the other baselines.

Graph Quality Evaluation Metrics. We adopt two key metrics to assess the quality of the constructed graphs. To ensure an objective and authoritative ground truth, we recruit two independent human annotators with expertise in GUI automation and computer science. Together, these metrics capture both structural validity and state-level accuracy:

- **State Redundancy Rate (SRR):** Measures the proportion of redundant states, i.e., cases where similar or equivalent screenshots are incorrectly split into multiple states. A higher SRR indicates more unmerged, redundant states.

- **State Purity (SP):** Measures within-state clustering quality, i.e., the proportion of screenshots grouped under the same state that are truly semantically equivalent. A higher SP indicates more accurate clustering and lower confusion in the abstraction process.

To ensure the objectivity of state-graph evaluation, we establish a standardized three-stage annotation protocol carried out collaboratively by two independent annotators. The procedure is detailed as follows:

Stage 1: Anchor State Identification and Representative Image Selection. First, we construct a reference image set for each generated state. From the raw screenshots assigned to a state, we select highly representative images as annotation anchors. The selection criteria include: (1) Statistical significance, i.e., the most frequently occurring visual states within the state; and (2) Semantic alignment, i.e., interfaces that most clearly reflect the functional label of the state. To enhance evaluation robustness, we intentionally retain visually diverse samples within the reference set.

Stage 2: Inter-State Redundancy Assessment (for SRR). At this stage, annotators perform cross-state comparisons of reference image sets to determine whether semantically and functionally equivalent states have been incorrectly split into multiple states. If two or more states are indistinguishable in functional logic (e.g., separated solely due to differences in scroll position), they are recorded as redundant states. Based on these annotations, we compute the SRR.

Method	State Number	SRR	SP
Pure Visual Similarity	20	50.00	45.66
Pure Textual Similarity	15	40.00	67.59
Fused Similarity	19	47.37	70.38
ImageHash	19	42.11	64.40
GUITAR	**12**	**0.00**	**95.50**

Table 12: Comparison GUITAR method with the ISGC approach.

Stage 3: Intra-State Consistency Assessment (for SP) Finally, annotators conduct a fine-grained audit of the image members within each state. Each screenshot is compared against the state's reference images. A screenshot is counted as correctly clustered only if it is fully equivalent in functional logic to the reference images. SP is computed as the proportion of correctly clustered screenshots relative to the total number of screenshots within the state, including those from similar-state clusters identified in Stage 2.

Comparison Result. Table 12 reports the comparison between GUITAR and four similarity-threshold-based graph construction baselines, including visual, textual, fused similarity, and perceptual hashing (ImageHash) in Maps application. We summarize the main findings as follows.

(1) **GUITAR mitigates state explosion.** GUITAR constructs only 12 abstract states, whereas the similarity-based methods produce between 15 and 20 states. This suggests that threshold-based methods are prone to over-segmentation, introducing spurious states due to superficial visual or textual variations in GUI screenshots.

(2) **GUITAR substantially reduces abstraction redundancy.** GUITAR achieves an SRR of 0%, while all similarity-based baselines exhibit high redundancy (SRR between 40% and 50%), indicating that a large fraction of their constructed states are semantically redundant. This highlights the difficulty of avoiding duplicate abstractions when relying solely on similarity thresholds.

(3) **GUITAR yields significantly higher semantic coherence.** GUITAR attains a State Purity (SP) of 95.50%, substantially higher than all baselines (64.40%–70.38%). This indicates that GUITAR not only merges states more aggressively, but does so in a semantically consistent manner, resulting in

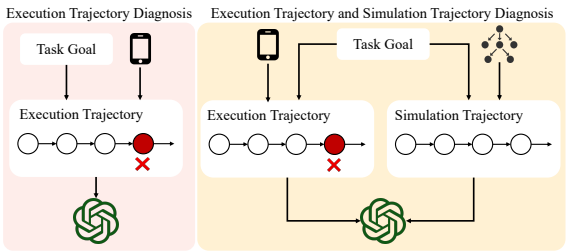

Figure 11: The comparison between these two online error diagnosis methods.

more coherent and interpretable abstractions.

(4) **Semantic reasoning outperforms similarity-based heuristics.** Among the threshold-based baselines, the Fused Similarity and ImageHash methods, which incorporate visual information, do illustrates a trade-off between over-merging and over-splitting: while Fused and ImageHash slightly improve SP over Pure Visual similarity, they also introduce more states and higher redundancy. In contrast, GUITAR avoids this trade-off by explicitly reasoning over GUI-level semantics, rather than relying on pairwise similarity heuristics.

Overall, these results suggest that similarity-threshold-based methods struggle to balance state compactness, redundancy, and semantic coherence, whereas GUITAR achieves a more favorable trade-off by leveraging semantic abstraction guided by large language models.

Why does ISGC fail in GUI state clustering? The shortcomings of traditional ISGC approaches arise from the inherently GUI visual nature:

(1) **High intra-class variability.** The same state can vary significantly due to dynamic content or asynchronous loading. ISGC often mistakes these variations for entirely new states.

(2) **Low inter-class visual contrast.** Different states may share nearly identical layouts and color schemes. Relying solely on pixel-level similarity makes it difficult for ISGC to distinguish such subtle semantic boundaries.

G Pilot Study on Online Diagnosis

To quantitatively evaluate GUITAR's effectiveness in online root cause analysis within dynamic environments, we utilize GPT-4o on the AndroidWorld benchmark to deliberately collect 27 task failure cases. We design a comparative experiment to assess GPT-4o's ability to identify the causes of GUI agent task failures. The prompts are provided in Appendix I.2. Figure 11 and Table 13 compares two distinct diagnostic settings:

Method	EIA	Win Rate
Only Execution Trajectory	77.78%	25.93%
GUITAR	92.59%	74.07%

Table 13: Accuracy of GPT-4o in diagnosing the causes of online task failures under the two methods. EIA represents Error Identification Accuracy

(1) Baseline (Execution Trajectory Only): The diagnoser is provided solely with the raw execution logs, including the sequence of thinking and actions.

(2) GUITAR-Enhanced (With Simulation Trajectory): The diagnoser is provided with the execution logs supplemented by the simulation trajectory generated by GUITAR.

In a preliminary experiment with 27 sampled failure cases, we observe that the GUITAR-enabled method correctly identifies the error cause in 25 instances (92.59%), compared to only 77.78% for the baseline. Furthermore, we conduct a qualitative analysis to assess the diagnostic quality and interpretability. We task an independent GPT-4o to compare the diagnostic output of the two settings based on five critical dimensions: Information Completeness, Logical Rigor, Practicality, Innovativeness, and Data/Evidence Support. The GUITAR-Enhanced method achieved a decisive 77.78% win rate. While the sample size is limited, these results qualitatively demonstrate that the structured abstraction provided by GUITAR is fundamentally necessary for robust and high-fidelity root cause analysis.

Figure 12 compares two error diagnosis methods: execution-only diagnosis and simulation-augmented diagnosis. In this task, the agent is required to create an event in a calendar application. The left panel shows the Execution Trajectory, while the middle panel presents the corresponding Simulation Trajectory.

H Additional Qualitative Analysis of Bottleneck Patterns

This section presents additional qualitative evidence demonstrating that transition-level failure concentration is not unique to the Maps application discussed in the main text. We visualize bottleneck patterns on two additional applications with distinct interface structures.

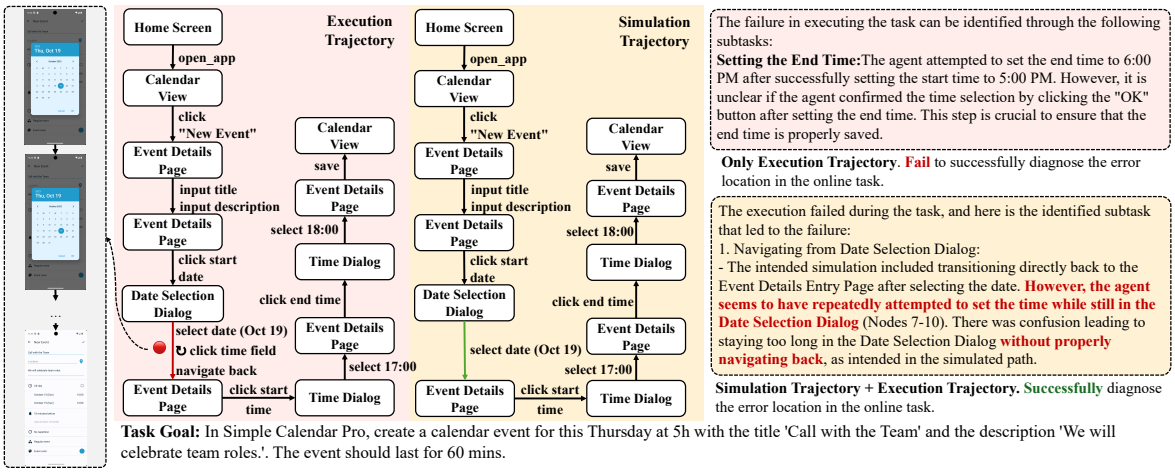

Figure 12: Comparison of two error diagnosis approaches. Execution-only diagnosis fails to locate the error, while simulation-assisted diagnosis successfully identifies the error and its cause.

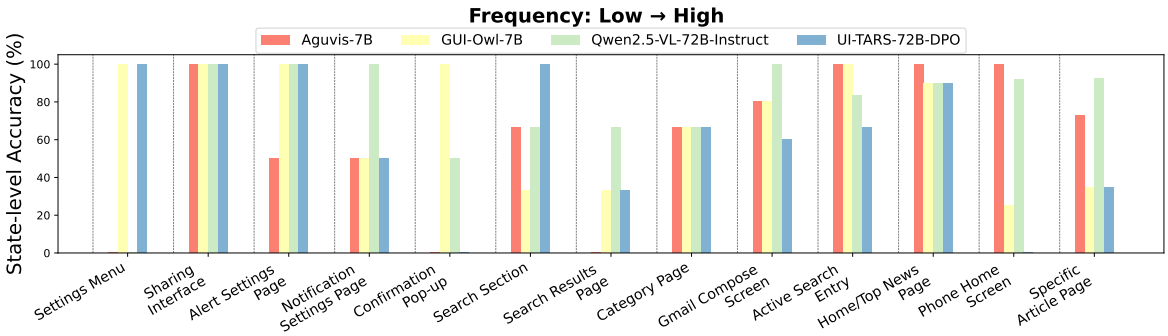

Figure 13: Transition-level bottleneck statistics on the CNN application, showing that a small number of transitions account for a disproportionate fraction of failures.

H.1 CNN

Figures 13 and 14 show that, similar to the Maps application, failures on CNN are highly concentrated on a small subset of transitions, despite diverse execution paths.

H.2 Kitchen Stories

Figures 15 and 16 reveal a similar failure concentration pattern, where a small number of transitions dominate overall errors, further supporting the generality of GUITAR's bottleneck analysis across applications.

I Prompts

I.1 The Prompts for Building STGs

I.2 The Prompts for Online Diagnosis

In this section, we present the prompts used for simulating task execution trajectories, GUITAR-based online diagnosis, and baseline online diagnosis.

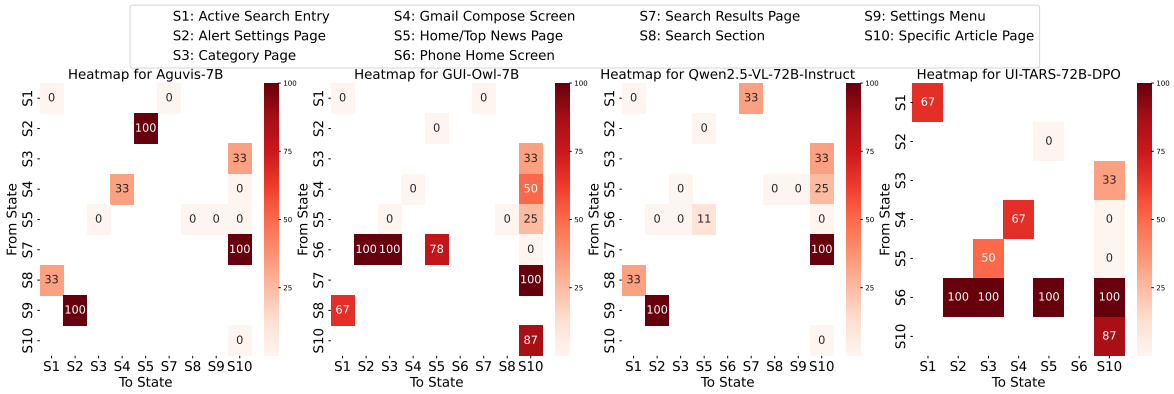

Figure 14: Transition-level error heatmap on the CNN state-transition graph. Darker edges indicate higher failure concentration.

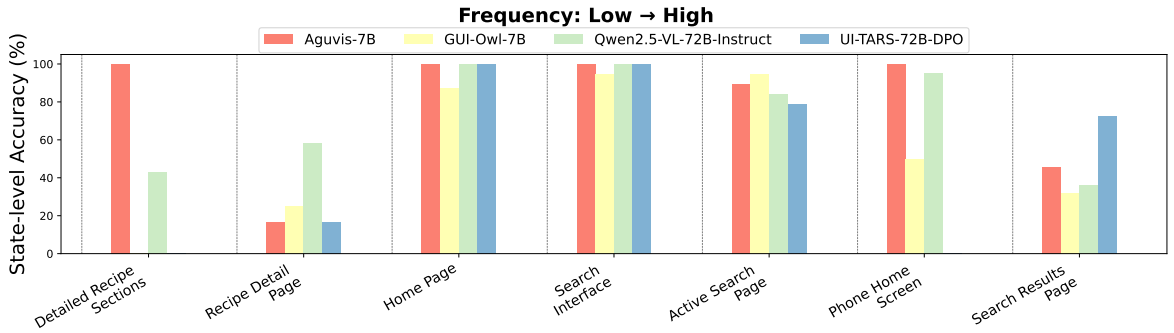

Figure 15: Transition-level bottleneck statistics on the Kitchen Stories application.

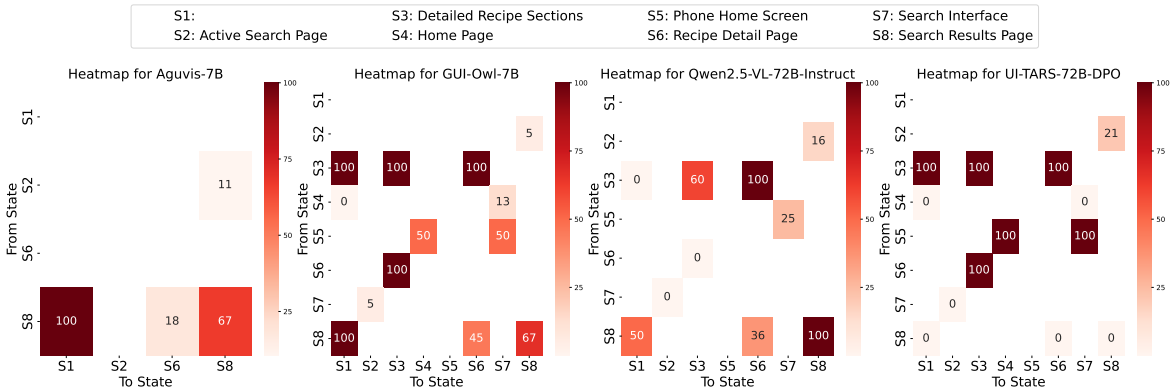

Figure 16: Transition-level error heatmap on the Kitchen Stories state-transition graph.

Prompt for Summarizing States and Transitions.

You are an assistant skilled in GUI operations. In a GUI environment, actions are taken according to instructions, and each action transitions from the current page to a new page. When performing different tasks in an app, similar pages may be encountered; these pages may display different content depending on the task, but the available operations on these pages remain the same. For example, on a product guide page, the items to be purchased may differ, but the operations (such as clicking on products) are the same.
I will provide you with several different task trajectories, including the pages visited during the process and the actions taken. I need you to analyze these trajectories and identify which pages are similar across different tasks.
Finally, please abstract the pages encountered in these trajectories into a state transition diagram, where states represent pages and edges represent actions.
The specific information I will provide is as follows: First, I will describe the trajectory with "Trajectory n: <Goal>", then I will list the pages visited from opening the app, along with the actions and corresponding page transitions, for example:
Action 1: Open app
Arrived page: <page content>
Action 2: ...
Please analyze the similarities between pages in these trajectories and construct a state transition diagram as described.
Task goal: <Gaol>
<Trajectory Execution History>
Here are some useful guidelines you need to follow:
- Pay attention to changes in the screen; significant changes often indicate arrival at a new state.
- The number of states is generally less than or equal to the number of screens.
- Whenever possible, environments with significant screen changes should be divided into two separate states.
Your answer should look like:
Analysis:...
Visualization of State Transition Diagram:...

Figure 17: Prompt for Summarizing States and Transitions.

Prompt for Aggregating into A Unified Graph.

You are an assistant skilled in GUI operations. In GUI environments, we perform actions based on instructions, and each action transitions the interface from the current screen to a new one. When executing different tasks in an app, we often pass through similar screens; these screens may display different content depending on the task, but the available operations are often the same. For example, on a product guide page, the items to shop for may differ, but the operation is always to click a product.
I will provide you with the visualized state transition information of multiple task trajectories, including the states passed through during execution and the actions performed. Your job is to analyze which states in these trajectories are similar. Finally, please consolidate these visualized state transition trajectories into a single state transition diagram.
Here is the specific information I will provide:
First, I will tell you "Trajectory n: <Goal>", then I will provide the visualized trajectory information for that task.
Please analyze the provided trajectories, identify similar states based on their functions and available operations (ignoring replaceable content), and consolidate all trajectories into a unified state transition diagram.
Task goal 1: <Gaol>
<Visualization of State Transition Trajectory>
Task goal 2: <Gaol>
<Visualization of State Transition Trajectory>
Here are some useful guidelines you need to follow:
- When a state extends outward, it often has multiple edges, each leading to different states. Be sure to retain all possible edges.
- Some actions may only appear in a few trajectories; make sure to include these actions.
- The initial state of a state transition diagram is usually the Home Screen, which is reached by performing the "Open App" action to enter the app's main page.
- The number of states should be sufficient to cover all possible situations that may occur in the app.
- An edge typically represents a single action. If there are multiple actions involved in the transition between two states, please represent each action with a separate edge.
- In the provided multiple trajectories, states may fall into two categories: states that are similar across multiple trajectories, and states that exist only in a single trajectory. Please merge the similar states across multiple trajectories, while retaining those states that exist only in a single trajectory. Ensure that the final state transition diagram includes all the states and edges from each provided trajectory, covering all possible paths.
Your answer should look like:
Analysis of Similar States:
...
Unified State Transition Diagram:
...
Key Operations:
...

Figure 18: Prompt for Aggregating into A Unified Graph.

Prompt for Collecting Structured States and Edge Transitions.

You are an assistant skilled in GUI operations. In GUI environments, actions are performed according to instructions, and each action transitions the interface from the current screen to a new one. When executing different tasks in an app, you often pass through similar screens; these screens may display different content depending on the task, but the available operations are often the same. For example, on a product guide page, the items to shop for may differ, but the operation is always to click a product.
Now, based on the state transition information from related trajectories, we have merged them into a corresponding state transition diagram. Please collect the states, edges, and transition information from this diagram.
State information includes: the state name and the meaning represented by the state. e.g. {"name": <>, "meaning": <>}
Edge information includes: the action and its meaning. You can use dictionaries to store the information for states and edges respectively. e.g. {"action": <>, "meaning": <>}
Transition information: For each transition, describe which state the transition starts from, what actions are performed, and which state the transition leads to. Ensure that the states and edges mentioned in the transition information are consistent with those in the state and edge information. Use a string to represent each transition. e.g. State A –Action–> State B
Please output your result in JSON format:
{
"States": <A list of dictionaries, each containing information about a state>,
"Edges": <A list of dictionaries, each containing information about an edge>,
"Transitions": <A list of strings, each representing a single transition; multiple transitions should be represented as multiple strings>
}
<State Transition Graph Information>
Now output the result in the correct JSON format.

Figure 19: Prompt for Collecting Structured States and Edge Transitions.

Prompt for Assigning Final States and Transitions.

You are an assistant skilled in GUI operations. In GUI environments, actions are performed according to instructions, and each action transitions the interface from the current screen to a new one. When executing different tasks in an app, you often pass through similar screens; these screens may display different content depending on the task, but the available operations are often the same. For example, on a product guide page, the items to shop for may differ, but the operation is always to click a product.
We have merged the state transition information from related trajectories into a corresponding state transition diagram. I will provide you with the states, edges, and transition information from the state transition diagram. Additionally, I will provide you with the complete trajectory of a task. Based on the information in the state transition diagram, please convert the task into the corresponding state transition trajectory.
Here is the specific information I will provide:
First, I will tell you the trajectory task: "Trajectory: <Goal>", then I will provide the sequence of pages visited after opening the app, as well as the actions that triggered page transitions. For example:
Screen 1: <page content>
Action 1: Open app
Screen 2: <page content>
Action 2: ...
Your task is to analyze how to map each screen to the corresponding state and each action to the corresponding edge in the state transition diagram.
Please output your result in JSON format:
{
"Screen 1": <state>,
"Action 1": <edge>,
"Screen 2": <state>,
"Action 2": <edge>,
}
Task goal: <Gaol>
<Trajectory Execution History>
Here is diagram information:
States Information:
...
Edges Information:
...
Transition Information:
...
Your answer should look like:
Analysis:...
JSON Output:...

Figure 20: Prompt for Assigning Final States and Transitions.

Prompt for Simulating Task Execution Trajectories.

You are an assistant proficient in GUI operations, tasked with simulating the completion path of a task. I will provide you with the task objective and state transition diagram, and you will begin executing the task from the Home Screen page.
Task goal: <Gaol>
Unified State Transition Graph: <STG>

Figure 21: Prompt for Simulating Task Execution Trajectories.

Prompt for GUITAR Online Diagnosis.

You are an assistant proficient in GUI operations, tasked with simulating the completion path of a task. I will provide you with the task objective and state transition diagram, and you will begin executing the task from the Home Screen page.
Task goal: <Gaol>
Unified State Transition Graph: <STG>
<Simulating Path Response>
Below is the complete trajectory of the Agent executing the task, which resulted in failure. Compared to our simulated path, identify which subtasks failed during execution.
<Trajectory Execution History>

Figure 22: Prompt for GUITAR Online Diagnosis.

Prompt for Baseline Online Diagnosis.

Below is the complete trajectory of the Agent executing the task, which resulted in failure. Identify which subtasks failed during execution.
Task goal: <Gaol>
<Trajectory Execution History>

Figure 23: Prompt for Baseline Online Diagnosis.