SAGE-GUI: Training-Free Structure-Aware Grounding Enhancement for GUI Agents

Anonymous ACL submission

Abstract

Reliable execution of graphical user interface agents requires accurate grounding of each action to the corresponding interface target. Existing methods commonly formulate GUI grounding as a screenshot-to-click localization task. However, the correct target is often determined by action semantics, element attributes, local structural context, and inter-element relations that extend beyond visual or textual similarity. We propose SAGE-GUI, a training-free and plug-and-play module for structure-aware GUI grounding. The proposed method constructs an action-oriented UI representation, resolves action targets over structured interface objects, and employs conservative visual arbitration for verification and recovery. In addition, we introduce StructSpot, a multi-level diagnostic benchmark designed to evaluate grounding performance under varying structural complexities, ranging from text matching to relational target resolution. Experiments on StructSpot and ScreenSpot demonstrate that SAGE-GUI consistently improves the performance of existing grounding backends without requiring model retraining, thereby highlighting the importance of structure-aware action-target resolution for GUI grounding.

1 Introduction

Effective execution in graphical user interfaces (GUIs) relies on accurately connecting each predicted action to its intended target element. Advances in large language and multimodal models have improved instruction understanding and action planning, yet GUI agents still struggle in realistic environments. Benchmarks like Mind2Web(Deng et al., 2023), Android in the Wild(Rawles et al., 2023), WebArena(Zhou et al., 2024), and OSWorld(Xie et al., 2024) show that failures often occur when a seemingly appropriate action is applied to the wrong interface element. Consequently, action grounding forms a crucial link between high-level action generation and low-level GUI execution. Even minor grounding errors in clicks, text inputs, or selections can disrupt subsequent execution.

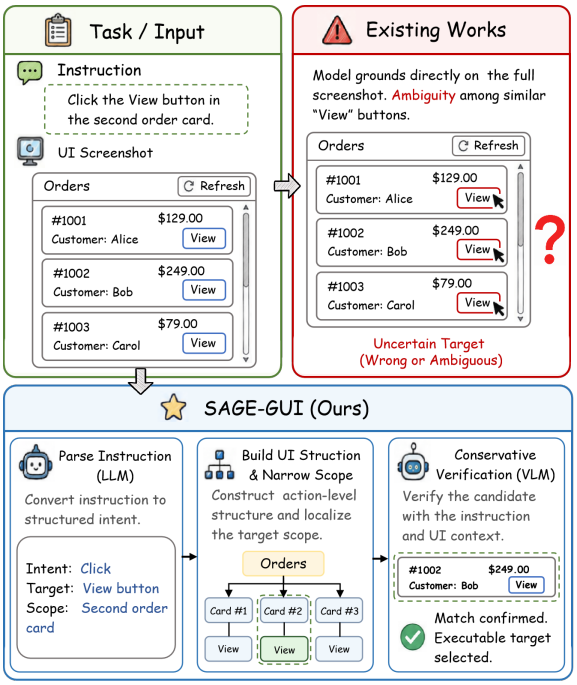

Figure 1: **Overview**. SAGE-GUI transforms screenshot-to-click grounding into structure-aware target resolution by parsing instructions, narrowing the UI scope, and verifying the executable target.

A growing body of research has examined how GUI agents perceive interfaces and identify action targets. Agent systems like WebAgent(Gur et al., 2024), OpenWebAgent(Iong et al., 2024), NNetNav(Murty et al., 2024), and AgentOccam(Yang et al., 2025a) use structured observations, including HTML, Document Object Model (DOM), or accessibility tree information, to expose interface elements and viable actions to language models. Grounding models, including SeeClick(Cheng et al., 2024), CogAgent(Hong et al., 2024), Ferret-UI(You et al., 2024), and ShowUI(Lin et al., 2025),

associate action instructions with visual targets in screenshots. Additionally, screen parsing techniques like OmniParser(Lu et al., 2024) provide element-level representations beyond raw pixel data.

However, existing grounding methods mainly treat target selection as screenshot-driven localization. Given a screenshot and an action instruction, the model predicts a click point, bounding box, or candidate element, while OCR results, layout information, element boxes, or parser outputs serve as auxiliary context or candidate sources(Xie et al., 2026). Although effective when the target is visually or textually prominent, this approach becomes less reliable when the correct element must be identified through local scopes, hierarchical relationships, repeated components, or contextual connections. This limitation has been highlighted by diagnostic grounding benchmarks and UI-decomposition studies(Jandial et al., 2026; Wang et al., 2026). In these cases, the target is not merely the region most closely resembling the instruction, but the executable interface object jointly determined by action semantics, element attributes, and structural context. Consequently, we argue that GUI grounding should be framed not merely as screenshot-driven localization, but as structure-aware action-target resolution across interfaces.

Motivated by this perspective, we propose SAGE-GUI, a training-free and plug-and-play module for structure-aware GUI grounding. Inserted between action generation and interface execution, it resolves action intents over structured UI objects and uses visual evidence for conservative verification. This design shifts GUI grounding from direct screenshot-to-target prediction toward structured target resolution while remaining compatible with existing grounding backends without retraining. We further introduce StructSpot, a multi-level diagnostic benchmark evaluating grounding under different structural complexities, from direct element matching to relation-dependent target resolution. It complements recent studies on referring expressions, hierarchical grounding, UI decomposition, and robustness under perturbations(Zhou et al., 2025; Xie et al., 2026; Liu et al., 2025). Experiments on StructSpot and ScreenSpot(Cheng et al., 2024; Wu et al., 2025b; Li et al., 2025) show that the proposed module improves existing grounding backends without retraining, suggesting that structure-aware target resolution usefully complements screenshot-based GUI localization.

The main contributions of this paper are summarized as follows:

- We propose **SAGE-GUI**, a training-free and plug-and-play module that enhances GUI grounding through action-oriented UI representation, structure-aware target resolution, and evidence-driven visual arbitration.
- We construct **StructSpot**, a multi-level diagnostic benchmark with structured UI annotations and action instructions stratified by structural complexity, enabling fine-grained evaluation of structure-aware GUI grounding.
- We provide a comprehensive empirical analysis of SAGE-GUI, validating its plug-and-play effectiveness across in-domain and standard GUI grounding benchmarks.

2 Related Works

2.1 GUI Agent

Recent research on GUI agents has increasingly moved away from relying on structured interfaces, such as DOMs and accessibility trees. Instead, it is focusing on visual agents that operate directly from screenshots. Methods like UI-TARS(Qin et al., 2025), Aguvis(Xu et al., 2024), and Aria-UI(Yang et al., 2025b) enhance interface operation and target localization across web, desktop, and mobile environments by leveraging large-scale GUI data, unified action spaces, and vision-language-action modeling. Additionally, SE-GUI(Yuan et al., 2026) and InfiGUI-G1(Liu et al., 2026) enhance grounding and execution in complex interfaces with reinforcement learning, self-evolution, and preference optimization. These studies demonstrate that large-scale GUI supervision and task-specific optimization can significantly improve the performance of visual GUI agents. Instead of training a stronger end-to-end visual agent, our approach improves the intermediate grounding stage that maps generated actions to executable interface objects.

2.2 Training-Free Grounding

Training-free GUI grounding has been studied as a way to improve target localization without additional supervised training or parameter updates. TAG(Xu et al., 2025) leverages attention patterns of pretrained multimodal models for tuning-free grounding. DiMo-GUI(Wu et al., 2025a) combines modality decoupling with dynamic local fo-

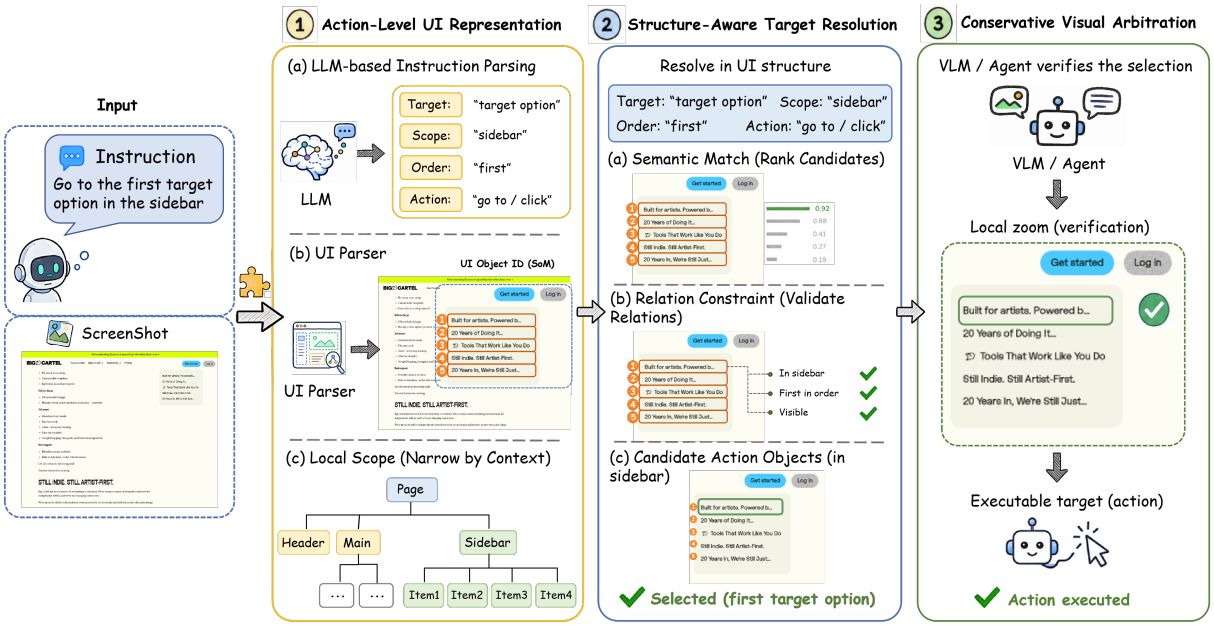

Figure 2: **Pipeline of SAGE-GUI**. SAGE-GUI enhances GUI agent grounding through three structure-aware steps: (1) Action-level UI Representation: It groups fragmented parser outputs into referable UI objects with SoM identifiers. (2) Structure-aware Target Resolution: It parses the action instruction into structured constraints and resolves the target using semantic, scope, and relational evidence. (3) Conservative Visual Arbitration: It uses local verification to confirm or correct the structural candidate before mapping it to an executable UI element.

cusing to refine predictions at inference time. RegionFocus(Luo et al., 2025) applies visual test-time scaling to reduce background interference through dynamic zoom-in. GUI-Actor(Wu et al., 2026) addresses spatial-semantic misalignment through coordinate-free grounding. Furthermore, V2P(Chen et al., 2025), ZoomClick(Jiang et al., 2025), and GUI-RC(Du et al., 2026) enhance localization by implementing attention calibration, zooming priors, and ensuring region consistency across sampled predictions. Overall, these methods primarily improve visual localization through strategies such as attention adjustment, cropping, zooming, coordinate reformulation, background suppression, spatial voting, or visual search. In contrast, our approach first reformulates the grounding decision space by resolving action targets across structured UI objects.

3 Method

3.1 Problem Formulation and Overview

We study one-step action grounding in GUI agent execution. Given an interface observation O and a one-step action intent a, the goal is to identify the executable target element e^* on which the action should operate. We represent the observation as $O = (I, R)$, where I denotes the screenshot and $R = \{r_i\}_{i=1}^{N}$ denotes a structured UI description. Each raw element r_i contains its bounding box, text, role, interactability, container relation, and local position.

In our implementation, R is obtained from a Set-of-Mark (SoM) style UI parser(Yang et al., 2023). The parser detects interactable regions and textual elements from the screenshot and renders a marked image with numbered bounding boxes. It also produces a mapping μ from mark identifiers to raw elements, enabling each visual mark to be traced back to its corresponding structured UI element. This representation aligns visual regions with structured UI elements without requiring additional human annotations. More generally, R can also be obtained from the DOM, accessibility tree, screen parser, or other interface observation APIs.

We focus on the grounding stage after an existing GUI agent has generated a one-step action intent, rather than on high-level planning or trajectory decision-making. Existing screenshot-to-click methods typically formulate grounding as $g(I, a) \rightarrow y$, where y is a click point, bounding box, or target element. Such methods primarily rely on vision-language similarity and can be confused by repeated texts, visually similar controls, and complex local structures.

In contrast, SAGE-GUI formulates GUI ground-

Algorithm 1 SAGE-GUI Inference

Input: Screenshot I, parsed UI R, action intent a, visual grounding backend g_{vis}
Output: Executable target element e^*
Render SoM image I^{som} and mark-element mapping μ from (I, R)
Construct action-level UI graph $G = (U, \mathcal{C}, \mathcal{R}, \pi)$ from R
Parse action intent a into structured action query $p = \langle t, \alpha, \rho, s, o, l \rangle$
Generate compatible candidates $C = \{u \in U \mid \text{compatible}(u, \alpha, \rho)\}$
Compute $S_{\text{struct}}(u \mid G, p)$ using activated constraints $\mathcal{K}(p)$
Select top candidates y_1, y_2 by S_{struct}
if visual arbitration is triggered **then**
    Determine local visual region Ω from structural candidates and scope
    Obtain visual evidence $z = g_{\text{vis}}(I^{\text{som}}, a; \Omega)$
    Map visual evidence to candidate $u_z = M(z, C, \mu)$
    Apply conservative arbitration to obtain y^*
else
    $y^* \leftarrow y_1$
end if
return $e^* = \pi(y^*)$

ing as structure-aware action target resolution:

$$f(I, R, a) \rightarrow e^* \tag{1}$$

As summarized in Algorithm1, the method proceeds in three stages. First, it converts parsed UI elements into an action-level UI graph, organizing fragmented parser outputs into referable interface objects with local scopes and structural relations. Second, it parses the action intent into a structured action query and resolves the target under semantic, role, scope, and relational constraints. Third, when the structural result is uncertain, it invokes a visual grounding backend on a SoM-marked local region and conservatively arbitrates between the structural candidate and visual evidence. The selected action-level target is mapped back to an executable raw element for interaction.

3.2 Action-Level UI Representation

A UI parser typically returns a set of low-level elements $R = \{r_i\}_{i=1}^{N}$ with fine-grained visual and structural attributes. However, these parser-level elements do not always correspond to executable interaction targets. For instance, a clickable button may be split into text, icon, and background regions, while a list item or table row may contain multiple structural nodes. Directly grounding actions over such fragmented elements can lead to duplicated candidates, target shifts, and scope errors.

To bridge this granularity gap, we abstract low-level elements into action-level UI objects $U = \{u_j\}_{j=1}^{M}$. Each object u_j represents an interface unit that can be referred to by an action and mapped back to executable raw elements:

$$u_j = \langle B_j, T_j, \tau_j, S_j, P_j, M_j, \pi_j \rangle \tag{2}$$

Here, B_j is the aggregated bounding box, T_j is the aggregated text or semantic description, τ_j is the object role or type, S_j denotes its scope or local region, and P_j denotes its position within the local region. $M_j \subseteq R$ is the set of low-level member elements contained in the object, and π_j records the executable mapping from the action-level object to its member raw elements. This mapping is used to select the actual executable element after an action-level target is determined.

Action-level objects are constructed by merging raw elements according to containment, spatial proximity, label-control association, clickable-region ownership, and repeated structures. For example, a label and its adjacent input box can form an input object, an icon and nearby text can form a button, and sub-elements within a list item can be grouped into the same local object. This abstraction shifts the grounding space from fragmented parser outputs to UI units that better align with real user actions.

Based on these objects, we builds a UI structure graph:

$$G = (U, \mathcal{C}, \mathcal{R}, \pi) \tag{3}$$

where $\mathcal{C}$ denotes containers or local regions, $\mathcal{R}$ denotes structural relations among action-level objects, and $\pi = \{\pi_j\}_{j=1}^{M}$ denotes mappings to raw elements. The relations in $\mathcal{R}$ include containment, group membership, local ordering, spatial neighborhood, row/column alignment, and repeated-group relations. This graph provides the structural context needed for grounding, allowing target resolution to check whether a candidate lies in the correct region, follows the expected local order, and satisfies the relational constraints implied by the action.

3.3 Structure-Aware Target Resolution

Given a one-step action intent a, we first convert it into a structured action query:

$$p = \langle t, \alpha, \rho, s, o, l \rangle \tag{4}$$

Here, t represents the target semantics; α, ρ, and s specify the action type, expected target role, and target scope; o denotes a reference object or anchor; and l encodes local selection constraints such as order, spatial relation, or state. Simple actions may activate only semantic and role constraints, while structurally complex actions additionally involve scope, anchor, or local relation constraints. The query can be produced by fixed rules or a frozen language model, which is used to generate executable constraints rather than directly select the final target.

Given the UI structure graph G and query p, target resolution is performed over action-level objects U. We first construct a candidate set:

$$C = \{u_j \in U \mid \mathrm{compatible}(u_j, \alpha, \rho)\} \tag{5}$$

where $\mathrm{compatible}(\cdot)$ checks whether the object type and interactability match the action type and expected role. For example, `type` actions prefer editable objects, while `click` actions prefer clickable objects such as buttons, links, and options.

Each candidate $u \in C$ is then scored by its structural consistency with the query:

$$S_{\mathrm{struct}}(u \mid G, p) = \frac{\sum_{k \in \mathcal{K}(p)} \lambda_k s_k(u, G, p)}{\sum_{k \in \mathcal{K}(p)} \lambda_k} \tag{6}$$

Here, $\mathcal{K}(p)$ denotes the constraints activated by query p, s_k and λ_k denote the corresponding score and fixed weight. The constraints cover semantic matching, role compatibility, scope consistency, local order, anchor relation, state, and interactability. Only activated constraints are evaluated, and all rules and weights are fixed across experiments without task-specific training.

The resolution path depends on the structure of query p. Direct target actions mainly rely on semantic and role cues; repeated components emphasize local scope and within-group order; group-level operations first identify the relevant row, card, or list item; and anchor-related actions first locate the reference object. Target selection is then constrained by containment, neighborhood, alignment, or relative-position relations, preventing structurally dependent actions from being reduced to global visual similarity matching.

The action-level target is selected as:

$$y = \arg\max_{u \in C} S_{\mathrm{struct}}(u \mid G, p) \tag{7}$$

The selected object y is finally mapped back to a raw executable element for interaction and element-level evaluation $e_{\mathrm{struct}} = \pi(y)$. When y contains multiple raw elements, the executable element is selected according to action type, interactability, member boxes, and local position.

3.4 Conservative Visual Arbitration

Structure-aware target resolution provides an explicit mechanism for modeling scope, local order, and relational constraints. In practical GUI environments, however, the structured observation itself may be noisy due to parser errors, bounding-box shifts, incomplete candidate extraction, or ambiguous mappings from action-level objects to raw elements. We therefore introduce visual evidence as a complementary signal for verification and recovery under uncertainty.

Visual arbitration is triggered only under runtime uncertainty. Let y_1 and y_2 denote the candidates with the highest and second-highest structural scores. Arbitration is invoked when the top-2 margin is small, the candidate set is large, the raw-element mapping is ambiguous, or the top structural candidate lacks sufficient textual or role evidence. All thresholds are fixed during experiments and do not rely on target annotations or evaluation results.

Instead of performing unconstrained full-screen search, the method derives a local visual region Ω from the structural context, such as the target scope, candidate cluster, anchor neighborhood, or pop-up region. The visual backend is applied within Ω to verify structural candidates, recover plausible alternatives, or refine the mapping from action-level objects to raw executable elements.

Given the SoM image I^{som}, action intent a, candidate set C, and local region Ω, the visual backend produces evidence $z = g_{\mathrm{vis}}(I^{\mathrm{som}}, a, \Omega)$, where z can be a mark identifier, click point, target box, or candidate region. This evidence is mapped back to an action-level candidate by $M(z, C, \mu) \rightarrow u_z$. Mark identifiers are resolved through μ, while points or boxes are matched using spatial overlap, center distance, interactability, and member-element relations. The evidence is used only when

it can be reliably aligned with a structural candidate or its member raw elements.

The final decision follows a conservative rule. The top structural candidate y_1 is retained when it is supported by visual evidence. An alternative candidate $u_z \in C$ is accepted only if it is both visually supported and structurally valid; otherwise, the visual prediction is rejected and the system falls back to y_1. If arbitration is not triggered, the final action-level target is directly set to $y^* = y_1$, and the executable element is obtained as $e^* = \pi(y^*)$.

The arbitration process uses only runtime-observable signals, such as structural score margins, visual confidence, text matching, scope consistency, and mapping reliability. It does not use instruction levels, target annotations, or evaluation results. Since the module operates only at the grounding stage without updating the upstream agent, UI parser, or visual backend, it can be used as an inference-time plugin for different GUI agents and grounding backends.

4 Experiments

4.1 Experimental Setup

4.1.1 Benchmarks

We evaluate GUI grounding capability on StructSpot and ScreenSpot. StructSpot is a structure-aware diagnostic benchmark constructed in this paper, containing 784 real-world web interfaces and 13,875 deduplicated action instructions. Each sample consists of an interface screenshot, a structured UI description, a natural language action instruction, and an annotated target raw element. Unlike coordinate-based evaluation, StructSpot anchors each target to an underlying interface element, directly measuring whether the model selects the correct executable object. StructSpot is organized into four levels, L0-L3, which correspond to direct text matching, semantic generalization, local structural localization, and relational action-target localization. Unless otherwise specified, all main experiments are conducted on the full benchmark.

We further evaluate on ScreenSpot, a standard GUI grounding benchmark covering web, desktop, and mobile scenarios, to examine whether the proposed module generalizes as a training-free plug-in beyond our diagnostic benchmark.

4.1.2 Implementation Details

We compare the proposed module with a frozen grounding baseline built on the same visual grounding backend. The baseline predicts the target region or element directly from the screenshot and action instruction, without action-level UI abstraction, structure-aware target resolution, or conservative visual-evidence arbitration. The proposed module is applied only at inference time and does not update the parameters of the underlying model, UI parser, or visual backend. Both settings use the same screenshots, parser outputs, and visual grounding backend to isolate the effect of structure-aware grounding. All experiments are conducted with deterministic inference on eight NVIDIA GeForce RTX 4090 GPUs, using Python 3.12, PyTorch 2.9.1, and CUDA 12.8.

To ensure consistent evaluation across different output formats, we convert each prediction into a point. If a method outputs a mark ID or bounding box, we use the center of the corresponding region as the predicted point, and count it as correct if it falls within the ground-truth bounding box.

4.1.3 Metrics

On StructSpot, the primary metric is element-level exact accuracy (EA):

$$\mathrm{EA} = \frac{1}{N} \sum_{i=1}^{N} I(\hat{e}_i = e_i^*) \tag{8}$$

where $\hat{e}_i$ denotes the predicted raw element, e_i^* denotes the annotated target element.

We report EA@L0, EA@L1, EA@L2, and EA@L3, and further report Macro EA and Micro EA:

$$\text{Macro EA} = \frac{1}{4} \sum_{\ell=0}^{3} \mathrm{EA@L}\ell \tag{9}$$

$$\text{Micro EA} = \frac{\sum_{\ell=0}^{3} C_\ell}{\sum_{\ell=0}^{3} N_\ell} \tag{10}$$

where N_ℓ and C_ℓ denote the number of samples and correctly grounded samples at level $\mathrm{L}\ell$, respectively. Macro EA measures balanced performance across different levels of structural complexity, while Micro EA measures average accuracy under the overall sample distribution.

4.2 Evaluation on Grounding Ability

4.2.1 Results on StructSpot

We first evaluate the proposed module on StructSpot with different grounding backends. As shown in Tab.1, across all evaluated backends, the

Backend	Base Model	Method	Macro EA	Micro EA	EA@L0	EA@L1	EA@L2	EA@L3
SeeClick (Cheng et al., 2024)	SeeClick-9.6B	Base	27.66	35.60	65.31	34.38	5.43	5.51
		+SAGE-GUI	50.26	60.45	94.38	66.16	24.14	16.36
		Δ	+22.60	+24.85	+29.07	+31.78	+18.71	+10.85
ShowUI (Lin et al., 2025)	ShowUI-2B	Base	40.73	49.21	75.54	56.76	21.43	9.19
		+SAGE-GUI	51.04	61.34	92.38	73.51	24.86	13.42
		Δ	+10.31	+12.13	+16.84	+16.75	+3.43	+4.23
CogAgent (Hong et al., 2024)	CogAgent-9B	Base	44.46	52.75	78.23	61.51	23.57	14.52
		+SAGE-GUI	54.92	64.43	**94.85**	72.00	31.14	21.69
		Δ	+10.46	+11.68	+16.62	+10.49	+7.57	+7.17
UGround (Gou et al., 2025)	UGround-V1-7B	Base	52.50	62.15	90.54	75.14	27.43	16.91
		+SAGE-GUI	55.23	64.92	94.46	**75.57**	30.86	20.04
		Δ	+2.73	+2.77	+3.92	+0.43	+3.43	+3.13
OmniParser (Lu et al., 2024)	Qwen3.5-35B-A3B	Base	51.85	60.28	86.38	69.30	29.29	22.43
		+SAGE-GUI	**56.52**	**65.64**	94.15	74.27	**33.57**	**24.08**
		Δ	+4.67	+5.36	+7.77	+4.97	+4.28	+1.65

Table 1: Grounding results on StructSpot. All values are reported as percentages. Δ denotes the absolute change of +SAGE-GUI over the corresponding Base backend. The best result in each column is bolded.

enhanced setting improves both Macro EA and Micro EA, indicating that the improvement is not specific to a particular visual grounding model. Since the underlying backends are kept frozen, these gains suggest that reorganizing the grounding process around structured UI objects can improve element selection beyond direct screenshot-to-target prediction.

The improvements are larger on weaker backends and smaller but consistent on stronger ones. For Macro EA, the enhanced setting brings gains of 22.60, 10.31, 10.46, 2.73, and 4.67 points on SeeClick, ShowUI, CogAgent, UGround, and OmniParser, respectively. This pattern shows that action-level UI objects and structural constraints are especially useful when the base model lacks reliable element-level resolution, but they also complement stronger grounding models.

The level-wise results show that the main remaining bottleneck lies in structurally complex grounding. Improvements are more pronounced on L0 and L1, where targets are often connected to visible text or semantic descriptions. L2 and L3 remain harder because target selection depends more on local scopes, repeated components, anchors, and relational context. Thus, structured UI information helps narrow the grounding space, but relation-dependent target resolution remains challenging and requires stronger cross-element reasoning. Qualitative examples in Fig.3 further illustrate these trends by comparing the original grounding backend with the enhanced setting.

4.2.2 Results on ScreenSpot

We further evaluate the proposed module on ScreenSpot to assess external generalization. Unlike StructSpot, ScreenSpot covers diverse web, desktop, and mobile GUI scenarios without structural-complexity levels, providing a complementary benchmark to our diagnostic setting.

As shown in Fig.4, each tested backend achieves higher accuracy after being enhanced with the proposed module. The gain is most pronounced on SeeClick, while stronger backends such as CogAgent, ShowUI, and OmniParser also show consistent improvements. These results indicate that the proposed module is not specific to StructSpot annotations or level definitions, and can transfer to a standard GUI grounding benchmark as a training-free enhancement that complements screenshot-based localization with structured target resolution.

4.3 Ablation Study

We conduct ablation experiments on StructSpot to assess the contribution of each component. All variants use OmniParser for SoM-style parsing and Qwen3.5-35B-A3B as the frozen base language model. As shown in Tab.2, Base is the basic grounding pipeline; Unit adds action-oriented UI representation; Intent further adds structured action queries and structure-aware target resolution; Visual adds visual-evidence verification and recovery; and Full denotes the complete method.

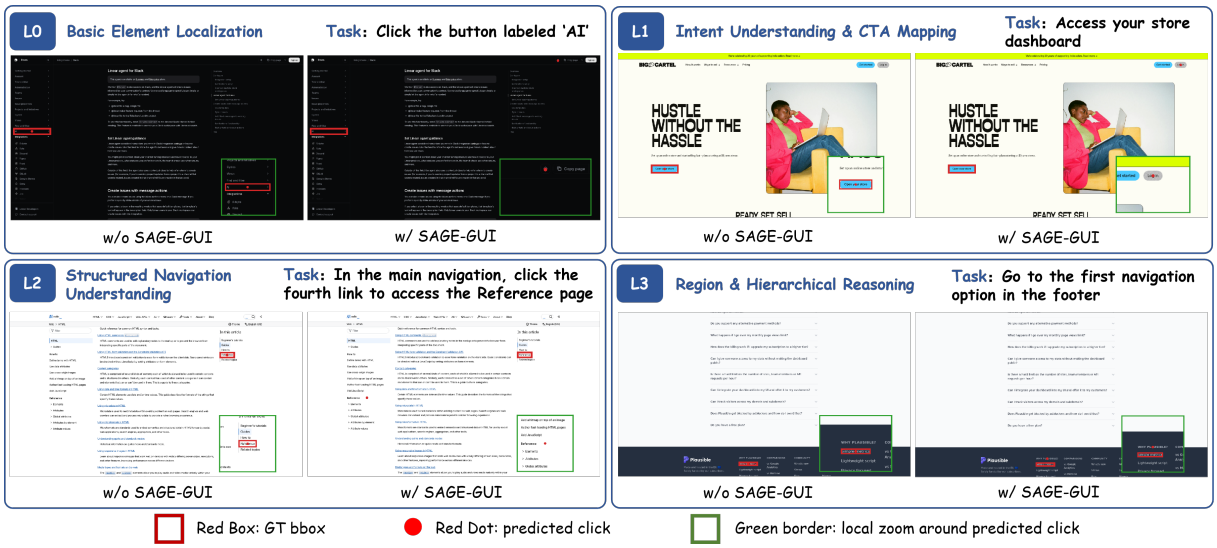

Figure 3: **Qualitative comparisons on StructSpot.** The original grounding backends often fail by selecting visually similar but structurally incorrect regions, while adding SAGE-GUI enables accurate target grounding across different task difficulty levels (L0-L3).

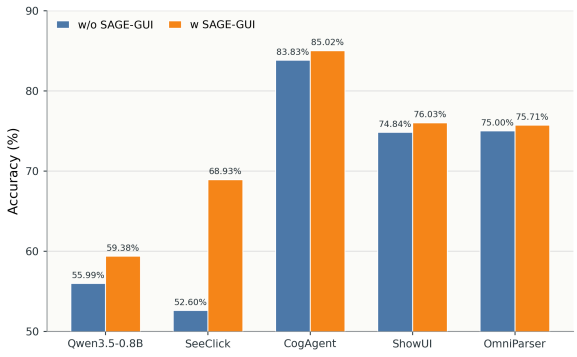

Figure 4: Accuracy comparison on ScreenSpot. Blue bars indicate results without SAGE-GUI, while orange bars indicate results with SAGE-GUI.

Method	Macro	Micro	L0	L1	L2	L3
Base	46.61	41.87	75.80	57.42	42.33	10.91
+Unit	51.20	44.91	82.56	73.22	39.66	9.42
+Intent	54.16	47.72	83.81	75.62	46.53	10.69
+Visual	56.83	51.18	90.48	68.16	55.57	13.11
Full	**59.68**	**53.33**	**91.01**	**77.26**	**56.37**	**14.09**

Table 2: Ablation results on StructSpot. All values are reported as percentages. Macro and Micro denote Macro EA and Micro EA, respectively. L0-L3 denote EA at different structural complexity levels. +Unit, +Intent, and +Visual indicate adding the corresponding module to Base individually.

Full achieves the best performance, improving Macro EA from 46.61 to 59.68 and Micro EA from 41.87 to 53.33. The ablation results show that the components contribute in different ways. +Unit mainly improves L0 and L1, indicating that action-level UI objects help align simple instructions with executable elements, but representation alone does not resolve harder structural cases. +Intent improves Macro EA to 54.16 and raises L2 from 39.66 to 46.53, showing the value of explicit scope and relational constraints. +Visual further improves L0 and L2, suggesting that visual evidence helps handle parser noise and uncertain mappings. These consistent gains show that the three components are complementary and jointly improve element-level grounding accuracy.

5 Conclusion

We present SAGE-GUI, a training-free and plug-and-play framework designed for structure-aware GUI grounding. It incorporates action-level UI representation, structure-aware target resolution, and conservative visual arbitration, which address the mismatch between fragmented parser outputs, screenshot-level localization, and executable interface targets. We further introduce StructSpot, a multi-level diagnostic benchmark for evaluating GUI grounding under different structural complexities. By resolving action intents over structured UI objects and using visual evidence for verification and recovery, it improves grounding performance, suggesting a structure-aware perspective for GUI grounding in which executable action targets are resolved within structured interface contexts rather than merely localized as visually salient regions.

Limitations

This work focuses on one-step action grounding, assuming that a GUI agent has already generated an action intent. It does not explicitly account for long-term execution feedback, where an incorrect action taken earlier could influence later observations, action intents, and recovery decisions. Expanding structure-aware grounding to include multi-step agent workflows, along with state tracking and execution-level feedback, represents a significant area for future research.

Another limitation is the reliance on parsed UI observations. Although the approach is not restricted to a specific parser, its effectiveness may be impacted by parser inaccuracies, incomplete element extraction, or incorrect element relationships. In our experiments, we used a SoM-style parser for observations. Future research could explore using DOM, accessibility-tree, or platform-specific interfaces to enhance robustness and applicability across different GUI environments.

References

Jikai Chen, Long Chen, Dong Wang, Leilei Gan, Chenyi Zhuang, and Jinjie Gu. 2025. V2p: From background suppression to center peaking for robust gui grounding task. *arXiv preprint arXiv:2508.13634*.

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.

Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Sam Stevens, Boshi Wang, Huan Sun, and Yu Su. 2023. Mind2web: Towards a generalist agent for the web. *Advances in Neural Information Processing Systems*, 36:28091–28114.

Yong Du, Yuchen Yan, Fei Tang, Zhengxi Lu, Chang Zong, Weiming Lu, Shengpei Jiang, and Yongliang Shen. 2026. Test-time reinforcement learning for gui grounding via region consistency. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pages 30593–30601.

Boyu Gou, Demi Ruohan Wang, Boyuan Zheng, Yanan Xie, Cheng Chang, Yiheng Shu, Huan Sun, and Yu Su. 2025. Navigating the digital world as humans do: Universal visual grounding for gui agents. In *International Conference on Learning Representations*, volume 2025, pages 30851–30883.

Izzeddin Gur, Hiroki Furuta, Austin Huang, Mustafa Safdari, Yutaka Matsuo, Douglas Eck, and Aleksandra Faust. 2024. A real-world webagent with planning, long context understanding, and program synthesis. In *International Conference on Learning Representations*, volume 2024, pages 52690–52717.

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.

Iat Long Iong, Xiao Liu, Yuxuan Chen, Hanyu Lai, Shuntian Yao, Pengbo Shen, Hao Yu, Yuxiao Dong, and Jie Tang. 2024. Openwebagent: An open toolkit to enable web agents on large language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*, pages 72–81.

Surgan Jandial, Yinheng Li, Justin Wagle, and Kazuhito Koishida. 2026. Do gui grounders truly understand ui elements? In *Findings of the Association for Computational Linguistics: EACL 2026*, pages 2772–2785.

Zhiyuan Jiang, Shenghao Xie, Wenyi Li, Wenqiang Zu, Peihang Li, Jiahao Qiu, Siqi Pei, Lei Ma, Tiejun Huang, Mengdi Wang, and 1 others. 2025. Zoom in, click out: Unlocking and evaluating the potential of zooming for gui grounding. *arXiv preprint arXiv:2512.05941*.

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. In *Proceedings of the 33rd ACM International Conference on Multimedia*, pages 8778–8786.

Kevin Qinghong Lin, Linjie Li, Difei Gao, Zhengyuan Yang, Shiwei Wu, Zechen Bai, Stan Weixian Lei, Lijuan Wang, and Mike Zheng Shou. 2025. Showui: One vision-language-action model for gui visual agent. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 19498–19508.

Xinyi Liu, Xiaoyi Zhang, Ziyun Zhang, and Yan Lu. 2025. Ui-e2i-synth: Advancing gui grounding with large-scale instruction synthesis. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 15668–15684.

Yuhang Liu, Zeyu Liu, Shuanghe Zhu, Pengxiang Li, Congkai Xie, Jiasheng Wang, Xueyu Hu, Xiaotian Han, Jianbo Yuan, Xinyao Wang, and 1 others. 2026. Infigui-g1: Advancing gui grounding with adaptive exploration policy optimization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pages 32267–32275.

Yadong Lu, Jianwei Yang, Yelong Shen, and Ahmed Awadallah. 2024. Omniparser for pure vision based gui agent. *arXiv preprint arXiv:2408.00203*.

Tiange Luo, Lajanugen Logeswaran, Justin Johnson, and Honglak Lee. 2025. Visual test-time scaling for gui agent grounding. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 19989–19998.

Shikhar Murty, Hao Zhu, Dzmitry Bahdanau, and Christopher D Manning. 2024. Nnetnav: Unsupervised learning of browser agents through environment interaction in the wild. *arXiv preprint arXiv:2410.02907*.

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, 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.

Yangyue Wang, Harshvardhan Sikka, Yash Mathur, Tony Zhou, Jinu Nyachhyon, and Pranav Guruprasad. 2026. Gui-perturbed: Domain randomization reveals systematic brittleness in gui grounding models. *arXiv preprint arXiv:2604.14262*.

Hang Wu, Hongkai Chen, Yujun Cai, Chang Liu, Qingwen Ye, Ming-Hsuan Yang, and Yiwei Wang. 2025a. Dimo-gui: Advancing test-time scaling in gui grounding via modality-aware visual reasoning. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 26257–26267.

Qianhui Wu, Kanzhi Cheng, Rui Yang, Chaoyun Zhang, Jianwei Yang, Huiqiang Jiang, Jian Mu, Baolin Peng, Bo Qiao, Reuben Tan, and 1 others. 2026. Gui-actor: Coordinate-free visual grounding for gui agents. *Advances in Neural Information Processing Systems*, 38:15101–15128.

Zhiyong Wu, Zhenyu Wu, Fangzhi Xu, Yian Wang, Qiushi Sun, Chengyou Jia, Kanzhi Cheng, Zichen Ding, Liheng Chen, Paul Pu Liang, and 1 others. 2025b. Os-atlas: Foundation action model for generalist gui agents. In *International Conference on Learning Representations*, volume 2025, pages 5090–5108.

Tianbao Xie, Jiaqi Deng, Xiaochuan Li, Junlin Yang, Haoyuan Wu, Jixuan Chen, Wenjing Hu, Xinyuan Wang, Yuhui Xu, Zekun Wang, and 1 others. 2026. Scaling computer-use grounding via user interface decomposition and synthesis. *Advances in Neural Information Processing Systems*, 38.

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.

Hai-Ming Xu, Qi Chen, Lei Wang, and Lingqiao Liu. 2025. Attention-driven gui grounding: Leveraging pretrained multimodal large language models without fine-tuning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pages 8851–8859.

Yiheng Xu, Zekun Wang, Junli Wang, Dunjie Lu, Tianbao Xie, Amrita Saha, Doyen Sahoo, Tao Yu, and Caiming Xiong. 2024. Aguvis: Unified pure vision agents for autonomous gui interaction. *arXiv preprint arXiv:2412.04454*.

Jianwei Yang, Hao Zhang, Feng Li, Xueyan Zou, Chunyuan Li, and Jianfeng Gao. 2023. Set-of-mark prompting unleashes extraordinary visual grounding in gpt-4v. *arXiv preprint arXiv:2310.11441*.

Ke Yang, Yao Liu, Sapana Chaudhary, Rasool Fakoor, Pratik A Chaudhari, George Karypis, and Huzefa Rangwala. 2025a. Agentoccam: A simple yet strong baseline for llm-based web agents. In *International Conference on Learning Representations*, volume 2025, pages 97533–97565.

Yuhao Yang, Yue Wang, Dongxu Li, Ziyang Luo, Bei Chen, Chao Huang, and Junnan Li. 2025b. Aria-ui: Visual grounding for gui instructions. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 22418–22433.

Keen You, Haotian Zhang, Eldon Schoop, Floris Weers, Amanda Swearngin, Jeffrey Nichols, Yinfei Yang, and Zhe Gan. 2024. Ferret-ui: Grounded mobile ui understanding with multimodal llms. In *European Conference on Computer Vision*, pages 240–255. Springer.

Xinbin Yuan, Jian Zhang, Kaixin Li, Zhuoxuan Cai, Lujian Yao, Jie Chen, Enguang Wang, Qibin Hou, Jinwei Chen, Peng-Tao Jiang, and 1 others. 2026. Se-gui: Enhancing visual grounding for gui agents via self-evolutionary reinforcement learning. *Advances in Neural Information Processing Systems*, 38:127658–127679.

Beitong Zhou, Zhexiao Huang, Yuan Guo, Zhangxuan Gu, Tianyu Xia, Zichen Luo, Fei Tang, Dehan Kong, Yanyi Shang, Suling Ou, and 1 others. 2025. Venusbench-gd: A comprehensive multi-platform gui benchmark for diverse grounding tasks. *arXiv preprint arXiv:2512.16501*.

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 *International Conference on Learning Representations*, volume 2024, pages 15585–15606.

A Benchmark Construction

A.1 StructSpot Construction Details

StructSpot evaluates whether GUI grounding systems can identify action targets with awareness

of interface structure. Rather than judging only whether a predicted click coordinate lies inside the target region, StructSpot links each action target to the executable interface element beneath it. This design allows the evaluation to directly measure whether a model selects the correct interactive object.

Each sample includes a screenshot, a structured UI description, a natural-language action instruction, and an annotated target raw element. Each instruction specifies a single executable step, so the benchmark focuses on grounding rather than high-level task planning. We remove duplicate action instructions and manually inspect ambiguous cases in which more than one target may be plausible. For samples involving repeated cards, tables, lists, forms, or pop-ups, the annotation also records the local scope, reference object, or structural relation needed to identify the target.

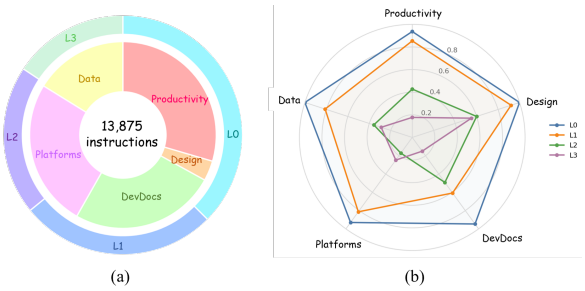

Figure 5: Overview of StructSpot. (a) Distribution of samples across task categories and structural complexity levels (L0–L3). (b) Grounding performance under different structural complexity levels, showing that accuracy decreases as structural reasoning becomes more demanding.

A.2 Structural Complexity Levels

StructSpot groups action instructions into four levels, L0-L3, based on the structural information needed to locate the correct target. L0 mainly captures direct text matching, L1 covers semantic generalization, L2 requires localization within a nearby structure, and L3 involves parsing relations between actions and targets. This stratification makes it possible to compare grounding systems under different structural demands.As shown in Fig. 5, StructSpot contains a balanced distribution of task categories and difficulty levels, while model performance consistently degrades as structural complexity increases from L0 to L3.

B SAGE-GUI Implementation Details

Our implementation of SAGE-GUI uses a SoM-style UI parser to produce structured UI observations. The parser combines interactive-region detection with OCR text extraction, yielding a structured element set R, a screenshot with numbered marks I^{som}, and a mapping μ from mark ids to raw elements. The method is not tied to this particular parser. When element bounding boxes, text, roles, interactability, and local structural information are available, R can also be obtained from the DOM, an accessibility tree, or other screen parsing interfaces.

The implementation first groups the low-level parser outputs into action-level UI objects. These low-level outputs may include text nodes, icon regions, clickable backgrounds, input boxes, links, and layout containers. Because such elements do not always match the objects that users intend to act on, the method aggregates related elements into interface objects that better support action reference. The aggregation uses containment relations, spatial proximity, label-control associations, clickable-region ownership, and repeated structures in lists, tables, and cards.

Each action-level object keeps its member raw elements and the mapping π to the underlying elements. This design allows structure-aware target parsing to be performed over action-level objects, after which the final prediction is mapped back to an executable or evaluable raw element. This procedure requires no additional training and does not alter the underlying raw element set. It only changes the granularity at which grounding is performed, making the candidate space closer to the interactive objects that users refer to in practice.

C Structured Action Query and Scoring

The action intent is parsed into a structured action query:

$$p = \langle t, \alpha, \rho, s, o, l \rangle. \tag{11}$$

Here, t denotes the target semantics, α the action type, ρ the expected target role, s the local scope, o the reference object or anchor, and l local selection constraints such as order, direction, state, or spatial relation.

The structured query determines which constraints are used during target parsing. For a candidate action-level object u, the structural matching

score is defined as

$$S_{\text{struct}}(u \mid G, p) = \frac{\sum_{k \in \mathcal{K}(p)} \lambda_k s_k(u, G, p)}{Z(p)},$$
$$Z(p) = \sum_{k \in \mathcal{K}(p)} \lambda_k. \tag{12}$$

Here, $\mathcal{K}(p)$ is the set of constraints activated by the current query, s_k is the matching score for the k-th constraint type, and λ_k is a fixed weight. The constraints cover semantic matching, role compatibility, scope consistency, local order, anchor relations, state conditions, and interaction availability. Constraints that are not activated by the current query are excluded from scoring, which prevents irrelevant penalties for simple actions. All rules and weights are fixed throughout the experiments, and no task annotations are used for training.

D Visual Evidence Arbitration

In this design, the visual model serves as an additional source of evidence after structural parsing, rather than as an independent final decision maker. Visual evidence arbitration is triggered only when structural parsing is uncertain. Typical cases include a small margin between the two highest structural scores, a large candidate set, an ambiguous mapping from action-level objects to raw elements, or a top-ranked structural candidate with weak text and role evidence.

When arbitration is triggered, the system does not perform full-screen visual search by default. Instead, it defines a local visual region Ω from the structural parsing result. This region may correspond to the target scope, a cluster of candidates, an area near an anchor, or a local interaction region such as a pop-up. The visual grounding backend then operates within the SoM-marked local region and returns a mark id, click point, target box, or candidate region.

The visual output is mapped back to an action-level object or raw element in the current UI representation. If the visual evidence supports a structural candidate, the reliability of that candidate is increased. If it supports a structurally valid but lower-ranked candidate, candidate recovery is allowed. If the visual output cannot be mapped stably to a candidate element, or if it conflicts with the scope and relation constraints, the visual result is rejected and the system falls back to the structural parsing result.

This arbitration mechanism uses only signals observable at runtime, including structural score margins, visual confidence, text matching, scope consistency, and mapping reliability. It does not use instruction levels, target labels, ground-truth bounding boxes, or evaluation results, so it does not introduce label leakage.

E Ablation Settings

The main experiments consider five ablation settings: Base, +Unit, +Intent, +Visual, and Full. These settings correspond to different conceptual modules of SAGE-GUI and help analyze how each component contributes to overall performance.

Base is the basic grounding pipeline without the enhancement mechanisms introduced by SAGE-GUI. +Unit adds action-level UI object representations to reduce fragmentation in parser outputs. +Intent adds structured action queries and structural constraint parsing, improving the modeling of action semantics, target roles, and local scopes. +Visual adds visual evidence arbitration for candidate verification, error recovery, and correction of raw-element mappings. Full denotes the complete SAGE-GUI system, which combines action-level UI representations, structure-aware target parsing, and visual evidence arbitration.

These ablations are intended mainly to clarify the problems addressed by different mechanisms, rather than to serve as fully independent single-factor experiments. Because the modules interact at runtime, some settings share the same parser, candidate generation procedure, or mapping logic.

F Training-Free Setting and Label Leakage

The proposed module is training-free and targets structure-aware grounding. The system does not train a new GUI agent, fine-tune a vision-language model, or learn a new policy network. External language and visual models are used only as replaceable backends for structured action-query parsing and visual evidence extraction. The core computation comes from action-level UI representations, structure-aware target parsing, and visual evidence arbitration, rather than from additional policy training.

The runtime prediction does not use target element labels, ground-truth bounding boxes, evaluation results, or instruction-level information. Instruction levels are used only for benchmark con-

struction, grouped statistics, and experimental analysis; they do not participate in model inference. The conservative fallback mechanism in the Full setting also relies only on runtime-observable evidence, including structural score margins, basic grounding predictions, visual evidence, text matching, and mapping reliability. As a result, the final system decision does not receive extra information from evaluation labels or difficulty annotations.

G Qualitative Case Analysis

We provide additional qualitative analyses to better understand both the strengths and remaining limitations of SAGE-GUI. The appendix includes representative failure patterns as well as successful correction cases that could not be fully presented in the main paper due to space constraints.

The first group corresponds to cases where the original grounding backend fails while SAGE-GUI successfully resolves the target. These examples usually involve scope drift, confusion among repeated components, or fragmented candidate regions. By organizing the screen into action-level UI objects and applying local structural constraints, SAGE-GUI narrows the grounding space to the correct structural scope and corrects errors made by direct screenshot-level grounding.

The second group highlights cases where visual evidence arbitration becomes important. When structural candidates have low confidence, parser outputs contain noise, or mappings between action-level objects and raw UI elements become unstable, local visual verification helps recover weakened candidates and refine the final grounding decision.

The third group presents remaining failure cases. These failures are often caused by abstract relational descriptions, underspecified intents, deeply nested structures, or incomplete UI parsing results. Such cases reveal the current limitations of structure-aware grounding and suggest that stronger cross-element relational reasoning is still required.

Fig. 3 presents representative successful correction examples across different difficulty levels. Compared with the original grounding backends, SAGE-GUI consistently shifts predictions from visually similar but structurally irrelevant regions to the correct target UI objects. The improvements are particularly visible in interfaces containing repeated elements, hierarchical layouts, or ambiguous local regions, where screenshot-level visual matching alone is insufficient for reliable target selection. These examples further demonstrate that introducing structured UI representations helps constrain the grounding process and improves robustness in complex GUI environments.