DRS-GUI: Dynamic Region Search for Training-Free GUI Grounding

Supplementary Material

1. Pseudo-code of Perceptual Actions

We first provide the pseudo-code of the three perceptual actions used in DRS-GUI, namely FOCUS, SCATTER, and SHIFT. These actions operate on parsed UI elements and are used by the MCTS-based planner to progressively refine, expand, or relocate the current region during dynamic region search.

Algorithm 1 Focus Perceptual Action

Input: GUI image I; current region R with bbox B_R; instruction T; UI elements inside R: $U_R = \{u_i = [b_i, d_i, i_i]\}$; similarity scores $\{s_i\}$; semantic top-ratio p; target shrink ratio ρ_{focus}.
Output: Focused region R_{focus}.

$U_{\text{top}} \leftarrow \text{TopKSemantic}(U_R, \{s_i\}, p)$
if $|U_{\text{top}}| = 0$ **then**
  return R
end if
$U_{\text{cluster}} \leftarrow \text{ClusterNoOutlier}(U_{\text{top}})$
if $|U_{\text{cluster}}| = 0$ **then**
  $U_{\text{cluster}} \leftarrow \{\arg\max_{u_i \in U_{\text{top}}} s_i\}$
end if
$U_{\text{focus}} \leftarrow \text{ShrinkWithin}(U_{\text{cluster}}, B_R, \rho_{\text{focus}})$
$R_{\text{focus}} \leftarrow \text{BBoxUnion}(U_{\text{focus}})$
return R_{focus}

2. Region Quality Reward Hyper-parameters

We further study how the three reward weights affect DRS-GUI by independently varying the coefficients of *Interaction Weighted Relevance* (α), *UI Coverage Consistency* (β), and *Semantic Concentration* (γ) in the composite reward $r(R, T)$ and reporting accuracy on the ScreenSpot-V2 Mobile, Desktop, and Web splits. As shown in Fig. 1, performance changes smoothly as each weight sweeps from 0.1 to 0.5. Accuracy consistently improves when increasing the contribution of interaction-weighted relevance (top row), is moderately sensitive to the coverage term (middle row), and shows a milder, peaked response to the semantic concentration weight (bottom row). Overall, the curves stay relatively flat around mid-range settings, suggesting that DRS-GUI is robust to moderate changes of α, β, and γ.

3. Efficiency and Robustness Analysis

We further analyze the computational overhead, component dependency, and general applicability of DRS-GUI. Since DRS-GUI introduces UI perception and MCTS-based region search before a single grounding call, it is important to examine both its efficiency and its robustness to different external components and base models.

Algorithm 2 Scatter Perceptual Action

Input: Full GUI image I; current region R with bbox B_R; global UI elements $U = \{u_i = [b_i, d_i, i_i]\}$; similarity scores $\{s_i\}$; semantic top-ratio p; ring scale λ_{ring}; max area scale $\kappa_{\max}$; max IoU threshold $\eta_{\max}$.
Output: Expanded region R_{scatter}.

$U_{\text{out}} \leftarrow \{u_i \in U \mid \text{Center}(b_i) \text{ is strictly outside } B_R\}$
if $|U_{\text{out}}| = 0$ **then**
  $R_{\text{scatter}} \leftarrow \text{ScaleAroundCenter}(B_R, \lambda_{\text{ring}})$
  return R_{scatter}
end if
$U_{\text{cand}} \leftarrow \text{TopKSemantic}(U_{\text{out}}, \{s_i\}, p)$
$U_{\text{exp}} \leftarrow \text{ExpandWithin}(U_{\text{cand}}, B_R, \kappa_{\max})$
if $|U_{\text{exp}}| > 0$ **then**
  $B_{\text{new}} \leftarrow \text{BBoxUnion}(\{B_R\} \cup U_{\text{exp}})$
  if $\text{Area}(B_{\text{new}}) > \text{Area}(B_R)$ **and** $\text{IoU}(B_{\text{new}}, B_R) \leq \eta_{\max}$ **then**
    $R_{\text{scatter}} \leftarrow B_{\text{new}}$
    return R_{scatter}
  end if
end if
$R_{\text{scatter}} \leftarrow \text{ScaleAroundCenter}(B_R, \lambda_{\text{ring}})$
return R_{scatter}

3.1. Inference overhead and rollout budget

Table 1 reports the effect of the rollout budget N on ScreenSpot-Pro accuracy and inference latency. DRS-GUI introduces additional overhead from UI perception and region planning, but still preserves a favorable accuracy–latency trade-off because the base MLLM is invoked only once on the selected region. As N increases, accuracy improves steadily from N=1 to N=8, while latency grows roughly linearly. At N=12, performance largely saturates, suggesting that a moderate rollout budget already captures most of the gains even on challenging high-resolution GUIs.

Table 1. Effect of rollout budget N on ScreenSpot-Pro accuracy and inference latency (s/sample).

Model	N	Dev	Cre	CAD	Sci	Office	OS	Avg	Lat.
Qwen2.5-VL-7B	–	26.1	24.0	13.0	31.1	45.2	23.5	26.8	12.8
+ *DRS-GUI*	1	30.1	27.0	19.2	33.5	48.7	28.1	30.6	13.6
+ *DRS-GUI*	4	35.1	35.2	29.9	38.2	54.3	35.7	37.6	20.0
+ *DRS-GUI*	8	**37.5**	**38.7**	34.1	**41.0**	**57.4**	**39.3**	**40.9**	27.1
+ *DRS-GUI*	12	36.8	38.7	**35.3**	40.2	56.5	38.3	40.5	36.8

3.2. Dependency on parser and embedding models

Table 2 evaluates DRS-GUI under different parser and embedding settings on ScreenSpot-V2. P1/P2 denote OmniParser v1/v2, where P1 is weaker and more prone to miss-

Algorithm 3 Shift Perceptual Action

Input: Full GUI image I; current region R with bbox B_R; global UI elements $U = \{u_i = [b_i, d_i, i_i]\}$; similarity scores $\{s_i\}$; semantic top-ratio p; min shift ratio ρ_{shift}; padding size δ_{pad}; contain threshold θ_{contain}; max IoU threshold η_{shift}.

Output: Shifted region R_{shift}.

$U_{\text{out}} \leftarrow \{u_i \in U \mid \text{Center}(b_i) \text{ is strictly outside } B_R\}$
if $|U_{\text{out}}| = 0$ **then**
  $R_{\text{shift}} \leftarrow \text{ScatterAction}(I, R, U, \{s_i\})$
  return R_{shift}
end if
$U_{\text{top}} \leftarrow \text{TopKSemantic}(U_{\text{out}}, \{s_i\}, p)$
$c_R \leftarrow \text{Center}(B_R)$
$D_R \leftarrow \text{DiagLength}(B_R)$
$d_{\min} \leftarrow \rho_{\text{shift}} \cdot D_R$
$U_{\text{far}} \leftarrow \{u_i \in U_{\text{top}} \mid \|\text{Center}(b_i) - c_R\|_2 \geq d_{\min}\}$
if $|U_{\text{far}}| = 0$ **then**
  $R_{\text{shift}} \leftarrow \text{ScatterAction}(I, R, U, \{s_i\})$
  return R_{shift}
end if
$\{G_d\} \leftarrow \text{GroupByDirection}(U_{\text{far}}, B_R)$
$U_{\text{shift}} \leftarrow \arg\max_{G_d} |G_d|$
$B^{(0)}_{\text{shift}} \leftarrow \text{BBoxUnion}(U_{\text{shift}})$
$B_{\text{shift}} \leftarrow \text{PadBox}(B^{(0)}_{\text{shift}}, \delta_{\text{pad}})$
if $\text{IoU}(B_R, B_{\text{shift}}) > \theta_{\text{contain}}$ **or** $\text{IoU}(B_{\text{shift}}, B_R) > \eta_{\text{shift}}$ **then**
  $R_{\text{shift}} \leftarrow \text{ScatterAction}(I, R, U, \{s_i\})$
  return R_{shift}
end if
$R_{\text{shift}} \leftarrow B_{\text{shift}}$
return R_{shift}

ing elements, and E1/E2 denote SBERT-NLI-large versus our instruction-guided embedding.

The results show two consistent trends. First, stronger UI parsing and better instruction-aware embedding improve region scoring and therefore improve final grounding accuracy. Second, DRS-GUI remains effective even under the weaker parser setting (P1+E2), indicating that the method is not overly brittle to parser misses. This is because the parser is only used to guide iterative region search rather than to directly determine the final prediction. The final coordinates are still predicted by the grounding model after multiple search steps, which helps mitigate errors introduced by imperfect external components.

Table 2. Parser and embedding ablations on ScreenSpot-V2. P1/P2 denote OmniParser v1/v2, and E1/E2 denote SBERT-NLI-large versus our instruction-guided embedding.

Method	Mobile		Desktop		Web		Avg.
	Text	Icon/Widget	Text	Icon/Widget	Text	Icon/Widget	
UGround-V1-7B	95.0	83.3	95.0	77.8	92.1	77.2	87.6
+ *P1+E1*	93.8	84.4	90.7	80.0	91.5	83.7	88.2
+ *P1+E2*	96.2	87.7	94.3	83.6	93.6	87.2	91.2
+ *P2+E2*	**96.6**	**88.2**	**94.9**	**84.3**	**94.0**	**87.7**	**91.8**

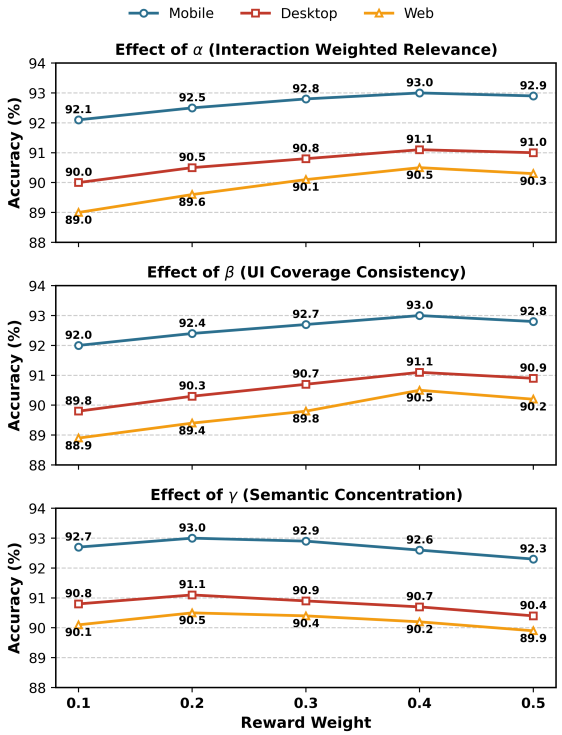

Figure 1. Sensitivity of DRS-GUI to reward weights α, β, and γ on the ScreenSpot-V2 Mobile, Desktop, and Web splits. Accuracy varies smoothly with each weight and peaks near mid-range settings, indicating robustness of the reward design.

3.3. Comparison with zoom-in methods and stronger agents

Table 3 compares DRS-GUI with representative zoom-in methods, recent GUI grounding approaches, and stronger GUI agents on ScreenSpot-Pro. Under the same backbone, DRS-GUI achieves the best overall performance among zoom-in methods, which supports our claim that MCTS-based dynamic perceptual planning is more robust than one-way, non-revisable zoom-in strategies. DRS-GUI also outperforms recent methods such as Aguvis-7B, despite requiring no model-specific training.

Moreover, the gains remain substantial when DRS-GUI is applied to stronger base agents such as UI-TARS-1.5-7B and Qwen3-VL-8B. This suggests that long-tail grounding failures still persist in high-resolution and element-dense GUIs, even for stronger models. As an inference-time plug-in, DRS-GUI improves grounding by narrowing the perceptual scope and filtering out redundant context before the final prediction stage.

4. GUI Domain-specific Prompts

Before encoding, we prepend a domain-specific prefix to the user instruction so that UI elements are embedded in an application-aware semantic space. For each (application a, platform p), the prompt follows a unified pattern: *"Repre-*

Table 3. Grounding accuracy comparison on ScreenSpot-Pro.

Model	Dev	Cre	CAD	Sci	Office	OS	Avg
Zoom-in methods							
Qwen2.5-VL-7B	26.1	24.0	13.0	31.1	45.2	23.5	26.8
+ *RegionFocus*	29.4	28.2	24.1	37.0	55.7	30.6	33.5
+ *UI-AGILE*	37.4	22.9	22.6	34.2	52.6	35.3	33.3
+ *DRS-GUI*	**37.5**	**38.7**	**34.1**	**41.0**	**57.4**	**39.3**	**40.9**
Stronger GUI agents							
UI-TARS-1.5-7B	38.5	39.3	22.2	53.1	68.7	33.2	42.1
+ *DRS-GUI*	**47.5**	**46.3**	**43.3**	**56.7**	**71.7**	**45.4**	**51.3**
Qwen3-VL-8B	52.8	49.1	49.0	56.7	75.2	50.5	55.0
+ *DRS-GUI*	**57.9**	**56.3**	**60.5**	**63.0**	**76.5**	**61.7**	**62.0**
Recent methods							
Aguvis-7B	16.1	21.4	13.8	34.6	34.3	19.4	22.9
GUI-R1-7B	27.8	26.1	19.6	36.6	51.3	30.7	31.3

sent this UI element for semantic matching with the user instruction in a $\mathrm{Desc}(a, p)$ *interface. Typical elements in this interface include:* $\mathcal{K}_{a,p}$*."* where $\mathrm{Desc}(a, p)$ is a short descriptor (e.g., "Photoshop Windows element") and $\mathcal{K}_{a,p}$ is a small set of UI-related keywords (e.g., layer panel, timeline, command palette). All concrete (application, platform, keyword) combinations used in our experiments are listed in Table 4. For screenshots that do not match any row, we fall back to application-only, platform-only, or a generic GUI prompt while keeping the same construction scheme.

5. More qualitative results

We provide additional visualizations of the MCTS-based perceptual planning process on real-world Windows desktop applications. These examples illustrate how DRS-GUI performs search-before-predict by expanding a perceptual tree with FOCUS, SCATTER, and SHIFT, and then selecting the most instruction-relevant region for the final grounding model. Figure 2 presents a MATLAB case, and Figure 3 presents a PowerPoint case. In both examples, DRS-GUI starts from the full screenshot and progressively refines the search trajectory into a compact region with reduced visual redundancy. These cases show that the proposed planner produces interpretable search behavior across different GUI layouts and application domains.

Table 4. Application- and platform-aware keyword sets $\mathcal{K}_{a,p}$ used in the domain-specific prompts. We keep a shared set of application-level semantic anchors and introduce platform-specific cues only when the UI terminology or layout conventions differ across systems.

Application a	**Platform** p	**Domain cues** $\mathcal{K}_{a,p}$
MATLAB	macOS	command window; workspace browser; plot toolbar; function editor; app designer
MATLAB	Windows	command window; workspace panel; plot toolbar; function editor; toolstrip
Origin	Windows	worksheet panel; graph toolbar; fitting dialog; statistics panel; analysis menu
Stata	Windows	data editor; command window; results viewer; variable manager; graph editor
EViews	Windows	workfile window; equation editor; forecasting dialog; series view; model dialog
SolidWorks	Windows	feature tree; sketch toolbar; assembly panel; drawing view; design library
AutoCAD	Windows	ribbon panel; command line; layer manager; block library; dimension tool
Inventor	Windows	browser panel; sketch constraint; assembly tool; drawing view; simulation panel
Blender	Windows	outliner panel; modifier stack; shader editor; timeline; render properties
Blender	macOS	outliner panel; modifier stack; node editor; timeline; render settings
Blender	Linux	scene outliner; geometry nodes; material editor; timeline; compositor
Unreal Engine	Windows	content browser; blueprint editor; level viewport; details panel; world outliner
Vivado	Windows	project navigator; synthesis settings; implementation flow; IP catalog; timing constraints
Vivado	Linux	flow navigator; synthesis settings; implementation flow; IP integrator; XDC constraints
Quartus Prime	Windows	project navigator; compilation flow; pin planner; timing analyzer; signal tap
Premiere Pro	Windows	timeline panel; effect controls; media browser; source monitor; export dialog
Premiere Pro	macOS	timeline panel; effect controls; media browser; source monitor; export dialog
DaVinci Resolve	macOS	color panel; node editor; fairlight mixer; timeline; deliver page
DaVinci Resolve	Windows	color panel; node editor; audio mixer; timeline; render settings
FL Studio	Windows	channel rack; mixer; piano roll; playlist; plugin browser
Photoshop	Windows	layer panel; adjustment panel; filter gallery; brush preset; selection tool
Photoshop	macOS	layer panel; adjustment panel; filter menu; brush preset; selection tool
Illustrator	Windows	artboard panel; pen tool; appearance panel; gradient editor; symbols panel
Illustrator	macOS	artboard panel; pen tool; appearance panel; gradient panel; symbols panel
Word	macOS	ribbon tab; style pane; review tools; references panel; formatting inspector
Word	Windows	ribbon tab; style pane; track changes; references panel; page layout
Excel	macOS	ribbon tab; formula bar; pivot table; chart tools; data validation
Excel	Windows	ribbon tab; formula bar; pivot table; conditional formatting; data tools
PowerPoint	Windows	slide pane; animation pane; transition tools; design theme; slide master
PowerPoint	macOS	slide pane; animation pane; transition tools; theme variants; slide master
VS Code	macOS	command palette; explorer sidebar; debug console; integrated terminal; extensions view
VS Code	Windows	command palette; explorer panel; debug console; integrated terminal; extensions view
VS Code	Linux	command palette; file explorer; debugger panel; terminal; extensions view
PyCharm	macOS	project panel; run configuration; debugger; python console; structure view
PyCharm	Windows	project panel; run configuration; debugger; python console; structure view
Android Studio	macOS	project panel; layout editor; logcat; build variants; device manager
Android Studio	Windows	project panel; XML editor; logcat; gradle tools; device manager
VMware	macOS	virtual machine library; settings panel; snapshot manager; network adapter; shared folders
VMware	Windows	virtual machine library; settings panel; snapshot manager; network editor; shared folders

Figure 2. Visualization of the MCTS-based perceptual planning process on a MATLAB Windows GUI. Starting from the full screenshot, the planner expands a tree of regions using FOCUS, SCATTER, and SHIFT actions, and finally selects the best region for grounding (highlighted).

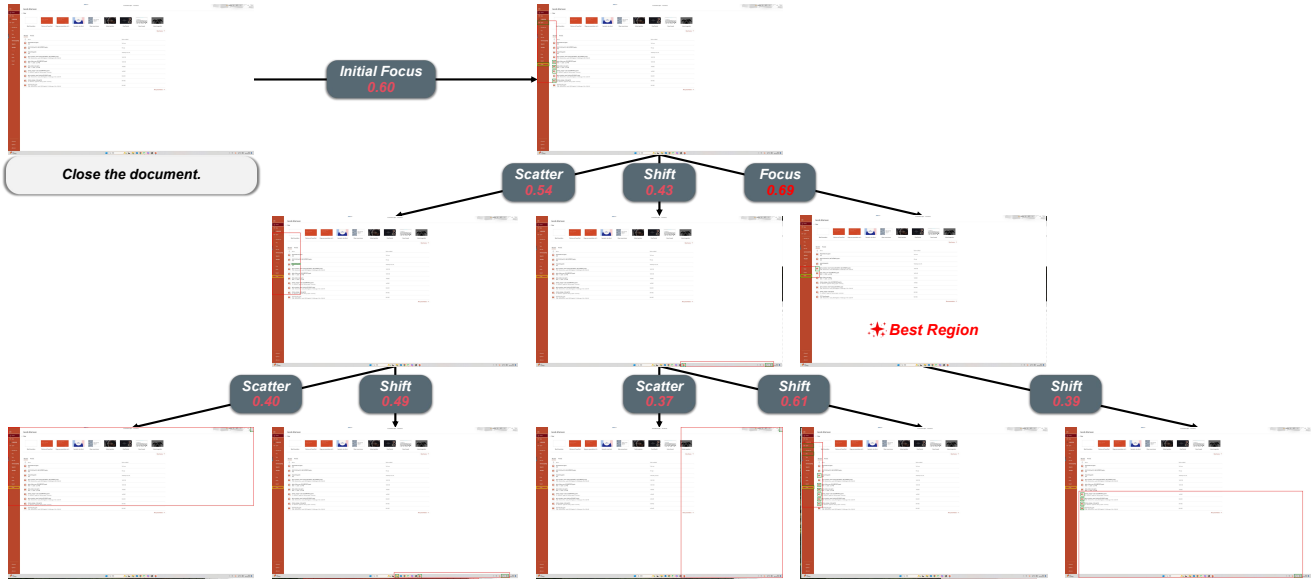

Figure 3. Visualization of the MCTS-based perceptual planning process on a PowerPoint Windows GUI. Starting from the full screenshot, the planner expands a tree of regions using FOCUS, SCATTER, and SHIFT actions, and finally selects the best region for grounding (highlighted).