FOCUSUI: Efficient UI Grounding via Position-Preserving Visual Token Selection

[1]Mingyu Ouyang, [2]Kevin Qinghong Lin, [1]Mike Zheng Shou[†], [1]Hwee Tou Ng[†]
[1]National University of Singapore   [2]University of Oxford

ouyangmingyu04@u.nus.edu, {kevin.qh.lin, mike.zheng.shou}@gmail.com, nght@nus.edu.sg
https://focusui_project_page
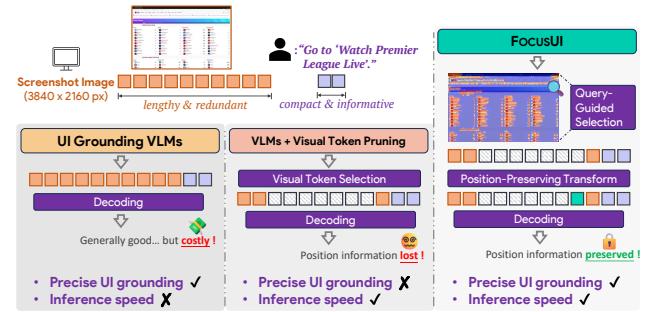
Abstract
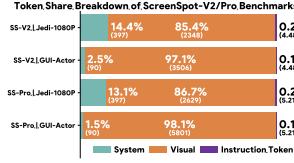
*Vision-Language Models (VLMs) have shown remarkable performance in User Interface (UI) grounding tasks, driven by their ability to process increasingly high-resolution screenshots. However, screenshots are tokenized into thousands of visual tokens (e.g., about 4700 for 2K resolution), incurring significant computational overhead and diluting attention. In contrast, humans typically **focus** on regions of interest when interacting with UI. In this work, we pioneer the task of efficient UI grounding. Guided by practical analysis of the task's characteristics and challenges, we propose FOCUSUI, an efficient UI grounding framework that selects patches most relevant to the instruction, while preserving positional continuity for precise grounding. FOCUSUI addresses two key challenges: (1) Eliminating redundant tokens in visual encoding. We construct patch-level supervision by fusing an instruction-conditioned and a rule-based UI-graph score that down-weights large homogeneous regions to select distinct and instruction-relevant visual tokens. (2) Preserving positional continuity during visual token selection. We find that general visual token pruning methods suffer from severe accuracy degradation on UI grounding tasks due to breaking positional information. We introduce a novel POSPAD strategy, which compresses each contiguous sequence of dropped visual tokens into a single special marker placed at the sequence's last index to preserve positional continuity. Comprehensive experiments on four grounding benchmarks demonstrate that FOCUSUI surpasses GUI-specific baselines. On the ScreenSpot-Pro benchmark, FOCUSUI-7B achieves performance improvement of 3.7% over GUI-Actor-7B. Also, even with only **30%** visual token retention, the performance of FOCUSUI-7B only drops by 3.2%, while achieving up to **1.44× faster** inference and **17% lower peak GPU memory**.*

1. Introduction
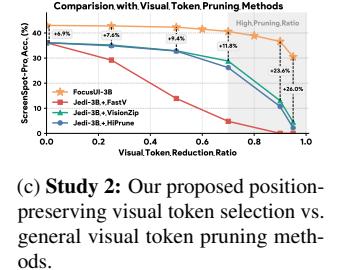
User interface (UI) visual grounding asks a model to locate a target region in a high-resolution screenshot given a nat-

---

(a) **Comparison of vanilla UI grounding VLMs**, VLMs with visual token pruning, and our FOCUSUI.

(b) **Study 1:** The exceptionally high proportion of visual (screenshot) vs. text (instruction) tokens in UI grounding tasks.

(c) **Study 2:** Our proposed position-preserving visual token selection vs. general visual token pruning methods.

Figure 1. FOCUSUI is an efficient UI grounding framework that selects *instruction-relevant* visual tokens while *preserving positional continuity*. **Study 1** provides motivation to address visual redundancy in UI grounding tasks, and **Study 2** demonstrates the effectiveness of the our position-preserving selection.

ural language instruction. Modern vision-language models (VLMs) have shown strong performance in UI tasks, including navigation and grounding, mainly driven by their abilities in processing high-resolution visual information. However, UI screenshots are typically high-resolution, and patchified into thousands of visual tokens that dominate the sequence budget (Fig. 1b). This extreme visual token skew causes substantial computational overhead. Although accuracy has improved rapidly, efficiency has been underexplored: naïve visual token pruning designed for natural images breaks *positional continuity* in multimodal sequences and yields severe accuracy drops on precise UI grounding tasks. Recent studies in token pruning strategies aim to mitigate the rapidly growing computational cost by visual tokens. It is typically
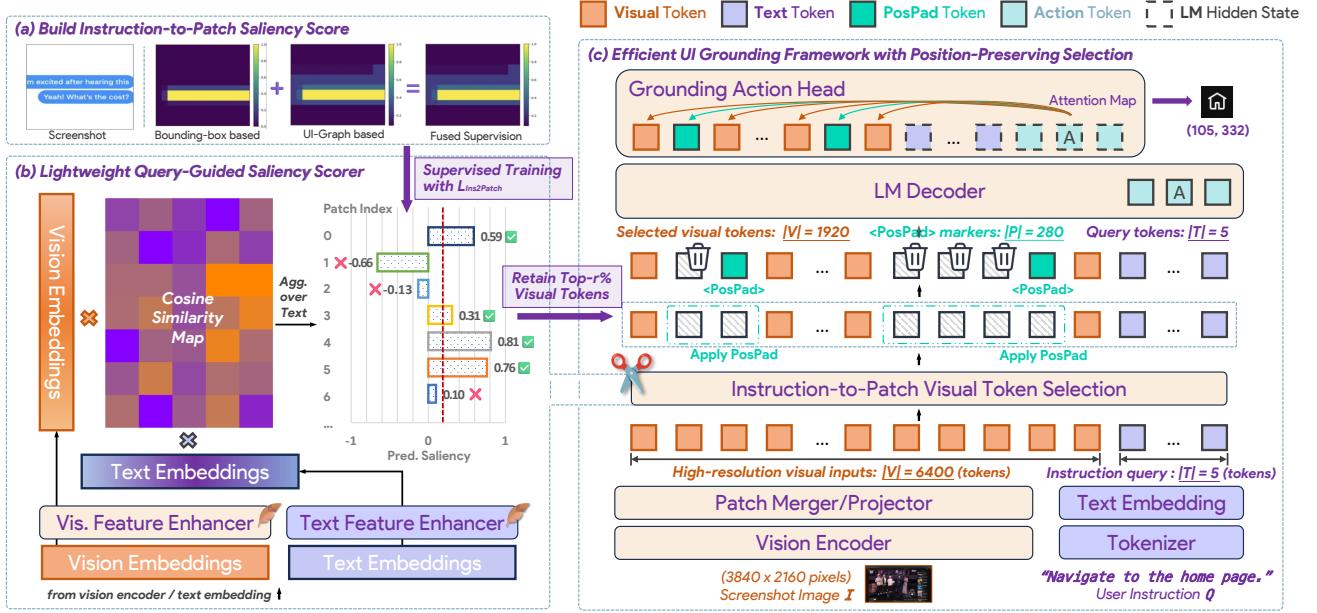
Figure 2. **Overview of our proposed FOCUSUI.** (a) Illustration of how the Instruction-to-Patch saliency score is constructed. (b) Query-guided Saliency Scorer and token selection. (c) Overall UI grounding framework illustrating how POSPAD is applied to dropped sequences to preserve positional continuity. For clarity, we omit the system prompt in the token sequence.

achieved by exploiting redundancy and importance variance, and applying selection in prefilling stage to reduce memory and computation costs during decoding. However, directly dropping visual tokens incurs position information loss, as sequence continuity is broken, leading to severe accuracy drops on precise UI grounding.

We present FOCUSUI, an efficient UI grounding framework that selects *instruction-relevant* visual tokens while preserving positional continuity needed for precise localization. First, a *lightweight* Query-Guided Saliency Scorer predicts per-patch relevance under dense supervision that fuses an instruction-conditioned bounding-box overlap signal with a rule-based UI-graph prior that down-weights large homogeneous regions. Second, we apply POSPAD, which compacts each dropped *contiguous* sequence into one learnable marker placed at the sequence's last index, preserving its positional information. This design mitigates sequence fragmentation and stabilizes grounding at aggressive retention ratios. FOCUSUI integrates seamlessly with VLMs based on Qwen2.5-VL [3] and Qwen3-VL [2] of multiple sizes. Across experiments on four benchmarks, FOCUSUI substantially speeds up inference and lowers peak GPU memory, while maintaining high accuracy.

The main contributions of this work include:

- **Pioneering the task of efficient UI grounding.** We study the task characteristics and challenges of efficient UI grounding, presenting a dedicated approach that preserves accuracy while reducing visual tokens.
- **Instruction-to-patch selection with dense supervision.**

We fuse a rule-based *UI-graph* prior with instruction-conditioned bounding-box overlap to train a lightweight Query-Guided Saliency Scorer that predicts per-patch saliency and filters irrelevant tokens.

- **Position-preserving transformation.** We introduce POSPAD to preserve sequence continuity during token selection, addressing the failure of general pruning methods on precise UI grounding tasks.
- **Practical integration and results.** We implement FOCUSUI with Qwen2.5-VL and Qwen3-VL backbones of multiple sizes (2B, 3B, and 7B). Our models outperform the best previous state-of-the-art models and show good accuracy-efficiency trade-offs across four UI grounding benchmarks.

2. Efficient UI Grounding: Task Characteristics and Challenges

We identify two key challenges in UI grounding: **(1)** *extreme token skew and redundancy from high-resolution screenshots*, and **(2)** *accuracy collapse under naïve visual token pruning due to broken positional continuity*. In this section, we provide a comprehensive empirical analysis of these challenges, thereby elaborating on the motivation for our efficient UI grounding framework.

2.1. High-Resolution Visual Understanding

The task of UI grounding differs from natural visual understanding mainly in input characteristics: **UI screenshots are typically high resolution** (e.g., 2K at 2560×1440 or 4K at

3840×2160), **compositionally structured**, and **dominated by large homogeneous panes interspersed with small widgets**. To quantify this skewness, **Study 1** in Fig. 1b shows that visual (screenshot) tokens account for $\geq 85.4\%$ of the tokens across two benchmarks and two grounding models, confirming a severe imbalance in visual tokens that incurs significant computational overhead.

This motivates an instruction-aware selection that prioritizes patches relevant to the instruction and de-emphasizes visually repetitive regions. We implement this with an Instruction-to-Patch saliency score (§3.2) that fuses: (i) bounding-box overlap with ground-truth box and (ii) a rule-based UI-graph prior that down-weights large connected components, to guide the selection.

2.2. Position Sensitivity in UI Grounding

VLMs process multimodal inputs as an interleaved sequence of visual patch tokens and text tokens [26]. In particular, Multimodal Rotary Position Embedding (M-RoPE) [28] is designed for modeling spatial and temporal relationships. In practice, Qwen2-VL's M-RoPE decomposes rotary dimensions into temporal, height, and width components to encode a (t, h, w) structure [14]. However, we find that **precise UI grounding is sensitive to the positional information of visual embeddings**, which makes token reduction more challenging. Direct pruning creates *positional jumps* in the (h, w) dimensions of M-RoPE sequence, leading to pronounced localization offsets on fine-grained targets. To investigate this sensitivity, in **Study 2** of Fig. 1c, we evaluate UI grounding models applied with advanced visual token pruning methods. The sharp accuracy drop suggests that although these pruning methods work well for general visual understanding scenerios, performance degrades dramatically on precise localization.

We address this with a POSPAD (§3.3) strategy: for each *contiguous* sequence of dropped visual tokens, we replace the sequence with a single learnable marker placed at the sequence's *last* index, inheriting that index's (h, w) positional information. This special marker preserves positional continuity and mitigates the disruption to the model's spatial understanding. Together, **Study 1** motivates *what* to remove (instruction-irrelevant or homogeneous regions), and **Study 2** dictates *how* to select (position-preserving rather than naïve dropping). These findings collectively form the motivation of our efficient UI grounding framework.

3. FOCUSUI

We introduce FOCUSUI, a query-guided efficient UI grounding framework that selects instruction-relevant visual tokens while preserving positional continuity. As illustrated in Fig. 2, FOCUSUI comprises the following key components designed for efficient UI grounding: **(i)** a fused supervision of per-patch saliency score to identify instruction-relevant

visual tokens, **(ii)** a lightweight Query-Guided Saliency Scorer for visual token selection, and **(iii)** a novel position-preserving POSPAD strategy to preserve positional information during token selection. In the following sections, we introduce each component in detail.

3.1. Instruction-to-Patch Saliency Score

Motivated by observations in §2, we first construct dense supervision of per-patch saliency scores to select relevant visual tokens. We fuse two complementary components: (i) instruction-conditioned bounding-box overlap and (ii) a UI-graph prior via union-find that down-weights large homogeneous regions.

Bounding-Box Saliency Score. As summarized in Alg. 1, we partition the image into a $G_h \times G_w$ patch grid with patch size p, and denote the patch cell by $R_{i,j} = [jp, ip, (j+1)p, (i+1)p]$. Given an element bounding box b_{gt}, each patch cell receives a score proportional to its overlap with b_{gt}. We set $S_{\text{bbox}} \in [0, 1]$ with normalized overlap area$(R_{i,j} \cap b_{gt})/p^2$ so that fully covered patches score 1

Algorithm 1: Building Bounding-Box Saliency Score

Input: $I \in [0, 1]^{H \times W \times 3}$, patch size p, ground-truth bbox
　　　$b_{gt} = (x_1, y_1, x_2, y_2)$
Output: $S_{\text{bbox}} \in [0, 1]^{G_h \times G_w}$
$G_h \leftarrow \lfloor H/p \rfloor, \quad G_w \leftarrow \lfloor W/p \rfloor$
for $i \leftarrow 0$ **to** $G_h - 1$ **do**
　　for $j \leftarrow 0$ **to** $G_w - 1$ **do**
　　　　$R_{i,j} \leftarrow [jp, ip, (j+1)p, (i+1)p]$;
　　　　$S_{\text{bbox}}[i, j] \leftarrow$ area$(R_{i,j} \cap b_{gt})/p^2$
return S_{bbox}

Algorithm 2: Building UI-Graph Saliency Score

Input: $I \in [0, 1]^{H \times W \times 3}$, threshold τ, patch size p
Output: $S_{\text{uig}} \in [0, 1]^{G_h \times G_w}$
$G_h \leftarrow \lfloor H/p \rfloor, \quad G_w \leftarrow \lfloor W/p \rfloor$
Form patch pixels $PP_{i,j} \in \mathbb{R}^{3 \times p \times p}$ for
　$0 \leq i < G_h, \ 0 \leq j < G_w$
Union-Find on nodes (i, j) **for** $i \leftarrow 0$ **to** $G_h - 1$ **do**
　　for $j \leftarrow 0$ **to** $G_w - 1$ **do**
　　　　if $j + 1 < G_w$ **and**
　　　　$\|\text{vec}(PP_{i,j}) - \text{vec}(PP_{i,j+1})\|_2 < \tau$ **then**
　　　　　　UNION$\big((i, j), (i, j + 1)\big)$
　　　　if $i + 1 < G_h$ **and**
　　　　$\|\text{vec}(PP_{i,j}) - \text{vec}(PP_{i+1,j})\|_2 < \tau$ **then**
　　　　　　UNION$\big((i, j), (i + 1, j)\big)$

Obtain component ids $r_{i,j} \leftarrow$ FIND(i, j)
Counts $n_u \leftarrow \big|\{(i, j) : r_{i,j} = u\}\big|$ for each unique root u
Assigning Weights: $w_u \leftarrow \big(\max\{1, \ln(n_u + 1)\}\big)^{-1}$
Set $S_{\text{uig}}[i, j] \leftarrow w_{r_{i,j}}$ for all i, j
return S_{uig}

---

and disjoint patches score 0, inducing a center-to-edge decay along the box boundary.

UI-Graph Saliency Score. To further suppress background regions and enrich supervision on non-annotated regions, we propose a UI-graph saliency score based on union–find over connected components of visual patches, which is inspired by the *UI-graph* prior in ShowUI [17]. Specifically, we treat each patch (i, j) in $R_{i,j}$ as a node and connect 4-neighborhood pairs whose ℓ_2 distance in the RGB space is below a threshold τ. Such union–find groups connected components whose size n_u reflects how visually repetitive a region is.

We then assign a weight $w_u = (\max\{1, \ln(n_u+1)\})^{-1}$ to each patch so that large homogeneous regions (*e.g.*, empty backgrounds) receive lower weights. The UI-graph score S_{uig} sets each patch to its component weight w_u. Such design naturally suppresses background regions and enhances the saliency of distinctive elements. This score is instruction-agnostic, annotation-free, and complements S_{bbox} for each patch. See Alg. 2 for the full procedure.

Fuse Supervision. Finally, we fuse the two scores to obtain joint supervision $S_{\text{Ins2Patch}}$ as Instruction-to-Patch saliency score:

$$S_{\text{Ins2Patch}} = \lambda\, S_{\text{bbox}} + (1 - \lambda)\, S_{\text{uig}} \qquad (1)$$

where $\lambda \in [0, 1]$ is a controllable weight and empirically set to 0.8 across experiments. Fig. 3 provides an illustration of the two components and the final fused supervision.
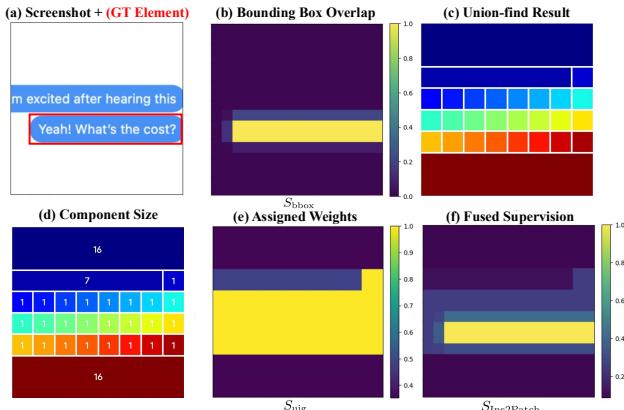

Figure 3. Illustrative example of building the Instruction-to-Patch saliency score. **(a)** Screenshot I with ground-truth bounding box b_{gt}. **(b)** Bounding-box saliency score S_{bbox}. **(c)** Union-find results. **(d)** Size of each connected component n_u. **(e)** UI-graph saliency score S_{uig}. **(f)** Fused supervision $S_{\text{Ins2Patch}}$ by combining **(d)** and **(e)**. Brighter regions represent positive patches and darker regions represent negative patches.

3.2. Lightweight Query-Guided Saliency Scorer

With the obtained per-patch supervision $S_{\text{Ins2Patch}}$ from Eq. (1), we train a *lightweight* module, Query-Guided Saliency Scorer, that predicts per-patch saliency from similarities between patch and query text embeddings in the VLM backbone, as shown in Fig. 2 (b).

Concretely, let $\{v_i\}_{i=1}^M$ be patch embeddings from the vision encoder and $\{e_j\}_{j=1}^N$ be query text embeddings (only the part corresponding to the instruction) in the language model (LM) space. We use a self-attention layer to enhance features in each modality, preserving the original embedding semantics while strengthening cross-modal interactions. A tanh constraint followed by ℓ_2 normalization is applied to each feature to bound the similarities. We then compute token-wise similarities $\mathcal{P} \in \mathbb{R}^{M \times N}$ by a matrix product between patch and text embeddings. Finally, we aggregate the similarities over text query dimensions with mean pooling to get per-patch saliency scores s_i:

$$\mathcal{P} = \tilde{V}\tilde{E}^\top \in \mathbb{R}^{M \times N}, \quad s_i = \frac{1}{N}\sum_{j=1}^N \mathcal{P}_{i,j}. \qquad (2)$$

To train the Query-Guided Saliency Scorer, we convert scores to probabilities and optimize a KL divergence objective. Given fused supervision from Eq. (1), we minimize:

$$\mathcal{L}_{\text{Ins2Patch}} = \text{KL}(\text{softmax}(S_{\text{Ins2Patch}}) \,\|\, \text{softmax}(s)). \quad (3)$$

3.3. POSPAD: **Positional Continuity Preservation**

Token Selection Policy. We first apply top-K selection over predicted per-patch saliency scores $\{s_i\}_{i \in \mathcal{I}}$ from Eq. (2). Given a retention ratio $r \in (0, 1]$, the number of kept patches is set to $K = \lfloor rM \rfloor$. Let γ be the K-th element of the sorted list $\{s_i\}_{i \in \mathcal{I}}$. We form the kept index set $\mathcal{K} = \{i \in \mathcal{I} \mid s_i \geq \gamma\}$ and drop the remaining indices $\mathcal{D} = \{i \in \mathcal{I} \mid s_i < \gamma\}$.

Sequence Transformation. After selecting instruction-relevant visual tokens, we further refine the sequence to alleviate positional information loss in the model's spatial understanding. We introduce POSPAD, a position-preserving sequence transformation that replaces each *contiguous sequence* of dropped visual tokens with a *single* learnable special token POSPAD placed at the *last index* of that sequence. The illustration of POSPAD is shown in Fig. 4.

Specifically, given the original visual token sequence $x_{1:M}$, the kept index set $\mathcal{K}$, and the drop index set $\mathcal{D}$ defined above, we partition $\mathcal{D}$ into *contiguous sequences* (*i.e.*, *maximal consecutive sequences*) $\mathcal{R}_1, \dots, \mathcal{R}_U$ with respect to the 1D flattened sequence order. For each sequence $\mathcal{R}_u$, we keep only its last index $r_u^{\text{end}} = \max \mathcal{R}_u$ and remove the others. Let $\mathcal{E}_{\text{seq-end}} = \{r_u^{\text{end}}\}_{u=1}^U$ denote the set of sequence-end indices, and define the preserved index set $\mathcal{S} = \mathcal{K} \cup \mathcal{E}_{\text{seq-end}}$.

4
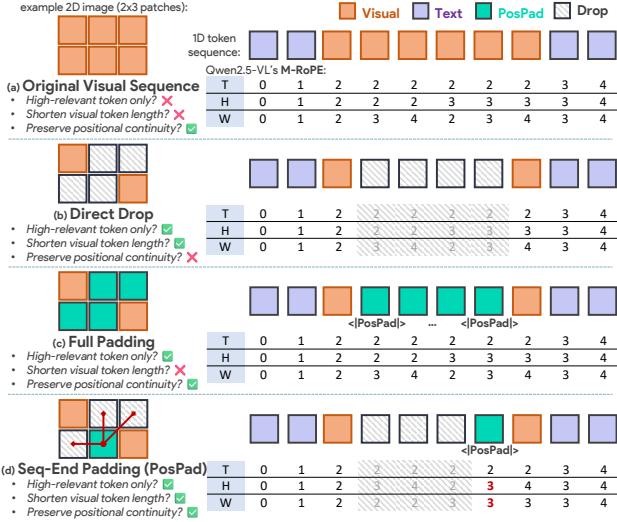
Figure 4. Illustration of POSPAD sequence transformation for positional continuity preservation via an example 2D image (2×3 patches) and its 1D sequence. A learnable `<pos_pad>` marker is placed at the last index of each contiguous sequence of dropped visual tokens, as illustrated by strategy (d).

We then replace each contiguous sequence with a single marker `<pos_pad>` and keep all other tokens unchanged:

$$x'_j = \begin{cases} \texttt{<pos\_pad>} & \text{if } j \in \mathcal{E}_{\text{seq-end}}, \\ x_j & \text{if } j \in \mathcal{K}, \end{cases} \quad (4)$$
$$\text{POSPAD}(x_{1:M}) = \{x'_j\}_{j \in \mathcal{S}}.$$

Thus, the final output length of visual tokens is $M' = M - (|\mathcal{D}| - U)$, with the total number of `<pos_pad>` tokens being U. Each dropped sequence $\mathcal{R}_u$ reduces the sequence by $|\mathcal{R}_u| - 1$ while preserving positional continuity at the sequence end. Concrete examples of M, M', and U under different retention ratios are investigated in Tab. 7c.

Compared to direct dropping, POSPAD preserves positional continuity and empirically stabilizes the model's spatial understanding. Alternative strategies are also studied in §4.2.5. Since POSPAD alters only sequence sparsity and not token indices or rotary bases, it is compatible with common M-RoPE implementations and requires no modifications to the downstream LM architecture.

3.4. Efficient UI Grounding Framework

Integration with VLMs. We integrate our visual token selection strategy into existing VLMs before visual patch embeddings are fed into the LM decoder. Concretely, the Query-Guided Saliency Scorer takes the patch features $\{v_i\}_{i=1}^{M}$ and the instruction token embeddings e_j, computes scores s_i via Eq. (2), and selects the top-K indices $\mathcal{K}$ for a given retention ratio r. We then refine the sequence with POSPAD, yielding a compact visual sequence of length $M' \ll M$ that preserves positional continuity. The LM decoder processes this

sequence without altering its original architecture. We apply our framework to Qwen2.5-VL and Qwen3-VL models. For the Qwen3-VL model with a DeepStack [20] vision encoder, deep visual embeddings are gathered only for the kept image tokens $\mathcal{K}$.

Coordinate-free UI Grounding with Selected Patches. We find the coordinate-free UI grounding scheme from GUI-Actor [30] most compatible with our selection: the model grounds elements directly at the patch embeddings with an extra action head on top of the LM decoder, while our visual token selection reduces candidates by discarding instruction-irrelevant regions. Specifically, the decoder LM outputs a sequence of action tokens:

$$\begin{aligned} \text{LM}(I, q) = \{ & x_{1:i-1}, \texttt{<ACTOR\_START>}, \\ & \texttt{<ACTOR>}, \texttt{<ACTOR\_END>}, x_{i+3:N} \}. \end{aligned} \quad (5)$$

Then the action head aligns $h_{\texttt{ACTOR}}$ with visual patches to produce an attention map over patches. We first contextually refine selected patch features $\{\tilde{v}_i\}_{i=1}^{M'}$ with a self-attention layer: $\tilde{v}_1, \ldots, \tilde{v}_{M'} = \text{SelfAttn}(v_1, \ldots, v_{M'})$. Then we project $h_{\texttt{ACTOR}}$ and each $\tilde{v}_i$ with separate MLP_T and MLP_V and compute attention scores:

$$\begin{aligned} z &= \text{MLP}_T(h_{\texttt{ACTOR}}), \quad z_i = \text{MLP}_V(\tilde{v}_i), \\ \alpha_i &= \frac{z^\top z_i}{\sqrt{d}}, \quad a_i = \text{softmax}(\alpha)_i. \end{aligned} \quad (6)$$

The distribution a_i identifies the most relevant regions for executing the action. With selected visual tokens, such an action head benefits from fewer visual candidates and retained patches that are more relevant to the instruction.

Training Objective. The Query-Guided Saliency Scorer is trained end-to-end with the downstream LM objective next-token prediction loss $\mathcal{L}_{\text{NTP}}$ and an action-attention loss $\mathcal{L}_{\text{Attn}}$ for grounding:

$$\mathcal{L}_{\text{Attn}} = \sum_{i=1}^{M'} p_i \log \frac{p_i}{a_i}, \ p_i = \frac{y_i}{\sum_{j=1}^{M'} y_j + \epsilon}, \ i = 1, \ldots, M' \quad (7)$$

where y_i denotes the attention score label for the i-th patch (1 if it overlaps with the ground-truth bounding box, 0 otherwise) and ϵ is a small constant for numerical stability. The overall training objective is:

$$\mathcal{L} = \mathcal{L}_{\text{Ins2Patch}} + \mathcal{L}_{\text{NTP}} + \mathcal{L}_{\text{Attn}}. \quad (8)$$

4. Experiments

4.1. Experimental Setup

Implementation Details We adopt the state-of-the-art VLMs Qwen2.5-VL [3] and Qwen3-VL [2] as our base models, with different sizes to demonstrate the generalizability of our approach. We conduct supervised fine-tuning to

Model	ScreenSpot-V2							ScreenSpot-Pro								
	Mob.-T	Mob.-I	Des.-T	Des.-I	Web-T	Web-I	Avg	Dev	Cre.	CAD	Sci.	Office	OS	Avg-T	Avg-I	Avg
Operator [22]	47.3	41.5	90.2	80.3	92.8	84.3	70.5	35.1	39.6	16.1	43.7	53.0	32.7	45.0	23.0	36.6
OS-Atlas-7B [31]	95.2	75.8	90.7	63.6	90.6	77.3	84.1	17.7	17.9	10.3	24.4	27.4	16.8	28.1	4.0	18.9
Aguvis-7B [35]	95.5	77.3	95.4	77.9	91.0	72.4	86.0	16.1	21.4	13.8	34.6	34.3	19.4	-	-	22.9
Tong-UI-7B [38]	93.1	81.5	96.4	82.9	90.2	84.7	88.7	22.7	21.1	15.3	34.3	38.3	18.4	35.1	8.0	25.7
UGround-V1-7B [13]	95.0	83.3	95.0	77.8	92.1	77.2	87.6	28.1	31.7	14.6	39.0	49.6	24.5	-	-	31.1
UI-TARS-7B [23]	96.9	89.1	95.4	85.0	93.6	85.2	91.6	36.1	32.8	18.0	50.0	53.5	24.5	47.8	16.2	35.7
UI-TARS-72B [23]	94.8	86.3	91.2	87.9	91.5	87.7	90.3	40.8	39.6	17.2	45.7	54.8	30.1	50.9	17.5	38.1
UI-TARS-1.5-7B [23]	-	-	-	-	-	-	90.0	31.8	40.2	31.8	47.2	65.6	33.2	-	-	42.6
Qwen2.5-VL-3B [3]	93.4	73.5	88.1	58.6	88.0	71.4	80.9	21.4	25.8	18.4	29.5	40.9	20.4	37.8	6.6	25.9
Qwen2.5-VL-7B [3]	97.6	87.2	90.2	74.2	93.2	81.3	88.8	29.1	24.9	13.8	31.1	45.7	22.4	39.9	7.6	27.6
Qwen2.5-VL-32B [3]	97.9	88.2	98.5	79.3	91.2	86.2	91.3	48.5	41.1	32.6	57.1	67.4	42.3	63.2	22.5	47.6
GUI-Actor-3B [30]	97.6	83.4	96.9	83.6	94.0	85.7	91.0	39.8	36.7	34.1	49.6	61.3	35.2	-	-	42.2
GUI-Actor-7B [30]	97.6	88.2	96.9	85.7	93.2	86.7	92.1	38.1	41.4	38.3	50.8	63.0	38.8	-	-	44.6
Jedi-3B [33]	96.6	81.5	96.9	78.6	88.5	83.7	88.6	38.1	34.6	23	38.6	57.0	25.0	49.8	13.7	36.1
Jedi-7B [33]	96.9	87.2	95.9	87.9	94.4	84.2	91.7	27.4	34	32.2	52.4	68.7	26.0	52.6	18.2	39.5
FOCUSUI-3B ($r = 100\%$)	99.2	85.9	96.1	87.3	95.4	81.9	91.5	43.1	37.0	37.6	48.4	61.7	38.3	59.3	18.9	43.8
FOCUSUI-3B ($r = 50\%$)	98.8	86.9	95.0	87.3	95.4	81.9	91.4	42.1	37.0	36.4	46.9	58.3	35.2	56.7	19.0	42.3
FOCUSUI-3B ($r = 30\%$)	98.5	85.3	96.1	87.3	94.3	81.9	91.0	38.1	35.8	33.3	44.5	57.8	37.2	55.0	17.4	40.6
FOCUSUI-7B ($r = 100\%$)	98.8	91.6	95.6	92.1	95.0	84.4	93.1	44.5	41.1	42.9	52.0	69.6	44.4	64.7	21.9	48.3
FOCUSUI-7B ($r = 50\%$)	98.8	92.2	93.9	87.3	95.0	85.2	92.6	42.8	40.5	40.2	51.6	67.0	40.3	61.7	21.9	46.5
FOCUSUI-7B ($r = 30\%$)	98.8	90.1	93.3	85.7	93.9	85.2	91.8	38.8	39.9	42.9	49.2	64.4	38.8	60.4	20.4	45.1

Table 1. Performance comparison on *ScreenSpot-V2* [31] and *ScreenSpot-Pro* [15].

Model	Text	Elem	Layout	Manip	Refuse	Avg
Gemini-2.5-Pro [12]	59.8	45.5	49.0	33.6	38.9	45.2
Operator [22]	51.3	42.4	46.6	31.5	0.0	40.6
UGround-V1-7B [13]	51.3	40.3	43.5	24.8	0.0	36.4
Aguvis-7B [35]	55.9	41.2	43.9	28.2	0.0	38.7
UI-TARS-7B [23]	60.2	51.8	54.9	35.6	0.0	47.5
UI-TARS-1.5-7B [23]	70.1	57.9	59.7	51.7	0.0	56.0
Qwen2.5-VL-3B [3]	41.4	28.8	34.8	13.4	0.0	27.3
Qwen2.5-VL-7B [3]	45.6	32.7	41.9	18.1	0.0	31.4
GUI-Actor-3B [30]	60.5	56.1	58.5	32.2	0.0	50.5
GUI-Actor-7B [30]	60.2	54.2	58.1	30.9	0.0	49.5
Jedi-3B [33]	67.4	53.0	53.8	44.3	7.4	50.9
Jedi-7B [33]	65.9	55.5	57.7	46.9	7.4	54.1
FOCUSUI-3B ($r = 100\%$)	65.9	57.6	59.7	37.6	0.0	53.4
FOCUSUI-3B ($r = 50\%$)	64.8	59.4	63.6	37.6	0.0	54.6
FOCUSUI-3B ($r = 30\%$)	62.5	56.7	62.9	33.6	0.0	51.8
FOCUSUI-7B ($r = 100\%$)	63.6	61.2	63.6	34.9	0.0	54.4
FOCUSUI-7B ($r = 50\%$)	64.0	62.1	63.6	31.5	0.0	54.1
FOCUSUI-7B ($r = 30\%$)	63.6	60.9	64.4	31.5	0.0	53.9

Table 2. Performance comparison on OSWorld-G [33].

Model	Basic	Functional	Spatial	Avg
Claude-3.7-Sonnet [1]	9.48	7.73	7.60	8.27
ShowUI-2B [17]	8.07	7.67	2.07	5.94
OSAtlas-7B [31]	12.2	11.2	3.67	9.02
UGround-7B [13]	11.5	12.2	2.79	8.83
UGround-V1-7B [13]	15.4	17.1	6.25	12.9
Aguvis-7B [35]	17.8	18.3	5.06	13.7
UI-TARS-7B [23]	20.1	24.3	8.37	17.6
UI-TARS-72B [23]	31.4	30.5	14.7	25.5
GUI-Actor-3B [30]	27.4	24.6	7.0	19.3
GUI-Actor-7B [30]	30.1	28.1	7.8	21.6
Jedi-3B [33]	22.3	25.2	9.35	18.7
Jedi-7B [33]	32.3	30.5	12.8	24.8
FOCUSUI-3B ($r = 100\%$)	30.0	26.9	8.7	21.5
FOCUSUI-3B ($r = 50\%$)	29.7	26.0	8.2	20.9
FOCUSUI-3B ($r = 30\%$)	29.1	26.4	7.6	20.6
FOCUSUI-7B ($r = 100\%$)	33.6	31.2	11.2	24.9
FOCUSUI-7B ($r = 50\%$)	32.5	31.0	11.3	24.5
FOCUSUI-7B ($r = 30\%$)	32.3	29.2	11.0	23.8

Table 3. Performance comparison on UI-Vision [21].

obtain the following variants: FOCUSUI-3B and FOCUSUI-7B with Qwen2.5-VL and FOCUSUI-QWEN3-VL-2B with Qwen3-VL.

For fair comparison, we align the training budget with the baseline method GUI-Actor [30], using approximately 1M screenshots collected from several public UI datasets. To ensure annotation quality, we follow V2P [6] to apply Omni-Parser [19] to filter samples whose IoU between ground-truth and detected boxes is below 0.3. The visual token retention ratio r is sampled uniformly from $(0.1, 1.0)$ during training. All models are trained with DeepSpeed [25] Zero-2 on $8\times$NVIDIA H200 GPUs for 1 epoch. More training details are provided in the Appendix.

Evaluation Benchmarks We conduct experiments on four UI grounding benchmarks, including ScreenSpot-V2 [31],

Model	ScreenSpot-V2			ScreenSpot-Pro		
	Avg-T	Avg-I	Avg	Avg-T	Avg-I	Avg
Qwen3-VL-2B† [3]	94.7	78.9	87.8	52.8	16.7	39.0
FOCUSUI-QWEN3-VL-2B (r = 100%)	95.8	85.6	91.4	51.5	20.9	39.8
FOCUSUI-QWEN3-VL-2B (r = 50%)	95.7	85.0	91.0	52.5	20.9	40.4
FOCUSUI-QWEN3-VL-2B (r = 30%)	93.5	84.3	89.5	49.7	20.2	38.5

Table 4. Performance comparison of models based on the **Qwen3-VL** backbone. † indicates results obtained from our own evaluation of the official model on HuggingFace [29].

ScreenSpot-Pro [15], OS-World-G [33], and UI-Vision [21]. Among them, ScreenSpot-Pro features higher-resolution interfaces that simulate multi-source real-world applications, serving as a practical testbed for evaluating the properties of efficiency and precise UI grounding.

4.2. Main Results

We organize our main results with the following five research questions (RQs):

- **RQ1 (§4.2.1 Performance)** : *Can FOCUSUI effectively reduce visual tokens while preserving accuracy?*
- **RQ2 (§4.2.2 Comparison to General Pruning Methods)**: *How does FOCUSUI compare to general visual token pruning methods?*
- **RQ3 (§4.2.3 Efficiency Analysis)**: *What efficiency gains does FOCUSUI achieve under different settings?*
- **RQ4 (§4.2.4 Qualitative Results)**: *How does FOCUSUI select instruction-relevant visual tokens?*
- **RQ5 (§4.2.5 Ablation Study)**: *How do the components and retention settings affect performance?*

4.2.1. RQ1: Performance

Tables 1, 2 and 3 report grounding performance on ScreenSpot-V2 [31] & ScreenSpot-Pro [15], OS-World-G [33], and UI-Vision [21], respectively. We test a series of retention ratios $r \in \{100\%, 50\%, 30\%\}$ to characterize degradation curves and compare to dense baselines that consume all visual tokens. Across all four benchmarks, FO-CUSUI exceeds GUI-specific baselines with the same size even at $30-50\%$ token retention, achieving state-of-the-art grounding performance. Additionally, we report the performance of FOCUSUI-QWEN3-VL-2B based on the more recent state-of-the-art Qwen3-VL [2] backbone in Tab. 4. More detailed breakdown and retention ratio results are provided in the Appendix.

4.2.2. RQ2: Comparison to General Pruning Methods

Tab. 5 presents a comparison with alternative general VLM visual token pruning methods. Specifically, we compare against Fast-V [7], HiPrune [18], and Vision-Zip [36]. Our FOCUSUI preserves near-baseline accuracy at 30% token retention (within 0.5/3.2/2.5 points on ScreenSpot-V2/Pro/OS-World-G), while general pruning severely degrades performance. Notably, our method is natively compatible with

Model + Pruning Method (Venue)	%Ret. Ratio	SS-V2 Avg	SS-Pro Avg	OSWorld-G Avg
Qwen2.5-VL-3B	100%	81.5	26.1	27.3
+ Fast-V (ECCV'24) [7]	30%	38.6 (-52.7%)	4.8 (-81.6%)	14.4 (-47.4%)
+ HiPrune (arXiv'25) [18]	30%	72.0 (-11.7%)	18.0 (-30.8%)	20.4 (-25.3%)
+ Vision-Zip (CVPR'25) [36]	30%	75.4 (-7.5%)	18.9 (-27.4%)	23.0 (-15.6%)
Jedi-3B	100%	88.9	36.1	48.8
+ Fast-V (ECCV'24) [7]	30%	51.0 (-42.6%)	14.1 (-60.9%)	23.9 (-51.0%)
+ HiPrune (arXiv'25) [18]	30%	80.9 (-9.0%)	26.2 (-27.3%)	40.4 (-17.1%)
+ Vision-Zip (CVPR'25) [36]	30%	82.8 (-6.9%)	28.8 (-20.3%)	41.5 (-14.9%)
FOCUSUI-3B	100%	91.5	43.8	53.4
+ Saliency Scorer w/ POSPAD	30%	91.0 (-0.5%)	40.6 (-7.3%)	51.8 (-3.0%)

Table 5. Comparison to general visual token pruning methods.

Model	%Ret. Ratio	#Vis. Token	per Sample Time (sec)	Max GPU Mem. (MB)	SS-Pro Acc
Base Model: Qwen2.5-VL, $max\_pixel = 6400 * 28 * 28 = 4816000$					
FOCUSUI-7B	100%	5319	1.75 (1.00×)	20994 (1.00×)	48.3
FOCUSUI-7B	70%	3989	1.67 (1.05×)	18334 (0.87×)	47.7
FOCUSUI-7B	50%	2659	1.49 (1.18×)	17944 (0.85×)	46.5
FOCUSUI-7B	30%	1329	1.22 (1.44×)	17392 (0.83×)	45.1
Base Model: Qwen3-VL, $max\_pixel = 6000 * 32 * 32 = 6144000$					
FOCUSUI-QWEN3-VL-2B	100%	4627	0.97 (1.00×)	6278 (1.00×)	39.8
FOCUSUI-QWEN3-VL-2B	70%	3470	0.90 (1.08×)	6142 (0.98×)	40.1
FOCUSUI-QWEN3-VL-2B	50%	2313	0.85 (1.14×)	5680 (0.91×)	40.4
FOCUSUI-QWEN3-VL-2B	30%	1156	0.71 (1.37×)	5170 (0.82×)	38.5

Table 6. Efficiency analysis on ScreenSpot-Pro benchmark under different retention ratios and model backbones of FOCUSUI. * The number of <pos_pad> tokens is not included.

FlashAttention [10] since it does not require any intermediate attention or activation information.

4.2.3. RQ3: Efficiency Analysis

In Tab. 6, we evaluate FOCUSUI with different Qwen2.5-VL and Qwen3-VL backbones to study efficiency gains and accuracy-efficiency trade-offs. Results show that reducing retention ratio from 100% to 30% yields up to 1.44× faster inference and about 17–18% lower peak memory with only 3.2-point accuracy loss.

4.2.4. RQ4: Qualitative Results

Fig. 5 shows qualitative examples of FOCUSUI. The predicted heatmaps show that the model effectively selects the relevant visual tokens for the instruction while suppressing background regions.

4.2.5. RQ5: Ablation Study

We highlight the effectiveness of our proposed components in Tab. 7. Models evaluated in experiments in Tab. 7a and 7b use Qwen2.5-VL-3B as the base model and are trained with 30% of the full data.

Visual Token Selection. We compare with the variants illustrated in Fig. 4: (a) *Original visual sequence*. (b) *Direct Drop*. (c) *Full Padding* which preserves continuity by inserting <pos_pad> at every dropped position. We also test zero-shot CLIP [24] as the scoring strategy. The performance of these variants is shown in Tab. 7a.

Instruction query: "Switch to explore projects."

Instruction query: "Show all cityscape wallpapers."

Instruction query: "Check personalized recommendations."
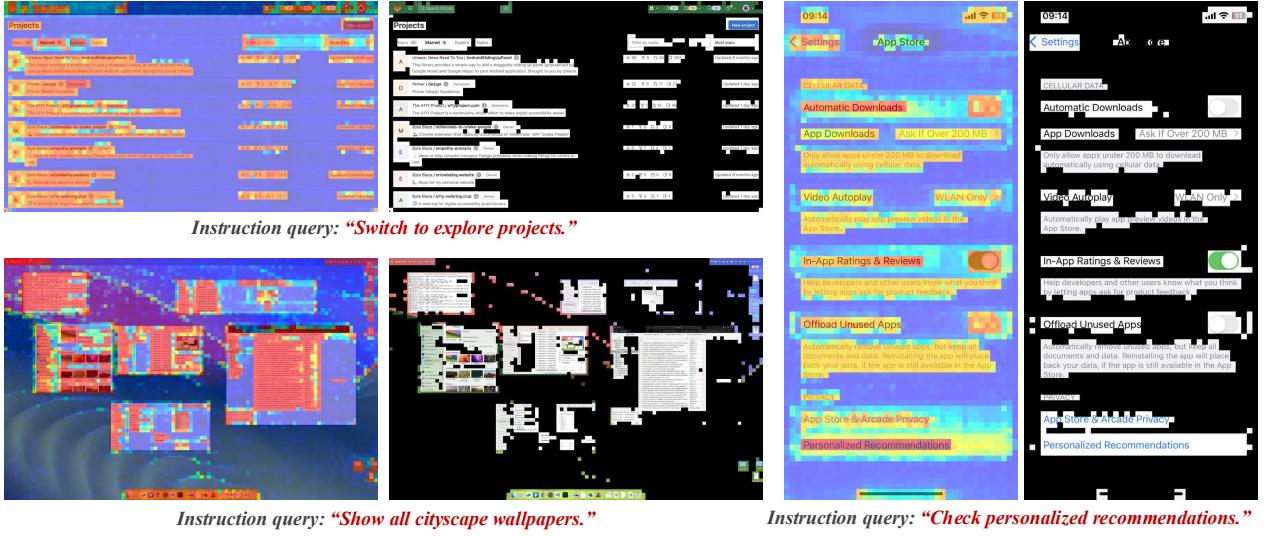
Figure 5. Qualitative visualization of predicted saliency heatmaps and retained patches under a **retention ratio** $r = \mathbf{30}\%$. Black regions denote dropped visual tokens that are not consumed by the LM during decoding. Examples are taken from the ScreenSpot-V2 and ScreenSpot-Pro benchmarks, spanning web, desktop, and mobile interfaces.

Scoring Variant	Retain Strategy	%Ret. Ratio	Reduce Token Len?	Preserve Pos. Continuity?	SS-Pro Acc
Baseline	(a) N/A	100%	✗	✓	40.9
CLIP Score [24]	(b) Direct drop	50%	✓	✗	28.5
	(c) Full padding	50%	✗	✓	38.7
	(d) POSPAD	50%	✓	✓	38.2
Ins2Patch Score	(b) Direct drop	50%	✓	✗	29.2
	(c) Full padding	50%	✗	✓	42.1
	(d) POSPAD	50%	✓	✓	**42.3**

(a) Different visual token selection methods and positional continuity retention strategies.

Variant	SS-Pro Acc
w/ UI-Graph Labeling only	41.1
w/ BBox-based Labeling only	39.8
Full FOCUSUI	**42.3**

(b) Ins2Patch score ablation with a reduction rate of 50%.

%Ret. Ratio	#Vis. Tokens	#POSPAD Tokens	#Total Tokens	SS-Pro Acc
100%	6019	0	6140	43.8
75%	4514	435	5070	43.3
50%	3009	433	3563	42.3
25%	1504	315	1941	40.6
10%	601	193	915	36.6

(c) Different retention ratios and numbers of tokens.

Table 7. Ablation of key components of FOCUSUI.

Instruction-to-Patch Saliency Score Supervision. Results in Tab. 7b indicate that removing either the UI-graph prior or the bounding-box overlap score degrades accuracy relative to the fused supervision of FOCUSUI.

Retention Ratio. Tab. 7c suggests a smooth accuracy-retention trade-off of our FOCUSUI: 100% matches the dense baseline, 50% still retains most performance, and further aggressive settings incur larger accuracy drops.

5. Related Work

VLM-Powered GUI Agents Recent advances in VLMs have accelerated progress on GUI agents that perceive, plan, and act in graphical interfaces [1, 3, 12, 22, 28]. Starting from text-dependent GUI agents [11, 42], it progressively transitions to purely visual solutions for task planning, element grounding, and interface control [9, 17, 23, 33] that fully utilizes VLMs' capability.

UI Visual Grounding Given a screenshot and a natural language instruction, UI grounding locates the target region for interaction on the screen. With more advanced model design [6, 9, 17, 23, 27, 30, 31, 37, 38] and data scaling [8, 13, 33, 35], the performance of UI grounding improves rapidly in recent times.

Visual Token Reduction Compared to information-dense text, visual tokens often exhibit substantial redundancy, motivating token reduction to lower computation cost [4, 7, 34, 40]. Recent work further explores training-free pruning based on token importance and redundancy [18, 41] or implement encoder-side compression [36, 39].

6. Conclusion

In this paper, we introduced FOCUSUI, a query-guided framework for efficient UI grounding that selects *instruction-relevant* visual tokens while preserving positional continuity. Integrated with state-of-the-art VLMs, FOCUSUI achieves strong accuracy-efficiency trade-offs across four UI grounding benchmarks.

Limitations and Future Work. FOCUSUI primarily gains efficiency from spatial visual token reduction. Future work may consider the temporal dimension, as UI interactions typically involve multi-round and sequential actions.

A. Implementation Details

A.1. Training Data

Raw Dataset Our training set compiles several public high-quality GUI datasets, following GUI-Actor. To ensure fair evaluation, samples from Wave-UI that overlap with the test sets of downstream tasks are excluded.

Refining Annotation Quality We apply OmniParser V2 [19] to filter samples whose IoU between ground-truth and OmniParser detected boxes is below 0.3. This results in a reduction of 22.9% in the number of elements. The final training statistics are in Tab. 8.

Dataset	#Screenshots	#Elements	Platform
UGround [13]	775K	8M	Web
GUI-Env [30]	70K	262K	Web
GUI-Act [30]	13K	42K	Web
AndroidControl [16]	47K	47K	Android
AMEX [5]	100K	1.2M	Android
Wave-UI	7K	50K	Hybrid
Total (Raw Dataset)	1012K	9.6M	–
Total (After Filtering)	**976K**	**7.4M**	–

Table 8. Statistics of training datasets used for FOCUSUI.

A.2. Training Details
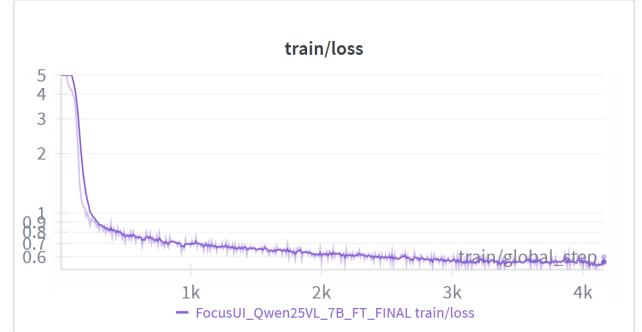
We train FOCUSUI on $8\times$ NVIDIA H200 GPUs using bfloat16 precision, DeepSpeed ZeRO-2 [25], and FlashAttention-2 [10]. The effective batch size per GPU is set to 32 (with gradient accumulation of 4), and the `max_pixels` is set to 5720064, matching GUI-Actor. Training proceeds in two stages:

Stage 1: Saliency Scorer Pre-training. We pretrain the randomly initialized Query-Guided Saliency Scorer for 1 epoch with a learning rate of $1e-4$. This takes about 12 hours for both 3B and 7B models.
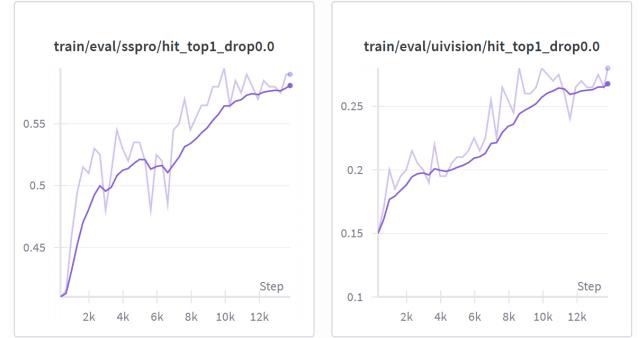
Stage 2: Full Model Fine-tuning. We fine-tune all parameters for 1 epoch with a learning rate of $5e-6$. This takes about 36 hours for the 3B model and about 48 hours for the 7B model.

Hyperparameter Details. During training, we construct per-patch saliency score supervision with $\tau = 2$ and $\lambda = 0.8$. Patch size p is set to 14 and 16 for Qwen2.5-VL and Qwen3-VL based models, respectively. The visual token retention ratio r is uniformly sampled from $(0.1, 1.0)$ for each training sample.
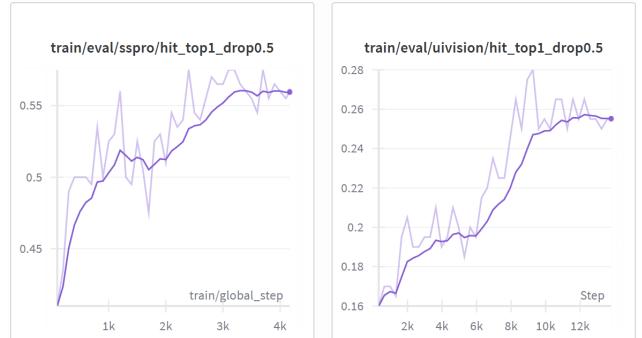
To enable reproducibility, we use the final checkpoint for all obtained FOCUSUI models. We also provide the full Weights & Biases (WandB) logs for all trained models. The training loss and evaluation curves during the training of FOCUSUI-7B are shown in Fig. 6.

(a) **Total Loss** curve during training.

(b) **Evaluation:** ScreenSpot-Pro and UI-Vision with **retention ratio = 100%**.

(c) **Evaluation:** ScreenSpot-Pro and UI-Vision with **retention ratio = 50%**.

Figure 6. WandB loss and evaluation results of FOCUSUI-7B.

A.3. UI Grounding Benchmarks

We evaluate on four public benchmarks containing screenshots paired with instructions: **ScreenSpot-V2** [31], **ScreenSpot-Pro** [15], **OS-World-G** [33], and **UI-Vision** [21]. The statistics of these benchmarks are shown in Tab. 9.

ScreenSpot-V2 [31]. A refined version of ScreenSpot [9] with 1,272 samples across mobile, desktop, and web environments.

ScreenSpot-Pro [15]. This benchmark contains 1,581 samples from 23 professional applications, targeting high-resolution interfaces and complex layouts to test generalization.

OS-World-G [33]. Sampled from OSWorld [32], this benchmark includes 564 samples categorized by task type (text matching, element recognition, layout understanding, fine-grained manipulation, and refusal).

UI-Vision [21]. A desktop-centric benchmark with 5,790 samples from 83 applications, evaluating element grounding, layout grounding, and action prediction.

Benchmark	#Samples	Avg Res.	Max Res.	Platform
ScreenSpot-V2	1272	1725×1657	2880×1800	Hybrid
ScreenSpot-Pro	1581	3267×1727	6016×3384	Desktop
OS-World-G	564	1696×955	1920×1080	Desktop
UI-Vision	5790	1851×1034	3360×2036	Desktop

Table 9. Overview of the evaluation benchmarks used in this work.

B. Discussion

B.1. Visual Redundancy Analysis

Tab. 10 provides the token statistics of **Study 1** (shown in Fig. 1(b) in the main paper). Using the default model settings on the ScreenSpot-Pro evaluation, we find that visual tokens occupy at least 84.3% of the sequence across the studied benchmarks, confirming significant visual redundancy in UI grounding tasks.

Benchmark	Model	#Sys. Tokens	#Vis. Tokens	#Inst. Tokens	Vis. Token %
ScreenSpot-V2	Jedi-1080p	397	2348	4.5	85.4%
	GUI-Actor	90	3506	4.5	97.1%
ScreenSpot-Pro	Jedi-1080p	397	2629	5.2	86.7%
	GUI-Actor	90	5801	5.2	98.1%
OS-World-G	Jedi-1080p	397	2244	21.3	84.3%
	GUI-Actor	90	2244	21.3	95.3%
UI-Vision	Jedi-1080p	397	2249	9.9	84.7%
	GUI-Actor	90	2566	9.9	96.3%

Table 10. Token statistics of **Study 1** shown in Fig. 1(b) in the main paper.

B.2. Position Sensitivity Analysis

Tab. 11 shows the detailed results of **Study 2** (shown in FIg. 1(c) in the main paper), comparing FOCUSUI with UI grounding models integrated with advanced visual token pruning methods.

C. More Experimental Results

C.1. Effective Visual Selection: Patch Recall@K%

We verify the effectiveness of our visual token selection using **Patch Recall@K%**, defined as the fraction of ground-truth (GT) regions captured within the top K% of saliency-

%Ret. Ratio	Model + Pruning Method	SS-V2 Avg	SS-Pro Avg	OSW-G Avg
100%	Qwen2.5-VL-3B	81.5	26.1	27.3
50%	+ Fast-V	43.5	13.9	14.3
	+ HiPrune	80.4	20.3	26.2
	+ Vision-Zip	81.0	21.0	27.1
30%	+ Fast-V	38.6	4.8	14.4
	+ HiPrune	72.0	18.0	20.4
	+ Vision-Zip	75.4	18.9	23.0
100%	Jedi-3B	88.9	36.1	48.8
50%	+ Fast-V	50.3	20.4	25.3
	+ HiPrune	88.3	32.8	46.4
	+ Vision-Zip	88.1	32.9	46.6
30%	+ Fast-V	51.0	14.1	23.9
	+ HiPrune	80.9	26.2	40.4
	+ Vision-Zip	82.8	28.8	41.5
100%	FOCUSUI-3B	91.5	43.8	53.4
50%	+ Full Settings	91.4	42.3	54.6
30%	+ Full Settings	91.0	40.6	51.8

Table 11. Detailed comparison with general visual token pruning methods for **Study 2** shown in Fig. 1(c) in the main paper.

ranked patches (*i.e.*, the top-K visual tokens):

$$\text{Patch Recall@}K\% = \frac{\left|\text{GT positive regions in top } K\%\right|}{\left|\text{Total GT positive regions}\right|}$$

where ground-truth positive regions correspond to the area of UI elements paired with the given instruction. We evaluate $K \in \{5\%, 10\%, 25\%, 50\%\}$. Additionally, we report the **Full Coverage Budget**, the percentage of visual tokens needed to fully cover the ground-truth elements. Results are shown in Tab. 12.

Model	Patch Recall@K% ↑					Full Coverage Budget ↓
	@5%	@10%	@25%	@50%	Avg	
Zero-shot Baselines						
Random	0.05	0.11	0.26	0.51	0.23	0.85
CLIP	0.12	0.21	0.41	0.65	0.35	0.61
Our Query-Guided Saliency Scorer						
FOCUSUI-3B	0.39	0.56	0.83	0.96	0.69	0.25
FOCUSUI-7B	0.43	0.60	0.84	0.97	0.71	0.24

Table 12. **Patch Recall@K%** and **Full Coverage Budget** performance comparison on the ScreenSpot-Pro benchmark.

C.2. Analysis of POSPAD

To better understand the effect of preserving the original spatial layout, we further analyze the placement of POSPAD within contiguous sequences of dropped visual tokens, comparing three variants: (i) *sequence-first*, (ii) *sequence-middle*, and (iii) *sequence-end* (our proposed POSPAD).
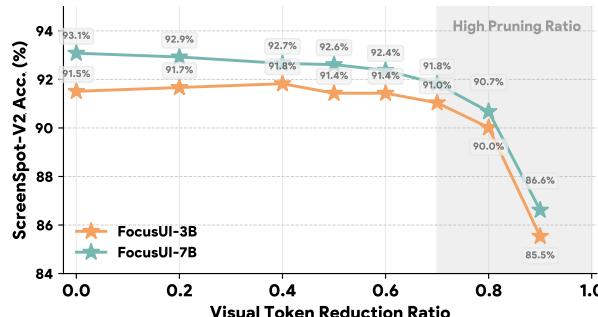
| Sequence Type	SS-Pro Avg.			
	$r = 100\%$	$r = 75\%$	$r = 50\%$	$r = 25\%$
sequence-first	42.0	41.8	39.8	36.8
sequence-middle	41.2	41.1	38.7	33.5
sequence-end (POSPAD)	42.3	42.1	40.4	37.7

Table 13. Different placement of the POSPAD token.
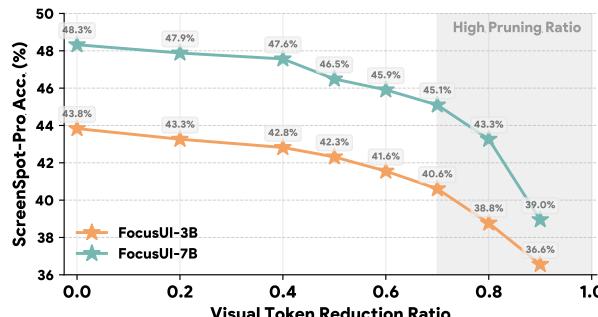
Tab. 13 reports a study that varies the visual token retention ratio r and the location of the POSPAD token. Across all retention ratios, placing POSPAD at the end of each dropped sequence achieves the best performance, especially under low retention ratios. The empirical results confirm our intuition: placing POSPAD at the end of the sequence is more compatible with the raster-scan ordering used by the vision encoder and M-RoPE. In contrast, placing POSPAD at the beginning or in the middle of the sequence pulls the whole region toward earlier positions, making it harder for the LM decoder to align with the original spatial structure.

C.3. Detailed Performance vs. Retention Ratio

Fig. 7 presents the detailed performance of FOCUSUI on ScreenSpot-V2 and ScreenSpot-Pro across varying reduction ratios ($1 -$ retention ratio r). The results demonstrate that FOCUSUI maintains high UI grounding accuracy even with significant visual token reduction.

(a) Performance vs. reduction ratio on ScreenSpot-V2.

(b) Performance vs. reduction ratio on ScreenSpot-Pro.

Figure 7. UI grounding accuracy under different token reduction ratios.

C.4. Qualitative Examples

Fig. 8 visualizes saliency maps on ScreenSpot-V2 and ScreenSpot-Pro, showing that our Query-Guided Saliency Scorer effectively highlights instruction-relevant regions while suppressing the background. We find that for straightforward tasks (Fig. 8 (a, d)), saliency scores peak significantly at the ground-truth locations. In more complex scenarios (Fig. 8 (b, c)), while the scores may be less concentrated, the model still successfully distinguishes potential targets from irrelevant background elements.

D. Prompt Templates

FOCUSUI (with Qwen2.5-VL base model) Below is the system prompt for FOCUSUI-3B and FOCUSUI-7B.

```
You are a GUI agent. Given a screenshot of the
current GUI and a human instruction, your task is
 to locate the screen element that corresponds to
 the instruction. You should output a PyAutoGUI
action that performs a click on the correct
position. To indicate the click location, we will
 use some special tokens, which is used to refer
to a visual patch later. For example, you can
output: pyautogui.click(<your_special_token_here
>).
```

FOCUSUI (with Qwen3-VL base model) Below is the system prompt for FOCUSUI-QWEN3-VL-2B.

```
You are a GUI agent. Your task is to locate the
screen element that corresponds to the
instruction. You should not call any external
tools. Output only the coordinate of one point in
 your response. Format: (x, y)
```
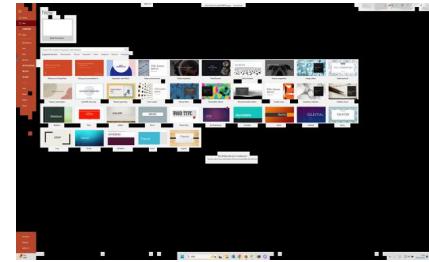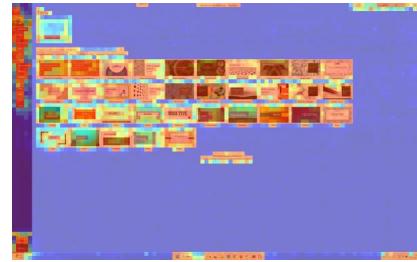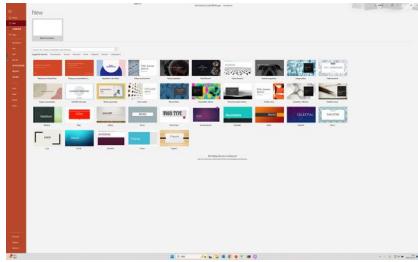
Qwen2.5-VL Below is the system prompt for evaluating Qwen2.5-VL models.

```
You are a GUI agent. Your task is to locate the
screen element that corresponds to the
instruction. You should not call any external
tools. Output only the coordinate of one point in
 your response. Format: (x, y)
```
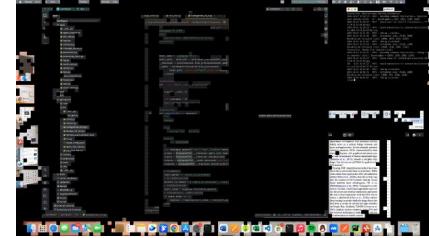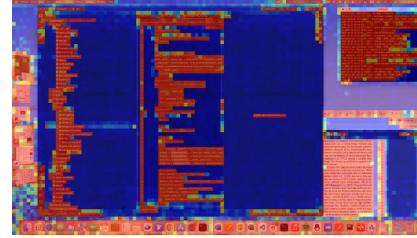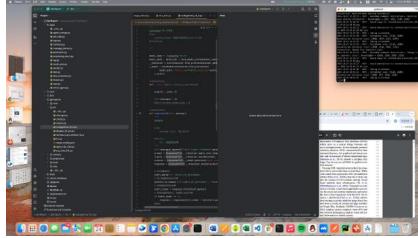
Jedi and Qwen3-VL Below is the system prompt for evaluating Jedi-3B, Jedi-7B, and Qwen3-VL models.

```
# Tools
You may call one or more functions to assist with
 the user query.

You are provided with function signatures within
<tools></tools> XML tags:
<tools>
{"type": "function", "function": {"name": "
computer_use", "description": "Use a mouse to
interact with a computer.\n* The screen's
resolution is {screen_width}x{screen_height}.\n*
Make sure to click any buttons, links, icons, etc
```
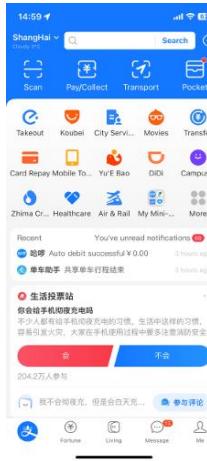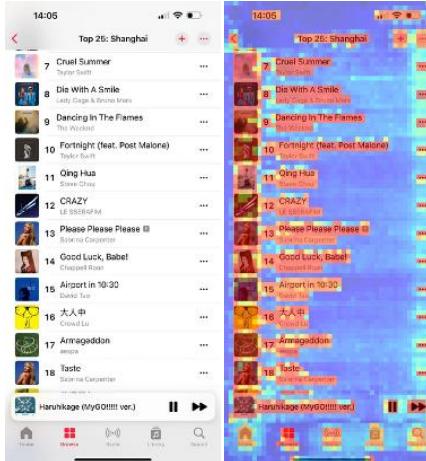
(a) Instruction query: "Create a Psychedelic vibrant presentation."

(b) Instruction query: "View hierarchy."

(c) Instruction query: "Pause the playing music."   *(d) Instruction query: "Click the search bar."*

Figure 8. Qualitative examples of predicted per-patch saliency. **Left:** original screenshot; **Middle:** predicted saliency map; and **Right:** visual token selection results with $r = 30\%$.

```
 with the cursor tip in the center of the element
. Don't click boxes on their edges unless asked.\
n* you can only use the left_click and mouse_move
 action to interact with the computer. if you can
't find the element, you should terminate the
task and report the failure.", "parameters": {"
properties": {"action": {"description": "The
action to perform. The available actions are:\n*
'mouse_move': Move the cursor to a specified (x,
y) pixel coordinate on the screen.\n* 'left_click
': Click the left mouse button with coordinate (x
, y).\n* 'terminate': Terminate the current task
and report its completion status.", "enum": ["
mouse_move", "left_click"], "type": "string"}, "
coordinate": {"description": "(x, y): The x (
pixels from the left edge) and y (pixels from the
```

```
 top edge) coordinates to move the mouse to.
Required only by 'action=mouse_move' and 'action=
left_click'.", "type": "array"}, "status": {"
description": "The status of the task. Required
only by 'action=terminate'.", "type": "string", "
enum": ["success", "failure"]}}, "required": ["
action"], "type": "object"}}}
</tools>

For each function call, return a json object with
 function name and arguments within <tool_call></
tool_call> XML tags:
<tool_call>
{"name": <function-name>, "arguments": <args-json
-object>}
</tool_call>
```

References

[1] Anthropic. Claude 3.7 sonnet system card, 2025. 6, 8

[2] Shuai Bai, Yuxuan Cai, Ruizhe Chen, Keqin Chen, Xionghui Chen, Zesen Cheng, Lianghao Deng, Wei Ding, Chang Gao, Chunjiang Ge, Wenbin Ge, Zhifang Guo, Qidong Huang, Jie Huang, Fei Huang, Binyuan Hui, Shutong Jiang, Zhaohai Li, Mingsheng Li, Mei Li, Kaixin Li, Zicheng Lin, Junyang Lin, Xuejing Liu, Jiawei Liu, Chenglong Liu, Yang Liu, Dayiheng Liu, Shixuan Liu, Dunjie Lu, Ruilin Luo, Chenxu Lv, Rui Men, Lingchen Meng, Xuancheng Ren, Xingzhang Ren, Sibo Song, Yuchong Sun, Jun Tang, Jianhong Tu, Jianqiang Wan, Peng Wang, Pengfei Wang, Qiuyue Wang, Yuxuan Wang, Tianbao Xie, Yiheng Xu, Haiyang Xu, Jin Xu, Zhibo Yang, Mingkun Yang, Jianxin Yang, An Yang, Bowen Yu, Fei Zhang, Hang Zhang, Xi Zhang, Bo Zheng, Humen Zhong, Jingren Zhou, Fan Zhou, Jing Zhou, Yuanzhi Zhu, and Ke Zhu. Qwen3-vl technical report. *arXiv preprint arXiv:2511.21631*, 2025. 2, 5, 7

[3] Shuai Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Sibo Song, Kai Dang, Peng Wang, Shijie Wang, Jun Tang, et al. Qwen2.5-vl technical report. *arXiv preprint arXiv:2502.13923*, 2025. 2, 5, 6, 7, 8

[4] Daniel Bolya, Cheng-Yang Fu, Xiaoliang Dai, Peizhao Zhang, Christoph Feichtenhofer, and Judy Hoffman. Token merging: Your ViT but faster. In *International Conference on Learning Representations*, 2023. 8

[5] Yuxiang Chai, Siyuan Huang, Yazhe Niu, Han Xiao, Liang Liu, Dingyu Zhang, Peng Gao, Shuai Ren, and Hongsheng Li. Amex: Android multi-annotation expo dataset for mobile gui agents. *arXiv preprint arXiv:2407.17490*, 2024. 9

[6] Jikai Chen, Long Chen, Dong Wang, Leilei Gan, Chenyi Zhuang, and Jinjie Gu. V2P: From background suppression to center peaking for robust GUI grounding task, 2025. 6, 8

[7] Liang Chen, Haozhe Zhao, Tianyu Liu, Shuai Bai, Junyang Lin, Chang Zhou, and Baobao Chang. An image is worth 1/2 tokens after layer 2: Plug-and-play inference acceleration for large vision-language models. In *European Conference on Computer Vision*, pages 19–35. Springer, 2024. 7, 8

[8] Wentong Chen, Junbo Cui, Jinyi Hu, Yujia Qin, Junjie Fang, Yue Zhao, Chongyi Wang, Jun Liu, Guirong Chen, Yupeng Huo, Yuan Yao, Yankai Lin, Zhiyuan Liu, and Maosong Sun. GUICourse: From general vision language model to versatile GUI agent. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 21936–21959, 2025. 8

[9] Kanzhi Cheng, Qiushi Sun, Yougang Chu, Fangzhi Xu, Li YanTao, Jianbing Zhang, and Zhiyong Wu. 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, 2024. 8, 9

[10] Tri Dao, Daniel Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. In *Advances in Neural Information Processing Systems*, 2022. 7, 9

[11] Difei Gao, Lei Ji, Zechen Bai, Mingyu Ouyang, Peiran Li, Dongxing Mao, Qinchen Wu, Weichen Zhang, Peiyi Wang, Xiangwu Guo, Hengxu Wang, Luowei Zhou, and Mike Zheng Shou. AssistGUI: Task-oriented desktop graphical user interface automation. *arXiv preprint arXiv:2312.13108*, 2023. 8

[12] Google. Introducing gemini 2.0. Available at: https://blog.google/technology/google-deepmind/google-gemini-ai-update-december-2024, 2024. 6, 8

[13] Boyu Gou, Ruohan Wang, Boyuan Zheng, Yanan Xie, Cheng Chang, Yiheng Shu, Huan Sun, and Yu Su. Navigating the digital world as humans do: Universal visual grounding for GUI agents. In *The Thirteenth International Conference on Learning Representations*, 2025. 6, 8, 9

[14] Jie Huang, Xuejing Liu, Sibo Song, Ruibing Hou, Hong Chang, Junyang Lin, and Shuai Bai. Revisiting multimodal positional encoding in vision-language models, 2025. 3

[15] Kaixin Li, Ziyang Meng, Hongzhan Lin, Ziyang Luo, Yuchen Tian, Jing Ma, Zhiyong Huang, and Tat-Seng Chua. Screenspot-pro: Gui grounding for professional high-resolution computer use. In *Proceedings of the 33rd ACM International Conference on Multimedia*, pages 8778–8786, 2025. 6, 7, 9

[16] Wei Li, William E Bishop, Alice Li, Christopher Rawles, Folawiyo Campbell-Ajala, Divya Tyamagundlu, and Oriana Riva. On the effects of data scale on ui control agents. *Advances in Neural Information Processing Systems*, 37:92130–92154, 2024. 9

[17] Kevin Qinghong Lin, Linjie Li, Difei Gao, Zhengyuan Yang, Shiwei Wu, Zechen Bai, Stan Weixian Lei, Lijuan Wang, and Mike Zheng Shou. ShowUI: One vision-language-action model for GUI visual agent. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 19498–19508, 2025. 4, 6, 8

[18] Jizhihui Liu, Feiyi Du, Guangdao Zhu, Niu Lian, Jun Li, and Bin Chen. HiPrune: Training-free visual token pruning via hierarchical attention in vision-language models, 2025. 7, 8

[19] Yadong Lu, Jianwei Yang, Yelong Shen, and Ahmed Awadallah. Omniparser for pure vision based GUI agent. *arXiv preprint arXiv:2408.00203*, 2024. 6, 9

[20] Lingchen Meng, Jianwei Yang, Rui Tian, Xiyang Dai, Zuxuan Wu, Jianfeng Gao, and Yu-Gang Jiang. DeepStack: Deeply stacking visual tokens is surprisingly simple and effective for LMMs. In *Advances in Neural Information Processing Systems*, 2024. 5

[21] Shravan Nayak, Xiangru Jian, Kevin Qinghong Lin, Juan A Rodriguez, Montek Kalsi, Nicolas Chapados, M Tamer Özsu, Aishwarya Agrawal, David Vazquez, Christopher Pal, et al. UI-Vision: A desktop-centric GUI benchmark for visual perception and interaction. In *Forty-second International Conference on Machine Learning*, 2025. 6, 7, 9, 10

[22] OpenAI. Computer-using agent. Available at: https://openai.com/index/computer-using-agent, 2025. 6, 8

[23] Yujia Qin, Yining Ye, Junjie Fang, Haoming Wang, Shihao Liang, Shizuo Tian, Junda Zhang, Jiahao Li, Yunxin Li, Shijue Huang, et al. UI-TARS: Pioneering automated GUI interaction with native agents. *arXiv preprint arXiv:2501.12326*, 2025. 6, 8

[24] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International Conference on Machine Learning*, pages 8748–8763, 2021. 7, 8

[25] Jeff Rasley, Samyam Rajbhandari, Olatunji Ruwase, and Yuxiong He. Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020. 6, 9

[26] Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024. 3

[27] Fei Tang, Yongliang Shen, Hang Zhang, Siqi Chen, Guiyang Hou, Wenqi Zhang, Wenqiao Zhang, Kaitao Song, Weiming Lu, and Yueting Zhuang. Think twice, click once: Enhancing GUI grounding via fast and slow systems. *arXiv preprint arXiv:2503.06470*, 2025. 8

[28] Peng Wang, Shuai Bai, Sinan Tan, Shijie Wang, Zhihao Fan, Jinze Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, et al. Qwen2-vl: Enhancing vision-language model's perception of the world at any resolution. *arXiv preprint arXiv:2409.12191*, 2024. 3, 8

[29] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. Huggingface's transformers: State-of-the-art natural language processing, 2020. 7

[30] Qianhui Wu, Kanzhi Cheng, Rui Yang, Chaoyun Zhang, Jianwei Yang, Huiqiang Jiang, Jian Mu, Baolin Peng, Bo Qiao, Reuben Tan, et al. GUI-Actor: Coordinate-free visual grounding for GUI agents. *arXiv preprint arXiv:2506.03143*, 2025. 5, 6, 8, 9

[31] Zhiyong Wu, Zhenyu Wu, Fangzhi Xu, Yian Wang, Qiushi Sun, Chengyou Jia, Kanzhi Cheng, Zichen Ding, Liheng Chen, Paul Pu Liang, et al. OS-ATLAS: Foundation action model for generalist GUI agents. In *The Thirteenth International Conference on Learning Representations*, 2024. 6, 7, 8, 9

[32] Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh J Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, et al. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments. *Advances in Neural Information Processing Systems*, 37:52040–52094, 2024. 10

[33] Tianbao Xie, Jiaqi Deng, Xiaochuan Li, Junlin Yang, Haoyuan Wu, Jixuan Chen, Wenjing Hu, Xinyuan Wang, Yuhui Xu, Zekun Wang, Yiheng Xu, Junli Wang, Doyen Sahoo, Tao Yu, and Caiming Xiong. Scaling computer-use grounding via user interface decomposition and synthesis, 2025. 6, 7, 8, 9, 10

[34] Long Xing, Qidong Huang, Xiaoyi Dong, Jiajie Lu, Pan Zhang, Yuhang Zang, Yuhang Cao, Conghui He, Jiaqi Wang, Feng Wu, and Dahua Lin. PyramidDrop: Accelerating your large vision-language models via pyramid visual redundancy reduction, 2025. 8

[35] Yiheng Xu, Zekun Wang, Junli Wang, Dunjie Lu, Tianbao Xie, Amrita Saha, Doyen Sahoo, Tao Yu, and Caiming Xiong. Aguvis: Unified pure vision agents for autonomous GUI interaction. *arXiv preprint arXiv:2412.04454*, 2024. 6, 8

[36] Senqiao Yang, Yukang Chen, Zhuotao Tian, Chengyao Wang, Jingyao Li, Bei Yu, and Jiaya Jia. Visionzip: Longer is better but not necessary in vision language models. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 19792–19802, 2025. 7, 8

[37] Yuhao Yang, Yue Wang, Dongxu Li, Ziyang Luo, Bei Chen, Chao Huang, and Junnan Li. Aria-UI: Visual grounding for GUI instructions. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 22418–22433, 2025. 8

[38] Bofei Zhang, Zirui Shang, Zhi Gao, Wang Zhang, Rui Xie, Xiaojian Ma, Tao Yuan, Xinxiao Wu, Song-Chun Zhu, and Qing Li. TongUI: Building generalized GUI agents by learning from multimodal web tutorials. *arXiv preprint arXiv:2504.12679*, 2025. 6, 8

[39] Ce Zhang, Kaixin Ma, Tianqing Fang, Wenhao Yu, Hongming Zhang, Zhisong Zhang, Yaqi Xie, Katia Sycara, Haitao Mi, and Dong Yu. VScan: Rethinking visual token reduction for efficient large vision-language models, 2025. 8

[40] Qizhe Zhang, Aosong Cheng, Ming Lu, Zhiyong Zhuo, MinQi Wang, Jiajun Cao, Shaobo Guo, Qi She, and Shanghang Zhang. [cls] attention is all you need for training-free visual token pruning: Make vlm inference faster. *arXiv preprint arXiv:2412.01818*, 2024. 8

[41] Yuan Zhang, Chun-Kai Fan, Junpeng Ma, Wenzhao Zheng, Tao Huang, Kuan Cheng, Denis A. Gudovskiy, Tomoyuki Okuno, Yohei Nakata, Kurt Keutzer, and Shanghang Zhang. SparseVLM: Visual token sparsification for efficient vision-language model inference. In *International Conference on Machine Learning*, 2025. 8

[42] Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, et al. Webarena: A realistic web environment for building autonomous agents. *arXiv preprint arXiv:2307.13854*, 2023. 8