AQUAUI: VISUAL TOKEN REDUCTION FOR GUI AGENTS WITH ADAPTIVE QUADTREES

Anonymous authors
Paper under double-blind review

ABSTRACT

Large Multimodal Models (LMMs) have recently emerged as promising backbones for Graphical User Interfaces (GUIs) agent models, where high-resolution GUI screenshots are introduced to the prompts at each iteration step. However, these screenshots exhibit highly non-uniform spatial information density: large regions may carry little information and are visually homogeneous, while key text and icons may require high visual fidelity. Existing approaches to this problem either require additional training or rely on attention-based token compression, ignoring the structured layout and spatial redundancy of GUI screenshots. To fill the gap, this paper proposes **AQUAUI**, a training-free inference-time token reduction method for GUI agent models that utilizes the non-uniform information density in screenshots. AQuaUI constructs an adaptive *quadtree* on each screenshot input and keeps one representative merged token per leaf of the quadtree. AQuaUI preserves the spatial positions of retained tokens throughout the pipeline to ensure that all position-encoding stages remain consistent. To further improve temporal consistency across multi-step GUI interactions, we propose a *conditional quadtree* algorithm that leverages the continuity between consecutive screenshots within a single request. Specifically, it refines the current quadtree using previous quadtrees as references, helping preserve fine-grained regions across static or mildly shifted GUI states. We implement AQuaUI on state-of-the-art GUI agent models and conduct experiments on standard grounding and navigational benchmarks. AQuaUI consistently shows improved accuracy–efficiency trade-offs over prior baselines. Notably, on GUI-Owl-1.5-32B-Instruct, AQuaUI reduces average latency by **13.10%** with **29.52%** fewer visual tokens while retaining **99.06%** of full-token performance, suggesting that the spatial redundancy of GUI screenshots can be exploited at inference time without retraining.

1 INTRODUCTION

With the rapid development of Large Multimodal Models (LMMs), current state-of-the-art models have gained the ability to understand and navigate through complex UI elements in Graphical User Interfaces (GUIs). This, combined with research progress within autonomous agents, has led to various GUI agents designed to interpret user instructions and autonomously perform complex tasks. As a result, GUI agents have emerged as a new research focus of agentic AI.

Earlier GUI agents often adopt a hybrid way that combines textual representations, such as accessibility trees or HTML, with visual information (Deng et al., 2023; Zhou et al., 2023; Wang et al., 2024a; Lu et al., 2024). Such systems, however, often face cross-platform adaptability and require frequent task-specific optimization. In response to these limitations, native GUI agent models that directly take screenshots as input (Xu et al., 2024; Anthropic, 2024; Lin et al., 2025a) have become increasingly popular as they demonstrate competent performance on trending GUI agent benchmarks (Xie et al., 2025; Rawles et al., 2024). Screenshots provide a natural perspective of understanding GUIs and a unified solution in cases where accessibility trees are unavailable. The end-to-end design also offers a data-centric view that allows model performance to improve through data scaling and iterative feedback.

Such GUI agent models typically need to process high-resolution screenshots at every interaction step, resulting in a massive number of visual tokens even with merging and patching techniques.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

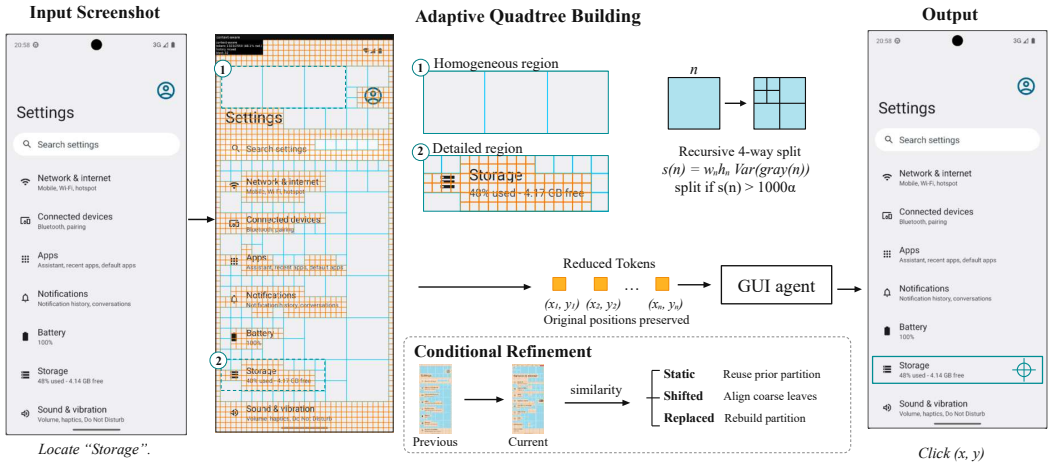

Figure 1: An overview of the entire pipeline of AQuaUI. Given a GUI screenshot, AQuaUI will first build an adaptive quadtree to capture the UI layout. For each leaf node in that quadtree, all but one representative tokens are discarded. These representative tokens are sent to the language model with their position information for the final output. Additionally, we design a conditional quadtree refinement algorithm that reuses previous screenshots when building the quadtree to remain token consistency.

For example, a typical Pixel 6 phone screenshot with a resolution of 2400×1080, when processed with Qwen3-VL (Bai et al., 2025), can produce around 2600 tokens. This substantially increases computational cost and limits the amount of conversation history that can be retained within the context window. Token compression, therefore, has become increasingly important for GUI agents, as it can reduce inference cost, improve throughput, and enable longer interaction histories.

Recent research observes that GUI screenshots contain substantially more low-information regions than natural images, while also exhibiting simpler and more regular spatial structures (Huang et al., 2025; Lin et al., 2025a). Previous works of token compression in Vision Language Models (VLMs) target natural images and are therefore not well suited to GUI images (Chen et al., 2024; Jiang et al., 2025; Zhang et al., 2025). Other works that focus on GUI images develop token reduction methods as part of GUI agent model training (Lin et al., 2025a; Ouyang et al., 2026), leaving open whether they can be directly applied to arbitrary GUI agent models without training.

To fill this gap, we propose AQuaUI, a training-free framework that can be easily applied to any downstream LMMs. We use *quadtree* (Hunter & Steiglitz, 1979) to select certain tokens before they are sent to the text transformer. The pipeline is illustrated in Fig. 1.

We evaluate AQuaUI on grounding benchmarks including UI-Vision (Nayak et al., 2025), ScreenSpot-Pro (Li et al., 2025), ScreenSpot-V2 (Wu et al., 2024), OSWorld-G (Xie et al., 2025) and MMBench-GUI (Wang et al., 2025). For navigational evaluation, we adopt AndroidWorld (Rawles et al., 2024) and AndroidControl (Li et al., 2024a). AQuaUI can outperform prior baselines with better accuracy-latency trade-off, especially with larger models. On Qwen3-VL models, AQuaUI can compress the visual tokens by 30% without hurting overall performance.

In summary, the main contribution of our paper can be summarized as follows:

- We propose AQuaUI, a training-free inference-time visual-token reduction method that exploits the structural layout of GUI screenshots using an adaptive quadtree.
- We introduce a conditional quadtree algorithm that uses previous screenshots within the same request to preserve fine-grained partitions across static or mildly shifted screenshots.
- We implement AQuaUI on various state-of-the-art GUI agent models and evaluate it on standard GUI grounding and navigation benchmarks, showing improved accuracy–efficiency trade-offs over prior baselines.

2 Related Work

2.1 GUI Agents

Recent work on GUI agents has rapidly expanded both the scale of backbone models and the range of supported platforms. UI-TARS (Qin et al., 2025) and GUI-Owl (Ye et al., 2025) train LMMs on screen-grounded trajectories, while UI-Voyager (Lin et al., 2026), MAI UI (Zhou et al., 2025), and ClawGUI (Tang et al., 2026) explore complementary directions in planning, multi-agent coordination, and tool use. Despite various differences, these systems share a common serving bottleneck: each interaction step introduces high-resolution screenshots whose visual tokens can dominate prefill latency, attention cost, and KV-cache memory.

Several recent methods recognize that UI screenshots require specialized treatment. ShowUI (Lin et al., 2025a) reduces visual tokens by leveraging UI structural regularity, while FocusUI (Ouyang et al., 2026) selects instruction-relevant UI patches and introduces a position-aware strategy to mitigate the loss of positional continuity. These methods highlight the importance of spatial structure for UI grounding, but their main results are obtained in model-specific training settings. In contrast, AQuaUI targets training-free deployment on off-the-shelf models.

2.2 Compression Methods for LMMs

Token Pruning. A broad line of work reduces the number of visual tokens by estimating token importance from attention and similarity. FastV (Chen et al., 2024) prunes visual tokens in deep layers using attention from the language stream; G-Prune (Jiang et al., 2025) models token importance through graph propagation; and SimIgnore (Zhang et al., 2025) uses cross-modal similarity to discard redundant patches. Recent generic pruners such as HiPrune (Liu et al., 2026) and VisionZip (Yang et al., 2025) further improve training-free visual-token reduction. These methods primarily frame reduction as token-importance estimation. GUI screenshots introduce a different situation where low-attention tokens may still be important because they define labels or coordinate context. Equally importantly, such methods rely on explicit calculation of attention maps, which could add extra complexity to FlashAttention (Dao et al., 2022) or KV Cache management.

KV-cache compression. Another line of work compresses the KV cache after tokens have already entered the transformer. SnapKV (Li et al., 2024b) selects KV entries using attention from an observation window, while PyramidKV (Cai et al., 2024) allocates different cache budgets across different layers. Subsequent methods improve selection with diversity terms, pre-RoPE scores, or mixed policies (Cai et al., 2025; Mao et al., 2026). GUI KV (Huang et al., 2025) specifically calculates the spatial and temporal scoring of KV cache based on GUI screenshots. These techniques are largely orthogonal to AQuaUI: they reduce memory and decoding cost after visual tokens have been encoded, whereas AQuaUI reduces the visual token sequence itself before KV-cache is computed.

3 Method

In this section, we formalize the problem in §3.1, then present the basic token reduction method for AQuaUI in §3.2. Building a conditional quadtree using previous images is discussed in §3.3.

3.1 Task Definition

At each step t, a GUI agent receives a screenshot s_t and a text prompt p_t containing the user instruction and interaction history. A native GUI agent model produces a response and action $(r_t, a_t) = \mathcal{M}(p_t, \{s_i\}_{i=0}^{t})$. Each screenshot s_t is encoded as a sequence of visual tokens tok_{vis} using a vision encoder, which are processed jointly with text tokens tok_{text}. The goal is to find a different set of tokens tok_{comp} whose length $|\text{tok}_{\text{comp}}| < |\text{tok}_{\text{vis}}|$ while preserving task performance.

3.2 Adaptive Quadtree Tokenization

For GUI screenshots, information is particularly non-uniformly distributed (Lin et al., 2025a), and we propose a *quadtree* algorithm inspired by previous work (Hunter & Steiglitz, 1979; Chickering et al., 2025). Unlike QLIP (Chickering et al., 2025), which replaces the fixed patchification of CLIP-based

MLLMs with a quadtree for natural images, AQuaUI prunes tokens on the native merged-token grid of GUI agents, keeps each retained token at its original position, and conditions the partition on previous screenshots. A quadtree is a hierarchical data structure that recursively partitions a surface into four quadrants until a stopping criterion is satisfied. In image processing, quadtrees have been widely used as compact image representations: homogeneous regions are represented by large leaves, while visually complex regions are recursively divided into smaller cells. This property makes quadtrees particularly suitable for GUI screenshots with large background panels, margins as well as small concentrated regions with text and icons. A quadtree can therefore adapt its partition to the information density of the screenshot, using coarse leaves for redundant regions and fine leaves for detail-rich regions.

Based on this observation, AQuaUI first builds an adaptive quadtree, and then uses its leaves as adaptive visual-token units, which we describe below. The detailed algorithm and implementation is discussed in Appx. §A.

Grid Alignment and Boundary Handling Since dimensions of GUI screenshots are not necessarily powers of two, which is a precondition of applying quadtree, AQuaUI therefore first decomposes the image into two parts: a centered region that can be tiled by square chunks of size $C \times C$, and boundary margins that cannot be included in this chunk layout, which are kept untouched. AQuaUI then runs an adaptive quadtree independently inside each chunk.

Splitting Criteria For a quadtree node n of size $w_n \times h_n$, the default split criterion is the area-weighted grayscale variance $s(n) = w_n h_n \cdot \mathrm{Var}(\mathrm{grey}(n))$, and we recursively split until $s(n) \leq 1000 \cdot \alpha$.

The implementation also supports a gradient-based criterion, where the score function s gives the maximum magnitude of the gradient of node n, which can be useful when the edges provide a stronger signal under certain circumstances. We will discuss the choice of different criteria in §5.2

Representative Token Selection For each final leaf, AQuaUI selects the representative token block. By default, this is the center block ($x' = \lfloor \frac{x_0+x_1}{2} \rfloor, \quad y' = \lfloor \frac{y_0+y_1}{2} \rfloor$) if the leaf spans block coordinates from x_0 to x_1 and y_0 to y_1.

3.3 Conditional Quadtree Building

GUI trajectories often contain adjacent screenshots that are nearly identical. A page may remain static, a menu may pop up, or a scroll operation may shift content by a few token blocks. If each screenshot is compressed independently, a later frame may discard details that were preserved in an earlier frame, which can weaken the reasoning and grounding ability.

AQuaUI introduces a conditional quadtree mechanism that infers history from the ordered images in the same request. It is a heuristic algorithm that builds quadtree conditioned on the state change of each chunk in one of three modes: static, shifted, or replaced.

Static mode We begin by identifying patterns that most pixels remain the same, which we call the Static mode. We define a similarity value between two chunks. Consider the chunk AQuaUI produced previously, we summarize it into a condensed grid signature S of size (H_c, W_c) by computing the greyscale of each block of size (b, b) so that each cell of S stores the rounded mean grayscale of the corresponding $b \times b$ block. (H_c, W_c) denotes the number of blocks within a chunk.

We define similarity of two adjacent signatures S_t and S_{t-1} as following:

$$\text{similarity}(S_t, S_{t-1}) = 1 - \frac{1}{255 H_c W_c} \times \sum_{i,j} |S_t(i,j) - S_{t-1}(i,j)|$$

A region is static if similarity $\geq \tau_{static}$.

Shifted mode If two chunks are not static, we then consider if a scroll operation happens. We call this the Shifted mode. To capture the scrolling feature, we add a block-level shift $\delta = (\Delta i, \Delta j)$ in

the previous definition and change to average over the overlapping area. We search over $|\Delta i|, |\Delta j|$ in certain range and classify the region as shifted if the best valid shift $\delta^\star$ satisfies

$$\mathrm{sim}(\delta^\star) \geq \tau_{\mathrm{shift}} \quad \text{and} \quad \mathrm{sim}(\delta^\star) \geq \mathrm{sim}(S_t, S_{t-1}) + \gamma.$$

Replaced mode Regions satisfying neither condition are treated as replaced. A new quadtree is constructed where no priors are used.

We use $\tau_{static} = 0.97$, $\tau_{shift} = 0.94$, $\gamma = 0.03$, and a shift search range of $|\Delta i|, |\Delta j| \leq 4$ blocks, i.e., a static region changes by at most 7.7 gray levels per block on average, and a shifted one by at most 15.3 levels after alignment. These values were fixed before any navigation experiment and are shared by all models and benchmarks. Since the prior can only split leaves of the independently built quadtree, a wrong mode decision can only add tokens; Appx. §A.4.2 shows that perturbing the thresholds adds at most 2.2% tokens, and only τ_{static} changes the input of more than 2% of screenshot pairs. The full algorithm can be seen in Alg. 1 (Appx. §A.4).

4 Experiment and Results

We introduce the basic setup of the experiment in §4.1, and then compare AQuaUI with the baseline in §4.2. We continue to explore the generalization of AQuaUI on different models with Qwen3-VL architecture. Navigational experiments using AndroidControl and AndroidWorld are discussed in §4.3.

4.1 Experiment Setup

Benchmarks. To evaluate the performance of AQuaUI, we evaluate it on multiple benchmarks. Thanks to ClawGUI's (Tang et al., 2026) effort on reproducibility of benchmarking GUI agent models, we reuse the majority of its experiment setting, which consists of various grounding benchmarks including ScreenSpot-Pro (Li et al., 2025), ScreenSpot-V2 (Wu et al., 2024), OSWorld-g (Xie et al., 2025), UI-Vision (Nayak et al., 2025), MMBench-GUI (Wang et al., 2025), and one offline navigational benchmark, AndroidControl (Li et al., 2024a). For online navigational benchmark, we choose AndroidWorld (Rawles et al., 2024). We use grounding benchmarks as the main evaluation because they provide controlled, reproducible measurement, and report navigation benchmarks as complementary interactive evaluations.

Models. We evaluate AQuaUI on the Qwen2-VL (Wang et al., 2024b) series and the ShowUI (Lin et al., 2025a) series for a direct comparison with SHOWUI. Additionally, to examine the generalization ability, we evaluate several series of SOTA GUI agent model such as the GUI-Owl-1.5 (Xu et al., 2026), the MAI-UI (Zhou et al., 2025) and the Qwen3-VL (Bai et al., 2025) series. These models share the same backbone model as Qwen3-VL but differs with different training strategy.

Metrics. For grounding benchmarks and AndroidControl, we report click accuracy (Acc.). For AndroidWorld, we report success rate (SR). Latency is the mean end-to-end time per sample, including preprocessing and quadtree construction.

Baseline. Our primary baseline is ShowUI's UI-guided token selection, the closest GUI-specific visual-token reduction method with available implementation. The model choice is limited by SHOWUI's implementation, as no recent implementation on Qwen3-VL series is provided. We also include a random reduction baseline, FastV and VTW (Lin et al., 2025b) in §5.1 to test whether AQuaUI actually benefits from quadtree structure.

4.2 Main Results

Tab. 1 reports the performance and latency differences between AQuaUI and SHOWUI for five grounding benchmarks. Tab. 2 reports the result of AQuaUI on other SOTA GUI agent models. The detailed breakdown performance and failure analysis is reported in detail in Appx. §B. We make the following observations:

AQuaUI preserves task accuracy better at comparable compression rates. At similar compression rates, AQuaUI consistently produces smaller average accuracy degradation than SHOWUI's UI-guided visual token selection. On Qwen2-VL-7B-Instruct, AQuaUI reduces tokens by 26.68% and drops

Table 1: Accuracy and latency across GUI benchmarks compared with SHOWUI's UI Guided Token Selection. UI_mask_ratio is set to default 0.5. Accuracy and compression are reported in percentages; latency is reported in seconds. SS-Pro: ScreenSpot-Pro; SS-v2: ScreenSpot-V2; OSW-G: OSWorld-G; UI-V: UI-Vision; MMB-GUI: MMBench-GUI.

Backend	SS-Pro		SS-V2		OS-G		UI-VISION		MMB-GUI		Comp.	Avg. Δ	
	Acc.	Lat.	Acc.	Lat.	Acc.	Lat.	Acc.	Lat.	Acc.	Lat.		Acc.	Lat.
Qwen2-VL-7B-Instruct													
Transformers	10.31	1.10	67.30	1.03	21.14	1.25	3.41	1.17	39.57	1.32	0	–	–
+SHOWUI	8.60	1.48	55.11	1.28	16.87	1.61	2.72	1.42	28.69	1.62	27.55	-5.95	+0.31
vLLM	10.44	1.87	67.14	1.29	19.36	1.48	3.34	1.38	39.90	1.57	0	–	–
+AQuaUI	7.34	1.96	56.68	1.27	18.12	1.45	2.72	1.23	29.33	1.54	26.68	**-5.20**	**-0.03**
ShowUI-2B													
Transformers	7.53	0.74	76.65	0.61	18.29	0.64	5.97	0.65	43.49	0.66	0	–	–
+SHOWUI	5.95	0.78	65.72	0.71	16.34	0.72	5.04	0.66	33.67	0.76	27.78	-5.04	**+0.07**
vLLM	7.59	1.18	76.49	0.53	18.12	0.45	6.06	0.44	44.05	0.64	0	–	–
+AQuaUI	6.58	1.35	64.07	0.62	16.34	0.52	5.84	0.55	36.00	0.70	28.24	**-4.70**	+0.10

accuracy by 5.20 points on average compared to SHOWUI's 5.95-point drop at 27.55% compression. A similar trend is observed on ShowUI-2B with a slightly higher compression rate and a smaller average accuracy drop.

Token reduction introduces noticeable overhead under Qwen2-VL implementations. On ShowUI-2B, both SHOWUI and AQuaUI introduce additional latency, despite reducing the number of visual tokens. This suggests that the overhead of token-selection logic on Qwen2-VL can not be neglected, especially for smaller models where the full-token inference cost is already low. On Qwen2-VL-7B-Instruct, SHOWUI adds 0.31s on average, whereas AQuaUI does not increase the average latency (-0.03s) because its pre-ViT reduction also shortens the vision-encoder input, indicating AQuaUI is more efficient when serving.

AQuaUI preserves accuracy consistently on Qwen3-VL. Across the Qwen3-VL series, AQuaUI reduces around 30% of visual tokens while incurring less than one point of average accuracy degradation. The average drops are only 0.61, 0.48, and 0.85 points for Qwen3-VL-2B, 8B, and 32B, respectively. The larger drop on Qwen2-VL mainly stems from our speed-oriented pre-ViT default, under which the vision encoder only sees a sparse patch sequence (Appx. §A.6). With post-ViT reduction, the average drop on Qwen2-VL-7B-Instruct narrows from 5.20 to 3.28 points at only +0.02s, still clearly ahead of SHOWUI (-5.95 points, $+0.31$s).

Latency gains emerge as model size increases. Although token reduction introduces additional overhead, the savings become visible for larger backbones. For Qwen3-VL-8B and Qwen3-VL-32B, AQuaUI reduces average latency by 0.09s and 0.23s. Similar trends appear on other larger models. In contrast, the 2B models show little latency changes. This suggests that inference-time visual-token reduction is especially valuable for larger GUI agents, where the language model computation dominates the reduction overhead.

4.3 NAVIGATIONAL RESULTS

Tabs. 3 and 4 evaluate AQuaUI on AndroidControl and AndroidWorld. For AndroidWorld, random seed is fixed to 30. For MAI-UI-8B's result on AndroidWorld, we report our reproduced dense vLLM baseline under the same evaluation protocol used for AQuaUI.

AQuaUI remains stable on AndroidControl. On AndroidControl, AQuaUI preserves performance under roughly 29% visual token compression. For Qwen3-VL-2B-Instruct, the average accuracy drop across high- and low-level instructions is only 0.56 points. This indicates that the quadtree reduction remains effective beyond static grounding benchmarks and transfers to navigational settings.

Interactive long-horizon evaluation is more sensitive. On AndroidWorld, AQuaUI also maintains most of the dense model performance, but the degradation is larger than on AndroidControl. UI-

Table 2: Accuracy and latency across GUI benchmarks for various GUI agent models. Accuracy and compression are reported in percentages; latency is reported in seconds. SS-Pro: ScreenSpot-Pro; SS-V2: ScreenSpot-v2; OSW-G: OSWorld-G; UI-V: UI-Vision; MMB-GUI: MMBench-GUI.

Backend	SS-Pro		SS-V2		OS-G		UI-VISION		MMB-GUI		Comp.	Avg. Δ	
	Acc.	Lat.	Acc.	Lat.	Acc.	Lat.	Acc.	Lat.	Acc.	Lat.		Acc.	Lat.
Qwen3-VL-2B-Instruct													
vLLM	43.96	1.17	88.84	0.43	48.49	0.31	15.24	0.38	73.34	0.50	0	–	–
+AQuaUI	42.88	1.17	88.84	0.46	48.49	0.31	15.06	0.41	71.56	0.52	29.52	-0.61	+0.02
Qwen3-VL-8B-Instruct													
vLLM	56.23	1.95	94.26	0.74	59.68	0.52	27.78	0.63	84.31	0.86	0	–	–
+AQuaUI	55.28	1.67	94.26	0.71	59.33	0.52	27.72	0.58	83.28	0.77	29.72	-0.48	-0.09
Qwen3-VL-32B-Instruct													
vLLM	58.57	3.24	94.89	1.29	66.25	0.97	36.17	1.35	87.15	1.51	0	–	–
+AQuaUI	56.17	2.44	95.05	1.17	64.30	1.03	35.96	1.23	87.31	1.33	29.52	-0.85	-0.23
MAI-UI-2B													
vLLM	58.89	1.25	92.45	0.50	55.24	0.37	29.99	0.41	82.64	0.57	0	–	–
+AQuaUI	57.50	1.25	91.43	0.51	55.24	0.36	29.57	0.43	81.44	0.58	29.52	-0.81	+0.01
MAI-UI-8B													
vLLM	64.52	2.15	94.81	1.04	64.12	0.86	40.45	0.94	88.62	1.20	0	–	–
+AQuaUI	59.14	1.88	94.18	0.97	63.59	0.85	39.82	0.89	87.51	1.09	29.25	-1.66	-0.10
GUI-Owl-1.5-8B-Instruct													
vLLM	68.88	2.69	93.40	0.88	67.14	0.65	37.27	0.92	82.61	1.08	0	–	–
+AQuaUI	67.68	2.21	93.32	0.85	64.65	0.66	36.72	0.83	81.86	0.97	29.52	-1.01	-0.14
GUI-Owl-1.5-32B-Instruct													
vLLM	70.97	3.30	94.42	1.28	66.43	0.95	37.45	1.29	87.15	1.50	0	–	–
+AQuaUI	69.89	2.77	94.03	1.16	65.90	0.89	36.72	1.11	86.53	1.30	29.52	-0.67	-0.22

Table 3: Accuracy for various GUI agent models on AndroidControl.

Backend	High		Low		Comp.	Avg. Δ	
	Acc.	Lat.	Acc.	Lat.		Acc.	Lat.
Qwen3-VL-2B-Instruct							
vLLM	54.80	0.67	73.83	0.66	0	–	–
+AQuaUI	54.14	0.78	73.37	0.73	29.33	-0.56	+0.09
Qwen3-VL-8B-Instruct							
vLLM	56.97	1.28	66.26	1.23	0	–	–
+AQuaUI	56.43	1.21	66.32	1.18	29.33	-0.24	-0.06

Voyager drops from 70.69% to 68.10%, while MAI-UI-8B drops from 57.76% to 56.03%. This is expected because AndroidWorld evaluates multi-step interaction, where small grounding or perception errors may accumulate over the trajectory. Nevertheless, the results suggest that AQuaUI can reduce visual token load while retaining competitive navigation performance.

Conditional quadtree building consistently helps. Removing it lowers the success rate of both models by 3 of 116 tasks (2.6 points) at nearly identical compression: from 68.10% to 65.52% for UI-Voyager and from 56.03% to 53.45% for MAI-UI-8B. This indicates that the gain is not simply due to retaining more tokens. Rather, conditional quadtree refinement helps preserve fine-grained partitions across adjacent screenshots.

Table 4: Success rate on AndroidWorld. Random seed is fixed to 30. Accuracy and compression are reported in percentages.

Backend	SR.	Comp.
UI-Voyager		
vLLM	70.69	–
+AQuaUI	68.10	31.90
- Conditional Quadtree	65.52	32.06
MAI-UI-8B		
vLLM	57.76	–
+AQuaUI	56.03	31.89
- Conditional Quadtree	53.45	32.06

Table 5: Grounding accuracy (%). Random dropping removes 30% of visual tokens (after the vision encoder for Qwen2-VL-7B-Instruct). FastV's retention ratio after the fourth layer (on average 52% for Qwen2-VL and 42% for Qwen3-VL) and VTW's withdrawal layer are chosen per benchmark to approximately match AQuaUI's vision-encoder and LLM-prefill TFLOPs per image.

	SS-Pro	SS-V2	OS-G	UI-Vision	MMB-GUI
Qwen2-VL-7B-Instruct					
vLLM	10.44	67.14	19.36	3.34	39.90
+Random Reduction	5.44	57.47	15.63	2.65	29.60
+FastV	8.16	58.25	14.56	2.43	30.30
+VTW	1.08	3.38	17.23	0.09	2.28
+AQuaUI	7.34	56.68	18.12	2.72	29.33
Qwen3-VL-8B-Instruct					
vLLM	56.23	94.26	59.68	27.78	84.31
+Random Reduction	39.47	86.95	46.18	21.15	74.21
+FastV	28.91	90.25	56.66	22.61	73.01
+VTW	0.13	5.11	58.08	0.46	2.48
+AQuaUI	55.28	94.26	59.33	27.72	83.28

5 Ablation Studies

In this section, we compare AQuaUI with other visual token compression methods in §5.1, then discuss the choice of hyper-parameter α in §5.2.

5.1 AQuaUI Preserves UI Layout

We compare AQuaUI with random dropping, a surprisingly strong baseline (Peng et al., 2025), FastV (Chen et al., 2024), and VTW (Lin et al., 2025b), using PACT (Dhouib et al., 2025) implementation. Since FastV retains the full vision-encoder and early-layer costs, it prunes at the fourth transformer layer and keeps on average 52% (Qwen2-VL) and 42% (Qwen3-VL) of visual tokens, with the ratio chosen per benchmark to approximately match AQuaUI's TFLOPs per image. For VTW, we choose per benchmark the shallowest withdrawal layer whose TFLOPs are at least those of AQuaUI, i.e., layers 14–20 on Qwen2-VL-7B and 10–26 on Qwen3-VL-8B.

Tab. 5 shows that AQuaUI outperforms random dropping on all five benchmarks with Qwen3-VL-8B. VTW performs worse than AQuaUI on all five benchmarks for both models, and so does FastV on Qwen3-VL-8B, where AQuaUI exceeds FastV by 4.01 and 26.37 points on ScreenSpot-V2 and ScreenSpot-Pro. On Qwen2-VL, AQuaUI outperforms FastV by 3.56 and 0.29 points on OSWorld-G and UI-Vision, but trails it by 0.82–1.57 points on the other three benchmarks. It is worth noting that VTW underperforms on most benchmarks but achieves stronger results on OSWorld-G. VTW is designed to select its withdrawal layer by KL divergence rather than under a fixed compute budget; under matched TFLOPs, the layer is instead determined by how strongly AQuaUI compresses each benchmark. The matched budget of OSWorld-G permits a much deeper withdrawal layer (layer 20 on Qwen2-VL-7B and layer 26 on Qwen3-VL-8B) that retains more graphical information.

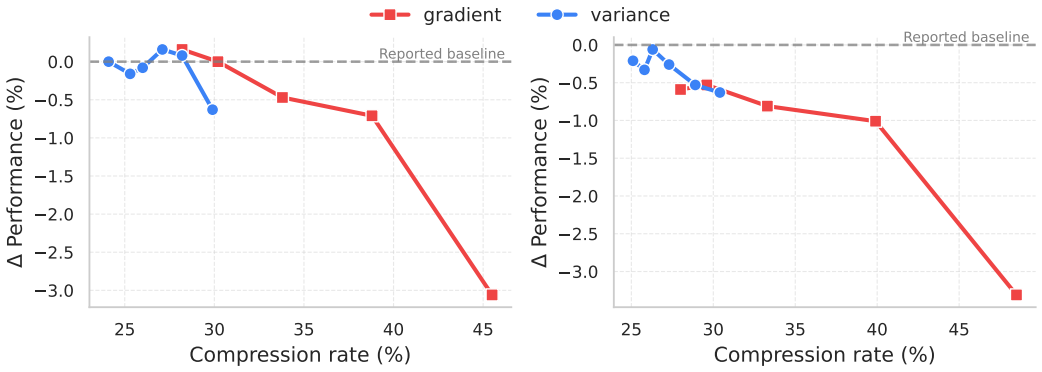

Figure 2: Performance change with respect to different parameter and score function on MAI-UI-8B. Left: ScreenSpot-V2. Right: UI-Vision. Each dot represents a choice of α. From left-to-right, for variance-based score function, $\alpha = \{1, 4, 8, 16, 32, 64\}$ and for gradient-based score function, $\alpha = \{10, 15, 30, 60, 120\}$.

5.2 Choice of Compression Parameter and Score Function

We select $\alpha = 64$ through a small pilot sweep and fix it across models and benchmarks in the main experiments. Fig. 2 examines other thresholds and score functions.

Variance-based compression is more stable, while gradient-based compression is more aggressive. Across ScreenSpot-V2 and UI-Vision, the variance-based strategy yields a relatively narrow compression range and preserves performance consistently. In contrast, the gradient-based strategy reaches substantially higher compression rates, but its performance degrades more noticeably under aggressive compression. This suggests a clear stability–compression trade-off between the two strategies.

Appropriate compression can preserve performance with fewer visual tokens. The results show that a properly chosen α can reduce visual tokens while maintaining comparable performance. In particular, moderate compression often incurs only a small accuracy change. This indicates that the model tolerates a meaningful amount of visual-token reduction, but excessive compression removes task-relevant visual details.

6 Conclusion

We presented AQuaUI, a training-free inference-time visual-token reduction method for GUI agent models. Instead of treating visual-token reduction as a generic token-importance estimation problem, AQuaUI exploits the spatial structure of GUI screenshots through adaptive quadtree partitioning. Homogeneous regions are represented with coarse leaves, whereas other areas retain finer leaves with each leaf represented by one visual token. We further introduced a conditional quadtree refinement mechanism that leverages temporal continuity of screenshots, reusing previous partitions only when the current screen region is static or mildly shifted and falling back to reconstruction of a quadtree when the content changes.

Across GUI grounding and navigation benchmarks, AQuaUI consistently reduces visual tokens while preserving accuracy. The results show that spatial redundancy in GUI screenshots can be exploited effectively at inference time, especially for larger GUI agent backbones. More broadly, our findings suggest that efficient GUI perception should not merely ask which tokens are important, but also where the tokens are located.

AI USE STATEMENT

LLM is used to improve the writing of this paper and debug certain vLLM implementation bugs. The core method development in this paper does not involve LLMs as an important component. The authors reviewed all the AI-assisted work and take responsibility for the final content, claims, and artifacts.

ETHICS STATEMENT

This work aims to improve the efficiency of GUI agents by reducing redundant visual tokens in high-resolution screenshots. More efficient GUI-agent can lower inference cost and reduce GPU memory usage. This could make screen-grounded agents more practical on edge devices, which may support broader access to mobile-device control and cross-platform user-interface interaction. This work adheres to the ICLR Code of Ethics.

REPRODUCIBILITY STATEMENT

The full experiment setup and ablation study setup is discussed in §4.1 and §5.1, respectively. The implementation details are discussed in Appx. §A. We also provide an anonymized version of code and trajectories in the supplementary material.

REFERENCES

Anthropic. Developing a computer use model, 2024. URL `https://www.anthropic.com/news/developing-computer-use`.

Shuai Bai, Yuxuan Cai, Ruizhe Chen, Keqin Chen, Xionghui Chen, Zesen Cheng, Lianghao Deng, Wei Ding, Chang Gao, Chunjiang Ge, et al. Qwen3-vl technical report. *arXiv preprint arXiv:2511.21631*, 2025.

Zefan Cai, Yichi Zhang, Bofei Gao, Yuliang Liu, Yucheng Li, Tianyu Liu, Keming Lu, Wayne Xiong, Yue Dong, Junjie Hu, et al. Pyramidkv: Dynamic kv cache compression based on pyramidal information funneling. *arXiv preprint arXiv:2406.02069*, 2024.

Zefan Cai, Wen Xiao, Hanshi Sun, Cheng Luo, Yikai Zhang, Ke Wan, Yucheng Li, Yeyang Zhou, Li-Wen Chang, Jiuxiang Gu, et al. R-kv: Redundancy-aware kv cache compression for reasoning models. *arXiv preprint arXiv:2505.24133*, 2025.

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*, pp. 19–35. Springer, 2024.

Kyle R Chickering, Bangzheng Li, and Muhao Chen. Qlip: A dynamic quadtree vision prior enhances mllm performance without retraining. *arXiv preprint arXiv:2505.23004*, 2025.

Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in neural information processing systems*, 35: 16344–16359, 2022.

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

Mohamed Dhouib, Davide Buscaldi, Sonia Vanier, and Aymen Shabou. Pact: Pruning and clustering-based token reduction for faster visual language models. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pp. 14582–14592, 2025.

Kung-Hsiang Huang, Haoyi Qiu, Yutong Dai, Caiming Xiong, and Chien-Sheng Wu. Gui-kv: Efficient gui agents via kv cache with spatio-temporal awareness. *arXiv preprint arXiv:2510.00536*, 2025.

Gregory M. Hunter and Kenneth Steiglitz. Operations on images using quad trees. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-1(2):145–153, 1979. doi: 10.1109/TPAMI.1979.4766900.

Yutao Jiang, Qiong Wu, Wenhao Lin, Wei Yu, and Yiyi Zhou. What kind of visual tokens do we need? training-free visual token pruning for multi-modal large language models from the perspective of graph. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 4075–4083, 2025.

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*, pp. 8778–8786, 2025.

Wei Li, William Bishop, Alice Li, Chris 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, 2024a.

Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. Snapkv: Llm knows what you are looking for before generation. *Advances in Neural Information Processing Systems*, 37:22947–22970, 2024b.

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*, pp. 19498–19508, 2025a.

Zhihang Lin, Mingbao Lin, Luxi Lin, and Rongrong Ji. Boosting multimodal large language models with visual tokens withdrawal for rapid inference. In Toby Walsh, Julie Shah, and Zico Kolter (eds.), *Thirty-Ninth AAAI Conference on Artificial Intelligence, Thirty-Seventh Conference on Innovative Applications of Artificial Intelligence, Fifteenth Symposium on Educational Advances in Artificial Intelligence, AAAI 2025, Philadelphia, PA, USA, February 25 - March 4, 2025*, pp. 5334–5342. AAAI Press, 2025b. doi: 10.1609/AAAI.V39I5.32567. URL https://doi.org/10.1609/aaai.v39i5.32567.

Zichuan Lin, Feiyu Liu, Yijun Yang, Jiafei Lyu, Yiming Gao, Yicheng Liu, Zhicong Lu, Yangbin Yu, Mingyu Yang, Junyou Li, et al. Ui-voyager: A self-evolving gui agent learning via failed experience. *arXiv preprint arXiv:2603.24533*, 2026.

Jizhihui Liu, Guangdao Zhu, and Feiyi Du. Hiprune: Training-free visual token pruning via hierarchical attention in vision-language models (student abstract). In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 41275–41277, 2026.

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

Weian Mao, Xi Lin, Wei Huang, Yuxin Xie, Tianfu Fu, Bohan Zhuang, Song Han, and Yukang Chen. Triattention: Efficient long reasoning with trigonometric kv compression. *arXiv preprint arXiv:2604.04921*, 2026.

Shravan Nayak, Xiangru Jian, Kevin Qinghong Lin, Juan A Rodriguez, Montek Kalsi, Rabiul Awal, Nicolas Chapados, M Tamer Özsu, Aishwarya Agrawal, David Vazquez, et al. Ui-vision: A desktop-centric gui benchmark for visual perception and interaction. *arXiv preprint arXiv:2503.15661*, 2025.

Mingyu Ouyang, Kevin Qinghong Lin, Mike Zheng Shou, and Hwee Tou Ng. Focusui: Efficient ui grounding via position-preserving visual token selection. *arXiv preprint arXiv:2601.03928*, 2026.

Tianfan Peng, Yuntao Du, Pengzhou Ji, Shijie Dong, Kailin Jiang, Mingchuan Ma, Yijun Tian, Jinhe Bi, Qian Li, Wei Du, et al. Can visual input be compressed? a visual token compression benchmark for large multimodal models. *arXiv preprint arXiv:2511.02650*, 2025.

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.

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

Fei Tang, Zhiqiong Lu, Boxuan Zhang, Weiming Lu, Jun Xiao, Yueting Zhuang, and Yongliang Shen. Clawgui: A unified framework for training, evaluating, and deploying gui agents. *arXiv preprint arXiv:2604.11784*, 2026.

Junyang Wang, Haiyang Xu, Jiabo Ye, Ming Yan, Weizhou Shen, Ji Zhang, Fei Huang, and Jitao Sang. Mobile-agent: Autonomous multi-modal mobile device agent with visual perception. *arXiv preprint arXiv:2401.16158*, 2024a.

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*, 2024b.

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

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: A foundation action model for generalist gui agents. *arXiv preprint arXiv:2410.23218*, 2024.

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. In Danielle Belgrave, Cheng Zhang, Laura N. Montoya, Hsuan-Tien Lin, Razvan Pascanu, Piotr Koniusz, Marzyeh Ghassemi, Nancy Chen, Iván Vladimir Meza Ruíz, and Arturo Loaiza-Bonilla (eds.), *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2025, NeurIPS 2025, San Diego, CA, USA, December 2-7, 2025 / Mexico City, Mexico, November 30 - December 5, 2025*, 2025. URL http://papers.nips.cc/paper_files/paper/2025/hash/22c868099177ee278eb7baccec649f35-Abstract-Datasets_and_Benchmarks_Track.html.

Haiyang Xu, Xi Zhang, Haowei Liu, Junyang Wang, Zhaozai Zhu, Shengjie Zhou, Xuhao Hu, Feiyu Gao, Junjie Cao, Zihua Wang, et al. Mobile-agent-v3. 5: Multi-platform fundamental gui agents. *arXiv preprint arXiv:2602.16855*, 2026.

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.

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 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 19792–19802, 2025.

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

Xiaofeng Zhang, Fanshuo Zeng, Yihao Quan, Zheng Hui, and Jiawei Yao. Enhancing multimodal large language models complex reason via similarity computation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 10203–10211, 2025.

Hanzhang Zhou, Xu Zhang, Panrong Tong, Jianan Zhang, Liangyu Chen, Quyu Kong, Chenglin Cai, Chen Liu, Yue Wang, Jingren Zhou, et al. Mai-ui technical report: Real-world centric foundation gui agents. *arXiv preprint arXiv:2512.22047*, 2025.

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.

A Detailed Algorithm

A.1 Grid Alignment and Boundary Handling

Given a resized screenshot of size $W \times H$, AQuaUI constructs the adaptive partition on the model's merged-token grid. Let p denote the ViT patch size and m denote the spatial merge size. Each LLM-side visual token corresponds to a block of size $b = p \cdot m$. For Qwen2-VL, $p = 14$ and $m = 2$, so $b = 28$ pixels. For Qwen3-VL, $p = 16$ and $m = 2$, so $b = 32$ pixels.

Since dimensions of GUI screenshots are not necessarily powers of two, directly applying a single quadtree to the whole image won't work. AQuaUI therefore first decomposes the image into two parts: a centered region that can be tiled by square chunks of size $C \times C$, and boundary margins that cannot be included in this chunk layout, which are kept as $b \times b$ token-sized leaves.

On the centered region, AQuaUI tiles square chunks and then runs an adaptive quadtree independently inside each chunk. This design keeps all split boundaries aligned to the merged-token grid and avoids malformed edge regions.

To be more specific, the chunk side length is computed from the maximum depth d supported by the image dimensions:

$$d_h = \lfloor \log_2(H/b) \rfloor, \quad d_w = \lfloor \log_2(W/b) \rfloor$$
$$d = \max(1, \min(d_h, d_w)), \quad C = b \times 2^{(d-1)}$$

A.2 Splitting Criteria

For a quadtree node n, we will describe the splitting criteria as follows. The default score function s is the area weight greyscale variance:

$$s(n) = w_n h_n \cdot \mathrm{Var}(\mathrm{gray}(n)).$$

A region is recursively split when $s(n) > 1000 \cdot \alpha$. Area weighting makes the threshold meaningful across different region sizes. Large non-uniform regions are split even when their per-pixel variance is moderate, while large uniform regions can collapse to one token. Increasing alpha therefore increases compression.

The implementation also supports a gradient-based criterion, where the score function s gives the max magnitude of gradient of node n, which can be useful when edges provide a stronger signal.

A.3 Representative Token Selection

For each final leaf, AQuaUI selects the center merged-token block. If the leaf spans block coordinates from x_0 to x_1 and y_0 to y_1, the representative block is$x' = \lfloor \frac{x_0+x_1}{2} \rfloor, \quad y' = \lfloor \frac{y_0+y_1}{2} \rfloor$. The representative merged-token coordinate is therefore (x', y')

A.4 Conditional Quadtree Building

GUI trajectories often contain adjacent screenshots that are nearly identical. A page may remain static, a menu may open, or a scroll operation may shift content by a few token blocks. If each screenshot is compressed independently, a later frame may merge away details that were preserved in an earlier frame, weakening temporal reasoning and action grounding.

AQuaUI introduces a conditional quadtree mechanism that infers history from the ordered images in the same request. We design a heuristic algorithm for this.

A.4.1 Static, Shifted, and Replaced Modes

Static mode We begin by identify patterns that most pixels remain the same, which we called the Static mode. For this purpose, we define a similarity value between two chunks. Consider the chunks we produced in §3.2, we compress it into a condensed grid signature S of size (H_c, W_c) by

computing the greyscale of each block of size (b, b) so that each cell of S stores the rounded mean grayscale of the corresponding $b \times b$ block with $S \in [0, 255]^{H_c \times W_C}$.

For the similarity of two adjacent signatures S_t and S_{t-1}, we define similarity as following:

$$\text{similarity}(S_t, S_{t-1}) = 1 - \frac{1}{255 H_c \cdot W_c} \times \sum_{i,j} |S_t(i,j) - S_{t-1}(i,j)|$$

A region is static if similarity $\geq \tau_{static}$.

Shifted mode If two chunks are not static, we then consider if a scroll operation happens. We call this the Shifted mode. To capture the scrolling feature, we add a block-level shift δ in previous definition. More specifically, given current and previous chunk signatures $S_t, S_{t-1} \in [0, 255]^{H_c \times W_c}$, we consider a candidate block-level shift $\delta = (\Delta x, \Delta y)$. We continue to calculate the similarity on the overlapping parts of the two signatures $\Omega(\delta)$. We define $\Omega(\delta)$ as the following:

$$\Omega(\delta) = \{(i,j) \mid 0 \leq i < H_c,\ 0 \leq j < W_c,\ 0 \leq i - \Delta x < H_c,\ 0 \leq j - \Delta y < W_c\}.$$

The overlap ratio is$\rho(\delta) = \frac{|\Omega(\delta)|}{H_c W_c}$. We only evaluate shifts satisfying $\rho(\delta) \geq \rho_{\min}$. Therefore, similarity becomes

$$\text{similarity}(\delta) = 1 - \frac{1}{255|\Omega(\delta)|} \sum_{(i,j) \in \Omega(\delta)} |S_t(i,j) - S_{t-1}(i - \Delta x, j - \Delta y)|\,.$$

We search over $|\Delta x|, |\Delta y| \leq d_{\max}$ and classify the region as shifted if the best valid shift $\delta^\star$ satisfies

$$\text{sim}(\delta^\star) \geq \tau_{\text{shift}} \quad \text{and} \quad \text{sim}(\delta^\star) \geq \text{sim}(S_t, S_{t-1}) + \gamma.$$

In shifted mode, only coarse previous leaves are translated and reused:

$$\max(w, h) \geq 2 \times b$$

Fine leaves are regenerated because small alignment errors around text and icons can be more harmful than independent recompression.

Replaced mode Regions satisfying neither condition are treated as replaced. A new quadtree is constructed where no priors are used.

A.4.2 Sensitivity to the Thresholds

A previous-frame prior can only split leaves of the independently built quadtree: a leaf is replaced only by previous leaves that exactly tile it (Alg. 1). A wrong mode decision therefore either adds tokens or leaves the independent quadtree unchanged; it never merges regions that the independent quadtree keeps apart. Single-image inputs, which include all grounding and AndroidControl results, have no history and do not depend on these thresholds.

To measure how much the thresholds matter, we perturb each of them in isolation ($\tau_{static} \in \{0.95, 0.99\}$, $\tau_{shift} \in \{0.92, 0.96\}$, $\gamma \in \{0.01, 0.05\}$, $d_{\max} \in \{2, 8\}$, $\rho_{min} \in \{0.3, 0.7\}$) and jointly (permissive: $0.95, 0.92, 0.01, 8, 0.3$; conservative: $0.99, 0.96, 0.05, 2, 0.7$). We apply every setting to 621 pairs of adjacent screenshots from 128 AndroidControl episodes, selected by hashing episode IDs, and rebuild the quadtree of the later screenshot with the earlier one as history, using the implementation and Qwen3-VL preprocessing of our navigation experiments with $\alpha = 64$. Tab. 6 reports how many pairs receive a different visual input than under the default setting and how many tokens the prior adds.

Three observations follow. First, the prior only refines: across all settings and pairs, no screenshot receives fewer tokens than its independent quadtree. Second, it is cheap: under the default setting it adds 16.9 tokens per screenshot on average, 0.95% of the 1,778 tokens of the independent quadtree, and no setting adds more than 39.4 tokens (2.2%). Third, only τ_{static} materially changes the input (131–150 of 621 pairs). The four shift-related thresholds change it for at most 12 pairs, because only 0.09% of chunk decisions select the shifted mode on AndroidControl. Since identical inputs yield identical greedy predictions, these thresholds can alter the outcome of at most 1.9% of the pairs.

Table 6: Sensitivity of conditional quadtree building to its thresholds on 621 adjacent AndroidControl screenshot pairs. Changed inputs: pairs whose token selection for the later screenshot differs from the default setting. Extra tokens: mean number of tokens that the history prior adds over the independent quadtree of the later screenshot (1,778 tokens on average). Shifted: chunk decisions in shifted mode, out of 319,356.

Setting	Changed inputs	Extra tokens	Shifted
Default	–	16.9	281
$\tau_{static} = 0.95$	150	39.3	266
$\tau_{static} = 0.99$	131	5.0	281
$\tau_{shift} = 0.92$	2	16.9	414
$\tau_{shift} = 0.96$	1	16.9	150
$\gamma = 0.01$	4	16.9	595
$\gamma = 0.05$	3	16.9	156
$d_{\max} = 2$	6	16.9	72
$d_{\max} = 8$	12	16.9	639
$\rho_{min} = 0.3$	0	16.9	281
$\rho_{min} = 0.7$	3	16.9	229
Joint, permissive	163	39.4	1,332
Joint, conservative	134	5.0	9
No history	172	0.0	0

A.5 Implementation Details

We implement AQuaUI on both Qwen2-VL and Qwen3-VL. We intentionally choose to implement on vLLM to prove its practicality because its standard interface does not easily support the modification. A vanilla Transformers implementation would make this integration more direct.

Although the high-level quadtree construction is shared across models, the two model families require small integration tweaks because their visual tokenization and position-encoding pipelines differ. We describe the model-specific details below.

A.5.1 Qwen2-VL Details

Rather than reimplementing Qwen2-VL preprocessing, AQuaUI first runs the vanilla fast processor and then indexes into the resulting dense patch tensor. This preserves Qwen's original resizing, normalization, and patch extraction behavior.

Let m denote the spatial merge size and let w denote the width of the dense merged-token grid. For a representative merged-token coordinate $(x_{\text{rep}}, y_{\text{rep}})$, the corresponding offset in the dense patch tensor is

$$k = (y_{\text{rep}} \cdot w + x_{\text{rep}}) \cdot m^2.$$

The selected patch rows are then

$$\mathcal{I}_{\text{rep}} = \{k, k+1, \ldots, k+m^2-1\}.$$

For the standard Qwen2-VL setting, $m = 2$, so each retained merged token corresponds to four ViT patch rows. This indexing keeps the selected token compatible with Qwen2-VL's original merge-block ordering.

A.5.2 Qwen3-VL Details

For Qwen3-VL, after the vanilla fast processor produces the dense patch tensor, we select the patch rows corresponding to the representative quadtree tokens and construct a reduced visual input.

The reduced visual tokens still need to be represented by a valid rectangular grid for vLLM's multimodal batching and placeholder expansion. We therefore use aspect-aware padded grid packing: the selected tokens are packed into a near-aspect rectangular grid, and when padding is necessary, the final selected token is duplicated. These duplicate tokens are used only to satisfy shape constraints; their true positions are still tracked through the quadtree metadata.

Algorithm 1 Conditional quadtree building for screenshot s_t given s_{t-1}

Require: Current and previous chunks $\{C_t^k\}, \{C_{t-1}^k\}$; thresholds $\tau_{\text{static}}, \tau_{\text{shift}}, \gamma, \rho_{\min}, d_{\max}$
Ensure: Refined partition $\mathcal{L}_t$ for s_t

for each chunk region k **do**
  $\mathcal{L}_t^k \leftarrow$ independent quadtree of C_t^k
  Compute signatures S_t^k, S_{t-1}^k
  if $\text{sim}(\mathbf{0}) \geq \tau_{\text{static}}$ **then**
    $\mathcal{P}^k \leftarrow$ leaves of C_{t-1}^k ▷ static
  else
    $\delta^\star \leftarrow \arg\max_{\delta:\, \rho(\delta) \geq \rho_{\min}} \text{sim}(\delta)$
    if $\text{sim}(\delta^\star) \geq \tau_{\text{shift}}$ **and** $\text{sim}(\delta^\star) \geq \text{sim}(\mathbf{0}) + \gamma$ **then**
      $\mathcal{P}^k \leftarrow$ coarse leaves of C_{t-1}^k translated by $\delta^\star$ ▷ shifted
    else
      $\mathcal{P}^k \leftarrow \emptyset$ ▷ replaced
    end if
  end if
  for each leaf $\ell \in \mathcal{L}_t^k$ **do**
    $\mathcal{P}^k(\ell) \leftarrow \{\ell' \in \mathcal{P}^k \mid \ell' \subseteq \ell\}$
    if $\mathcal{P}^k(\ell) \neq \emptyset$ **and** $\sum_{\ell' \in \mathcal{P}^k(\ell)} \text{area}(\ell') = \text{area}(\ell)$ **then**
      Replace ℓ with $\mathcal{P}^k(\ell)$ in $\mathcal{L}_t^k$
    end if
  end for
end for
return $\mathcal{L}_t \leftarrow \bigcup_k \mathcal{L}_t^k$

We patch Qwen3-VL's ViT-side rotary position construction to use true quadtree positions rather than compact packed-grid positions. These coordinates follow Qwen's original merge-block-interleaved ordering.

Qwen3-VL also includes an absolute position interpolation path. This path must use the original dense grid extent rather than the maximum retained coordinate.

Finally, we patch the LLM-side M-RoPE construction. For image tokens, the height and width rotary positions are replaced with the true quadtree coordinates. After injection, the M-RoPE delta is recomputed so that subsequent text tokens remain correctly aligned with the multimodal position sequence. This step is essential: without it, the model would receive fewer visual tokens but would interpret them as occupying compacted positions, which corrupts the geometry needed for GUI grounding.

A.6 Post-ViT and Pre-ViT Design Choice

In our default Qwen2-VL and Qwen3-VL implementation, reduction is applied before the vision encoder. The selected patch rows are passed directly into the vision encoder. This path can reduce both ViT-side and LLM-side computation, but it is more aggressive because the vision encoder receives a sparse visual sequence. We use pre-ViT reduction as the default Qwen2-VL setting as a speed-oriented choice.

We also implement a post-ViT ablation. The image is first processed by the dense ViT, and AQuaUI then selects the visual embeddings corresponding to the representative merged-token coordinates. This conservative path does not reduce ViT computation, but it preserves dense vision encoding quality while reducing the visual sequence passed to the LLM. The reduced visual embeddings are accompanied by their original merged-token coordinates, which are injected into the LLM-side M-RoPE position construction. This avoids treating the sparse selected tokens as a compact dense image grid. On Qwen2-VL, the post-ViT implementation improves grounding accuracy by 1.92 points on average across the five benchmarks, but increases average latency by 0.05s. For Qwen3-VL, post-ViT implementation and pre-ViT implementation has comparable performance.

Qwen3-VL is the more natural target for AQuaUI's pre-ViT reduction, because both vision-side and LLM-side position encodings can be patched to remain consistent with the sparse quadtree layout.

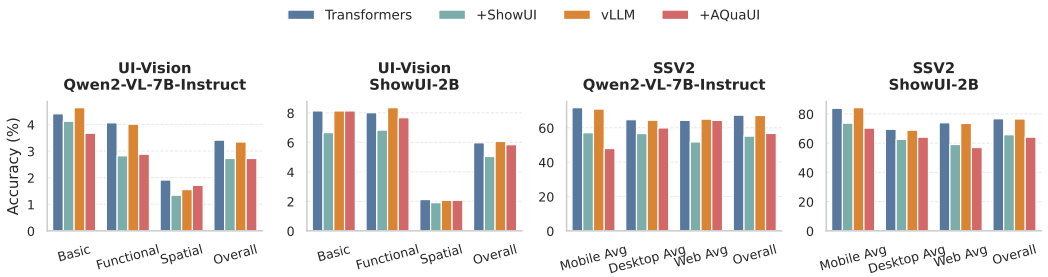

Figure 3: Performance breakdown on UI-Vision and ScreenSpot-V2. More detailed results can be seen in Appx. §B.

Table 7: ScreenSpot-Pro breakdown. Accuracy is reported in percentages.

Backend	Text	Icon	Overall
Qwen2-VL-7B-Instruct			
Transformers	15.05	2.65	10.31
+ SHOWUI	12.28	2.65	8.60
vLLM	15.35	2.48	10.44
+ AQuaUI	11.16	1.16	7.34
ShowUI-2B			
Transformers	10.24	3.15	7.53
+ SHOWUI	8.09	2.48	5.95
vLLM	10.85	2.32	7.59
+ AQuaUI	9.31	2.15	6.58

B BREAKDOWN STATISTICS ON GROUNDING BENCHMARKS

Tab. 7, Tab. 8, Tab. 9, Tab. 10, Tab. 11 display the performance breakdown of different benchmarks.

Platform breakdown. On ScreenSpot-V2 (Tab. 8), AQuaUI degrades less than SHOWUI on Desktop for both backbones but more on Mobile. We analyze the failure modes in Appx. §B.1.

B.1 FAILURE-CASE ANALYSIS

We identify two primary failure modes. The first one is **local-detail failure**. Very small or low-contrast elements may not produce sufficient local variance to trigger further quadtree splitting (Fig. 4a–b). The grayscale variance criterion may also underrepresent color differences with similar luminance. Elements that are particularly vulnerable include: (a) tiny disclosure arrows or status indicators; (b) thin separators and one-pixel borders; (c) low-contrast icons on dark or uniform backgrounds; and (d) chromatically different but similarly luminous regions.

The second failure pattern is **context-loss** failure. A target may itself be retained while its surrounding structure is compressed too aggressively. GUI understanding often requires relational context—such as sibling controls, container boundaries, labels, or group membership—in addition to the target pixels.

For instance, when the task is “volume control among the playback controls” in HuLu (Fig. 4c), the volume icon and the neighboring playback buttons are retained, but most of the dark player surface around them is represented by coarse leaves, and AQuaUI clicks inside the video, 510 px above the target. Another example is the task "expand the file structure" in EViews (Fig. 4d). The 8×8 px disclosure triangle and the whole options tree are retained, while the surrounding near-black workspace is heavily compressed; AQuaUI nevertheless selects another tree item 206 px below the target.

These cases explain why some subcategories can degrade even when the target region is not directly removed. Pixel preservation is not always equivalent to semantic preservation; preserving the target’s local grouping and relational context is also important.

Table 8: ScreenSpot-V2 breakdown. Accuracy is reported in percentages.

Backend	Mobile			Desktop			Web			Avg.
	Text	Icon	Avg.	Text	Icon	Avg.	Text	Icon	Avg.	
Qwen2-VL-7B-Instruct										
Transformers	75.86	65.88	71.66	74.23	51.43	64.67	69.23	58.62	64.30	67.30
+ SHOWUI	55.17	59.72	57.09	64.43	45.71	56.59	56.41	46.31	51.72	55.11
vLLM	74.14	66.35	70.86	73.71	51.43	64.37	69.66	59.61	64.99	67.14
+ AQuaUI	51.38	43.13	47.90	67.53	49.29	59.88	71.37	56.16	64.30	56.68
ShowUI-2B										
Transformers	91.72	72.99	83.83	77.32	58.57	69.46	83.76	62.56	73.91	76.65
+ SHOWUI	80.00	64.93	73.65	67.53	55.71	62.57	67.09	49.75	59.04	65.72
vLLM	91.38	74.41	84.23	76.80	57.86	68.86	82.48	63.05	73.46	76.49
+ AQuaUI	75.52	63.03	70.26	71.65	53.57	64.07	61.11	52.22	56.98	64.07

Table 9: OSWorld-g breakdown. Accuracy is reported in percentages. Refusal samples are excluded from Overall accuracy. TextMatch: Text Matching; ElemRecog: Element Recognition; LayoutUnd: Layout Understanding; FineManip: Fine-grained Manipulation.

Backend	TextMatch	ElemRecog	LayoutUnd	FineManip	Overall
Qwen2-VL-7B-Instruct					
Transformers	31.67	25.16	22.59	16.67	21.14
+ SHOWUI	25.83	19.61	17.99	12.12	16.87
vLLM	28.75	22.55	21.34	16.67	19.36
+ AQuaUI	29.58	20.59	18.83	15.15	18.12
ShowUI-2B					
Transformers	31.67	23.53	22.18	14.39	18.29
+ SHOWUI	26.25	22.22	19.25	11.36	16.34
vLLM	31.67	23.86	21.76	12.88	18.12
+ AQuaUI	27.50	21.24	19.67	12.12	16.34

C COMPUTATIONAL AND LATENCY ADVANTAGE OF AQUAUI

In §5.1, AQuaUI is compared with FastV on matched FLOPs. Tab. 12 and Tab. 13 justifies the choice by establishing both computational and latency advantage of AQuaUI when token budget is matched. Since the vision encoder and first $K-1$ layers are irreducible in FastV, AQuaUI is noticeably better at computation and latency with matched token budget.

D LIMITATIONS

Model- and serving-stack-specific implementation. AQuaUI is training-free, but it is not entirely implementation-free. Correctly preserving visual-token positions requires modifying model-specific position-encoding paths and transporting quadtree metadata through the serving stack. Our current implementation merely focuses on Qwen2-VL and Qwen3-VL, but it can be migrated to other architectures by following two main integration steps: (1) preserving or reconstructing the spatial information of retained tokens according to that model's positional-encoding scheme; and (2) passing the quadtree-derived metadata through the inference framework.

Limited gains when reduction overhead dominates. Although AQuaUI reduces the number of visual tokens, it introduces preprocessing overhead from quadtree construction, metadata transport, and position injection. As a result, latency gains are not guaranteed for small models or settings where the vision encoder and LLM-side computation are already inexpensive. Our results show that the speedup becomes more visible on larger backbones, where the saved visual-token computation

Table 10: UI-Vision breakdown. Accuracy is reported in percentages.

Backend	Basic	Functional	Spatial	Overall
Qwen2-VL-7B-Instruct				
Transformers	4.40	4.06	1.91	3.41
+ SHOWUI	4.12	2.82	1.34	2.72
vLLM	4.63	4.01	1.55	3.34
+ AQuaUI	3.67	2.88	1.71	2.72
ShowUI-2B				
Transformers	8.13	8.01	2.12	5.97
+ SHOWUI	6.66	6.83	1.91	5.04
vLLM	8.13	8.35	2.07	6.06
+ AQuaUI	8.13	7.67	2.07	5.84

Table 11: MMBench-GUI (L2 Grounding) breakdown. Accuracy is reported in percentages.

Backend	Windows	MacOS	Linux	iOS	Android	Web	Overall
Qwen2-VL-7B-Instruct							
Transformers	25.41	26.63	21.71	59.01	52.18	42.88	39.57
+ SHOWUI	19.52	14.91	18.09	41.15	39.24	33.66	28.69
vLLM	26.70	27.79	20.93	58.54	51.90	43.69	39.90
+ AQuaUI	20.07	23.44	20.67	35.40	30.10	42.23	29.33
ShowUI-2B							
Transformers	28.36	30.54	19.12	63.82	56.96	49.84	43.49
+ SHOWUI	21.92	18.38	17.05	50.62	45.85	39.81	33.67
vLLM	29.28	31.26	19.38	64.29	56.68	51.13	44.05
+ AQuaUI	24.86	27.35	10.08	51.55	47.26	42.56	36.00

Table 12: AQuaUI has a latency advantage over FastV at matched token budget with Qwen3-VL-8B. GPU prefill time is measured per image on Nvidia RTX Ada 6000.

Benchmark	Dense	FastV	AQuaUI
ScreenSpot-V2	598.3 ms	544.7 ms	410.3 ms
ScreenSpot-Pro	1718.6 ms	1637.1 ms	1039.4 ms

can outweigh the extra reduction overhead. More importantly, there are operations in our algorithm that can be vectorized for speedup, but we don't implement these improvements for now.

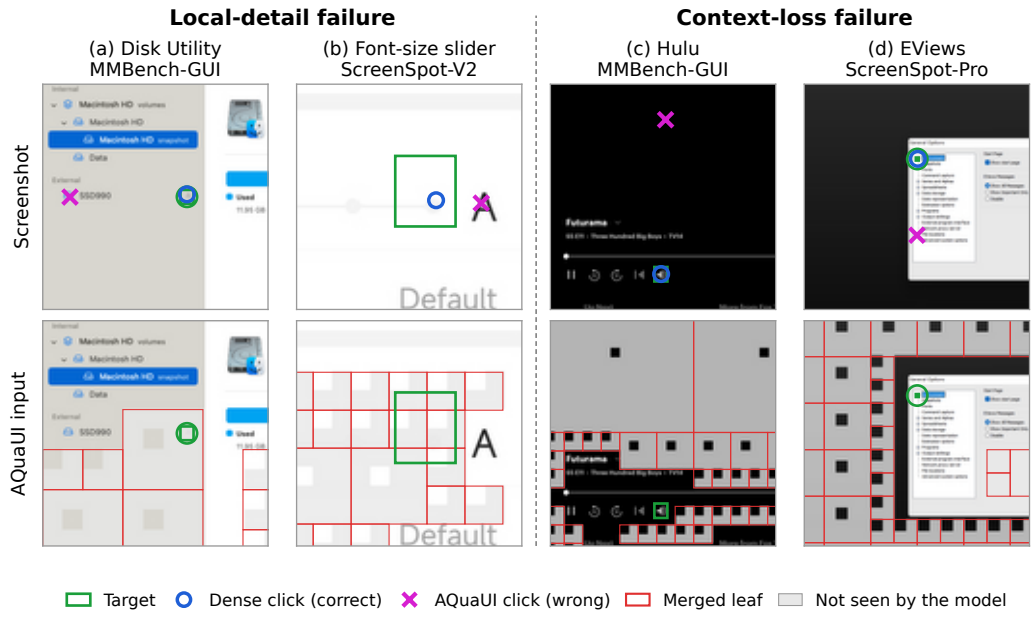

Figure 4: Failure cases of AQuaUI on Qwen3-VL-8B-Instruct ($\alpha = 64$, vLLM). Each column is a sample that the dense model answers correctly but AQuaUI does not. Top: screenshot crop with the target (green; tiny targets are also circled), the dense click (blue circle), and the AQuaUI click (magenta cross). Bottom: the pixels AQuaUI passes to the model; each merged quadtree leaf (red) is represented only by its central block, and the remaining pixels (gray) are not seen. (a) "A small square button with a light outline located to the right of the SSD990 external drive": the 21×22 px low-contrast button lies inside a 4×4-block merged leaf, and AQuaUI clicks the drive label instead. (b) "Set to use the largest font size": the faint slider track near its right end is merged, and AQuaUI clicks the adjacent large "A" label. (c) Hulu, "A volume/sound control button displayed among the media playback controls": the icon and the neighboring buttons are retained, but most of the dark player surface is merged; AQuaUI clicks inside the video, 510 px above the target. (d) EViews, "expand the file structure": the 8×8 px expander and the options tree are retained, while the near-black workspace is merged; AQuaUI clicks another tree item 206 px below the target. Cases like (a) and (b) are rare: among the 343 samples that the dense model answers correctly but AQuaUI does not, the target center lies inside a merged leaf in only 8 (2.3%).

Table 13: AQuaUI has a computational advantage over FastV at matched token budget with Qwen3-VL-8B.

Benchmark	Dense	FastV	AQuaUI
ScreenSpot-V2	66.6 TFLOPs	55.2 TFLOPs	44.1 TFLOPs
ScreenSpot-Pro	183.4 TFLOPs	155.0 TFLOPs	114.4 TFLOPs