REVISION: Scaling Computer-Use Agents via Temporal Visual Redundancy Reduction

Amirhossein Abaskohi[1*], Yuhang He[2], Peter West[1], Giuseppe Carenini[1], Pranit Chawla[2], Vibhav Vineet[2]

[1]University of British Columbia, [2]Microsoft Research

Abstract

Computer-use agents (CUAs) rely on visual observations of graphical user interfaces, where each screenshot is encoded into a large number of visual tokens. As interaction trajectories grow, the token cost increases rapidly, limiting the amount of history that can be incorporated under fixed context and compute budgets. This has resulted in no or very limited improvement in the performance when using history unlike other domains. We address this inefficiency by introducing **REVISION**, which is used to train multimodal language models on trajectories where redundant visual patches are removed using a learned patch selector that compares patch representations across consecutive screenshots while preserving spatial structure required by the model. Across three benchmarks, OSWorld, WebTailBench, and AgentNetBench, when processing trajectories with 5 history screenshots using QWEN2.5-VL-7B, **REVISION reduces token usage by 46% on average while improving success rate by 3% over the no drop baseline**. This establishes a clear efficiency gain, enabling agents to process longer trajectories with fewer tokens. With this improved efficiency, we revisit the role of history in CUAs and find that performance continues to improve as more past observations are incorporated when redundancy is removed[1].

1 Introduction

Multimodal large language models (MLLMs) have enabled agents that interact with graphical user interfaces (GUIs) by combining visual understanding with language-based reasoning (Wang et al., 2025b,a). These computer-use agents (CUAs) operate on screenshots to generate grounded actions such as clicks, typing, and navigation for multi-step tasks. Benchmarks such as VisualWebArena (Koh et al., 2024a), OSWorld (Xie et al., 2024), and AgentNetBench (Wang et al., 2025b) demonstrate their ability to handle complex workflows across web and desktop environments. Most systems rely primarily on the current screenshot, sometimes with limited history (Sager et al., 2026), despite many tasks requiring memory of past states or actions. Scaling such long-horizon reasoning remains challenging due to the need to process extended visual trajectories under constrained context budgets (Chen et al., 2026).

A straightforward way to provide memory is to append past screenshots to the model context. However, this is highly inefficient: each additional image adds hundreds or thousands of visual tokens, quickly exhausting the context budget. As shown in Figure 1, much of this cost is redundant because consecutive GUI screenshots largely overlap. The model therefore repeatedly processes unchanged visual content, wasting computation and limiting longer-history reasoning. Reducing this redundancy is not just an efficiency optimization, but a key enabler of better decision-making: it frees context budget for longer, more informative histories that support long-horizon interactions.

To address this inefficiency, we introduce **REVISION**, a redundancy-aware training framework for MLLMs that operates on trajectories where redundant visual patches are removed across consecutive screenshots. At the core of REVISION is a learned patch selector that compares patch-level representations over time and filters out visually redundant regions while preserving the spatial structure required by the model. Rather than applying token reduction only at inference time, we train the model on filtered trajectories, allowing it to reason over compact visual histories. This enables REVISION to remove redundant visual tokens without architecture changes while remaining compatible with existing MLLMs.

This efficiency improvement directly translates

*Corresponding author: aabaskoh@cs.ubc.ca. This work was done during the internship at Microsoft Research (MSR), Vancouver Lab.

[1]Code is available on GitHub.

Findings of the Association for Computational Linguistics: EMNLP 2026, pages 46621–46646
October 24-29, 2026 ©2026 Association for Computational Linguistics

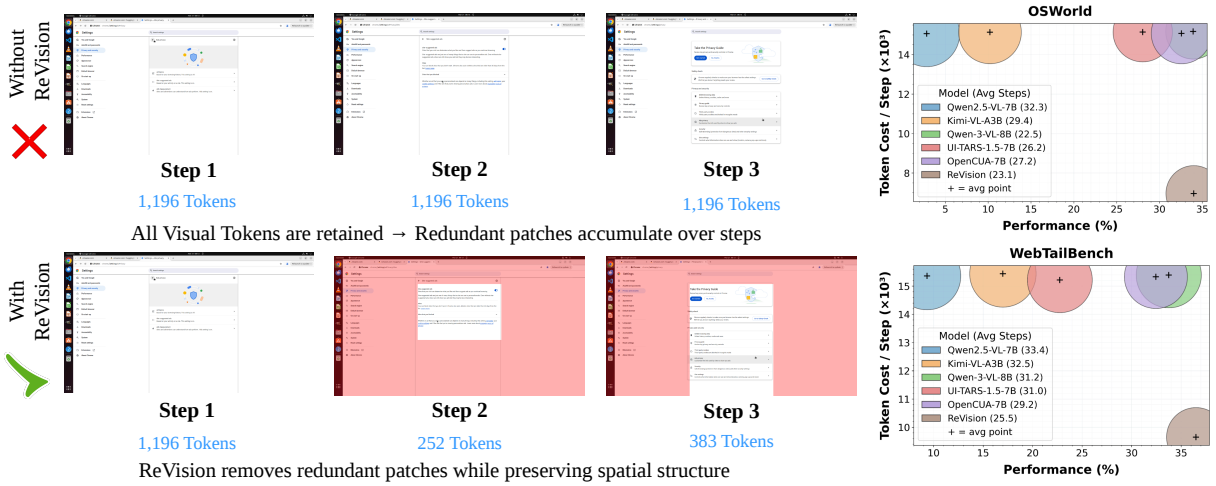


Figure 1: **Token efficiency with REVISION. Left:** REVISION removes redundant patches across steps, reducing token accumulation while preserving spatial structure. **Right:** REVISION achieves higher success rates at maximum 100 steps OSWorld and WebTailBench, with lower token cost across models. Circle size indicates average steps to complete tasks.

to better performance. Across OSWorld (Xie et al., 2024), WebTailBench (Awadallah et al., 2025), and AgentNetBench (Wang et al., 2025b), when using 5 history screenshots with QWEN2.5-VL-7B (Bai et al., 2025b), **REVISION reduces token usage by approximately 46% on average while achieving a +3% gain in success rate over the no-drop baseline**. With only 3 history images, REVISION achieves performance close to the some of the best baseline while using roughly half of the visual tokens. As the history length increases, the gains become more pronounced: with 5 or more images, REVISION consistently outperforms most of the baselines with the same size by at least 2% on average (Figure 1, right). **REVISION shifts the efficiency frontier by enabling longer visual histories under similar compute budgets while achieving higher success rates**.

Our contributions are as follows: **(i)** we identify and quantify substantial temporal redundancy in sequential screenshots from long computer-use trajectories, showing that a large fraction of visual tokens remains unchanged across consecutive steps; **(ii)** we introduce **REVISION**, a QWEN2.5-VL-7B based model trained with a temporal patch scorer that performs patch-level filtering across consecutive screenshots, enabling the model to reason over compact visual histories without modifying the underlying architecture; **(iii)** we demonstrate across long-horizon computer-use benchmarks that redundancy-aware history filtering reduces token usage, improves success rates, and **delays the saturation point of visual history**.

2 Related Work

Computer-use agents and benchmarks. Recent progress in CUAs has been driven by multimodal models that interact with digital environments through screenshots and natural language. Early systems such as WebShop and WebArena rely on structured representations like DOM or accessibility trees (Yao et al., 2022; Zhou et al., 2023). In contrast, a growing line of work adopts a vision-first paradigm, reasoning directly over pixels. Methods such as CogAgent, AGUVIS, OpenCUA, FARA, WebSTAR, and UI-TARS operate purely on visual inputs (Hong et al., 2023; Xu et al., 2024; Wang et al., 2025b; Awadallah et al., 2025; He et al., 2026; Qin et al., 2025; Wang et al., 2025a). Other approaches, including WebVoyager, SeeAct, and ScaleCUA, incorporate both visual observations and structured signals to improve robustness in complex environments (He et al., 2024; Zheng et al., 2024; Liu et al., 2025). Benchmarks including WebArena, VisualWebArena, OSWorld, and AgentNetBench enable evaluation in long-horizon settings (Zhou et al., 2023; Koh et al., 2024a; Xie et al., 2024; Wang et al., 2025b). Despite this progress, agents typically rely on limited visual history, and increasing history length yields diminishing returns, highlighting inefficiencies in naive context scaling (Abhyankar et al., 2025; Kerboua et al., 2025). Our setting instead requires filtering visual history at patch granularity across consec-

Dataset/Benchmark	Avg. Steps / Task	Avg. # of Patches / Image	Avg. Redundant Patches / Image	Avg. (%) Redundant Patches / Image
AgentNetBench 2025b	12.1	2,284	1,014	44.4%
OSWorld 2024	16.9	2,769	1,556	56.2%
WindowsAgentArena 2024	11.7	2,769	1,462	52.8%
WebTailBench 2025	22.4	2,769	1,174	42.4%
Mind2Web2 2025	13.4	2,769	1,199	43.3%
VisualWebArena 2024b	6.8	2,769	1,373	49.6%
AndroidWorld 2024	7.6	1,196	456	38.2%
GUIAct 2025	5.5	1,196	435.3	36.4%
Average	12.1	2,315	1,083	45.4%

Table 1: **Dataset-level visual redundancy in computer-use benchmarks.** We report average steps, patches per image, and redundant patches across environments. While *GUIAct* and *AgentNetBench* are offline benchmarks with fixed steps, others depend on agent performance. We use GPT-5.4 to ensure minimal and consistent trajectories for fair comparison. Results show that 36%–56% of visual tokens are redundant across steps, motivating **REVISION**.

utive screenshots while preserving temporal evidence for long-horizon decisions.

Visual token pruning and context compression. Prior work reduces visual token usage either within images or across trajectory steps. Methods such as ShowUI and FocusUI prune spatially redundant regions within a single screenshot (Lin et al., 2024; Ouyang et al., 2026), while approaches like Focus-Scan-Refine and adaptive compression further remove tokens based on saliency or importance (Tong et al., 2026; Huang et al., 2026a,b). At the trajectory level, methods such as FocusAgent reduce the number of past steps included in context (Kerboua et al., 2025). However, these approaches operate either spatially within images or temporally at the step level, without explicitly modeling redundancy across consecutive screenshots, leading to repeated processing of unchanged visual regions.

Temporal redundancy in sequential visual data. Temporal redundancy has been extensively studied in video understanding, where consecutive frames share similar content. Prior work addresses this via keyframe selection, feature reuse, and token compression (Choudhury et al., 2024; Korbar et al., 2019; Choi et al., 2024; Tao et al., 2025; Yao et al., 2025). However, CUAs differ: screenshots evolve through agent actions and must be processed jointly with textual reasoning. Existing approaches operate at the frame or feature level within vision models, whereas our setting requires patch-level filtering in the token space of multimodal LLMs while preserving temporally distributed evidence for long-horizon decision making.

3 Temporal Visual Redundancy

CUAs operate over sequences of screenshots that capture the evolving state of a digital environment. At each step, the model encodes the current screenshot into a large number of visual tokens and processes them together with accumulated textual context to predict the next action. However, consecutive screenshots in a trajectory often exhibit substantial visual overlap: in most steps, only a small region of the interface changes (e.g., a button click or text update), while the majority of the screen remains unchanged. Despite this, standard multimodal models process each image independently, resulting in repeated encoding and consumption of nearly identical visual tokens across time.

To quantify this, we analyze pairs of consecutive screenshots (I_{t-1}, I_t) across multiple benchmarks and measure redundancy by comparing corresponding patches. As shown in Table 1, redundancy is consistently high, with an average of **45.4%** of patches unchanged across steps and over **56%** in some settings. This corresponds to over **1,000 redundant patches per step** on average. These findings show that a large portion of computation is spent on repeated visual content and that the context budget is dominated by redundant tokens, limiting the model's ability to incorporate useful history. This motivates REVISION, which removes redundant visual tokens across time while preserving task-relevant information.

4 REVISION

As illustrated in Figure 2, REVISION reduces redundant visual tokens across sequential GUI observations by learning to selectively retain only

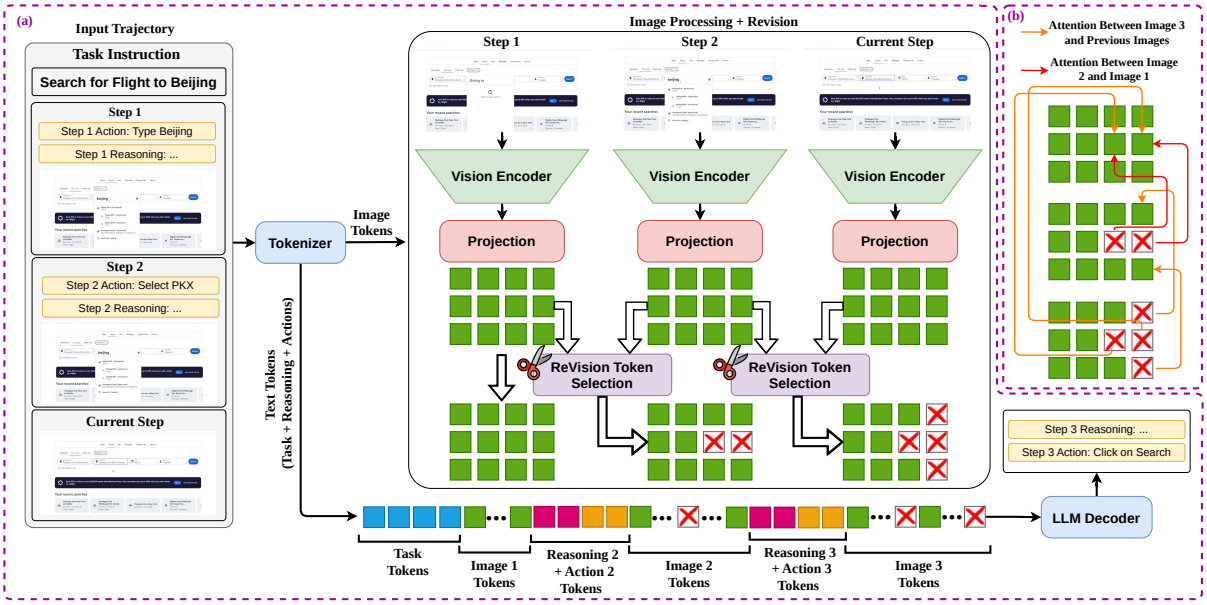


Figure 2: **Overview of ReVision. (a)** ReVision removes redundant patches by comparing corresponding tokens across consecutive screenshots, reducing visual tokens while preserving spatial alignment before passing them to the LLM. **(b)** The model learns to attend to relevant regions in previous images, enabling effective reasoning with reduced visual input.

informative patches. Our approach consists of two main components. **First**, we train a lightweight **three-layer MLP classifier**, referred to as *ReVision Token Selection* (RTS), which takes as input the embeddings of two corresponding patches from consecutive screenshots and predicts whether the patch in the current image is redundant given the previous one. **Second**, we integrate RTS into the pipeline of a MLLM and fine-tune the model on AgentNet (Wang et al., 2025b) trajectories (with a fixed history image window) where redundant patches are removed from all but the first image. This training setup encourages the model to recover omitted visual information from earlier observations, enabling efficient use of longer histories.

4.1 Problem Formulation

CUAs operate over trajectories $\{(I_t, T_t, A_t)\}_{t=1}^{T}$, where I_t is the screenshot, T_t is the accumulated textual context (all action and reasoning from the preiovus steps and the task description), and A_t is the predicted action at step t. Each image is encoded into visual tokens $V_t = \{v_1^t, \ldots, v_N^t\}$, and the model conditions on $\{V_1, \ldots, V_t\}$ and $\{T_1, \ldots, T_t\}$ to generate the next reasoning and action. As t increases, the number of visual tokens grows linearly, introducing substantial redundancy due to high similarity between consecutive screenshots. As illustrated in Figure 2, our goal is to construct a reduced set of tokens $V_t' \subseteq V_t$ that preserves task-relevant information while removing redundancy. To achieve this, for each step t, we compute a binary mask $\mathbf{m}_t \in \{0,1\}^N$ by comparing two corresponding patches from I_{t-1} and I_t, where $\mathbf{m}_t[j] = 1$ indicates that the j-th patch in I_t is retained. The filtered tokens are then given by $V_t' = V_t[\mathbf{m}_t]$, and the model uses $\{V_1', \ldots, V_t'\}$ together with $\{T_1, \ldots, T_t\}$ to generate the next action and reasoning.

4.2 Training

Training the RTS classifier. We train the ReVision Token Selection (RTS) module as a lightweight three-layer MLP that predicts whether a patch in the current image is redundant given corresponding patch from the previous image. To obtain supervision, we use OmniParserV2 (Lu et al., 2024) to segment screenshots into semantic regions and match regions across consecutive images. This allows us to generate labels based on region overlap (IoU), identifying which patches correspond to unchanged content. We adopt this approach instead of relying on raw pixel or embedding similarity, as region-level matching is more robust to small visual variations (e.g., rendering noise, cursor movement) while still capturing semantic redundancy.

Following the formulation in Section 4.1 and the procedure outlined in Algorithm 1, we construct training data from AgentNet trajectories by applying RTS to each pair of consecutive images. At

Algorithm 1 ReVision Visual Token Dropping

Require: Input: current step N, image window size k, task information q, previous actions $\mathcal{A}_{1:N-1}$, previous reasoning $\mathcal{R}_{1:N-1}$, recent images $\mathcal{I}_{N-k+1:N} = [I_{N-k+1}, \ldots, I_N]$

$\mathbf{x} \leftarrow \textsc{Tokenizer}\big(q, \mathcal{A}_{1:N-1}, \mathcal{R}_{1:N-1}, \langle\text{image}\rangle^k\big)$ ▷ Insert one <IMAGE> placeholder per image in its actual step
$\mathcal{V} \leftarrow [\,], \quad \mathcal{P} \leftarrow [\,]$
for $i = N - k + 1$ to N **do**
  $(\mathbf{u}_i, \mathbf{p}_i) \leftarrow \textsc{VisionEncoder}(I_i)$
  $(\mathbf{v}_i, \mathbf{f}_i) \leftarrow \textsc{ProjectionLayer}(\mathbf{u}_i)$ ▷ $\mathbf{v}_i$: visual tokens, $\mathbf{f}_i$: patch features, $\mathbf{p}_i$: original position ids
  if $i = N - k + 1$ **then**
    $\hat{\mathbf{v}}_i \leftarrow \mathbf{v}_i$
    $\hat{\mathbf{p}}_i \leftarrow \mathbf{p}_i$ ▷ Keep the first image in the window intact
  else
    $\mathbf{m}_i \leftarrow \textsc{ReVisionTokenSelection}(\mathbf{f}_{i-1}, \mathbf{f}_i)$ ▷ $\mathbf{m}_i$ is a binary indicator of which features/tokens to keep
    $\hat{\mathbf{v}}_i \leftarrow \mathbf{v}_i[\mathbf{m}_i]$
    $\hat{\mathbf{p}}_i \leftarrow \mathbf{p}_i[\mathbf{m}_i]$ ▷ Retained tokens keep their original position ids
  end if
  append $\hat{\mathbf{v}}_i$ to $\mathcal{V}$
  append $\hat{\mathbf{p}}_i$ to $\mathcal{P}$
end for
$(\mathbf{e}, \mathbf{pos}) \leftarrow \textsc{BuildMultimodalInput}(\mathbf{x}, \mathcal{V}, \mathcal{P})$ ▷ Pass all previous actions and reasoning, but only the last k images
return $\textsc{LLMDecoder}(\mathbf{e}, \mathbf{pos})$

each step t, RTS compares each patch in I_t with its corresponding patch in I_{t-1} to produce a mask $\mathbf{m}_t$, which is then used to obtain the filtered tokens V_t'. The first image in each window is kept unchanged. This results in sequences where the model observes $\{V_1', \ldots, V_t'\}$ together with the full textual context $\{T_1, \ldots, T_t\}$ to generate the next reasoning and action. We then fine-tune the MLLM on these filtered trajectories, training it to operate under partially observed visual inputs where redundant patches are removed and must be implicitly recovered from previous steps.

Training MLLM with filtered trajectories. RTS is applied as a plug-in token filtering mechanism on top of MLLMs, following the same formulation and pipeline used during both training and inference (Algorithm 1). At each step t, the model conditions on the full textual context $\{T_1, \ldots, T_t\}$, while only the most recent k images within a fixed history window are included from the trajectory. Each image I_t is encoded into visual tokens V_t, and for consecutive images, RTS compares each patch in I_t with its corresponding patch in I_{t-1} to produce a binary mask $\mathbf{m}_t$, which is used to construct the filtered tokens $V_t' = V_t[\mathbf{m}_t]$. The first image in the window is kept unchanged, while subsequent images retain only non-redundant patches. The model then operates on $\{V_1', \ldots, V_t'\}$ together with $\{T_1, \ldots, T_t\}$ to generate the next reasoning and action. We construct training samples from AgentNet (Wang et al., 2025b) trajectories using this sliding window of k, ensuring that all trajectories are used while only images within the history window are provided at each step. By training under the same filtered setting used at inference time, the model learns to recover missing visual information from earlier images, enabling efficient use of longer histories without requiring full observations at every step.

5 Experiments and Results

5.1 Experimental Settings

We build on the OpenCUA framework (Wang et al., 2025b) and follow its default setup. We train on AgentNet (Wang et al., 2025b), using a separate model for each history window size k to match training and inference conditions. Training uses standard autoregressive next-token prediction with the same optimizer and hyperparameters as OpenCUA, and decoding temperature is fixed to T=0.0 to isolate the effect of token filtering. All results are averaged over three runs, with error analysis in Appendix C. Additional training details and metrics are provided in Appendix I.

Benchmarks & Metrics. We evaluate on OSWorld (Xie et al., 2024), AgentNetBench (Wang et al., 2025b), and WebTailBench (Awadallah et al., 2025), covering long-horizon desktop and web-based tasks. We report success rate (SR), defined as the percentage of tasks completed successfully. For OSWorld and WebTailBench, we report SR under different step budgets, following their interactive evaluation settings. AgentNetBench is an offline benchmark with fixed trajectories, so we report average SR over its task categories. For WebTailBench, we execute tasks in the OSWorld environment and use an LLM-as-a-judge (GPT-

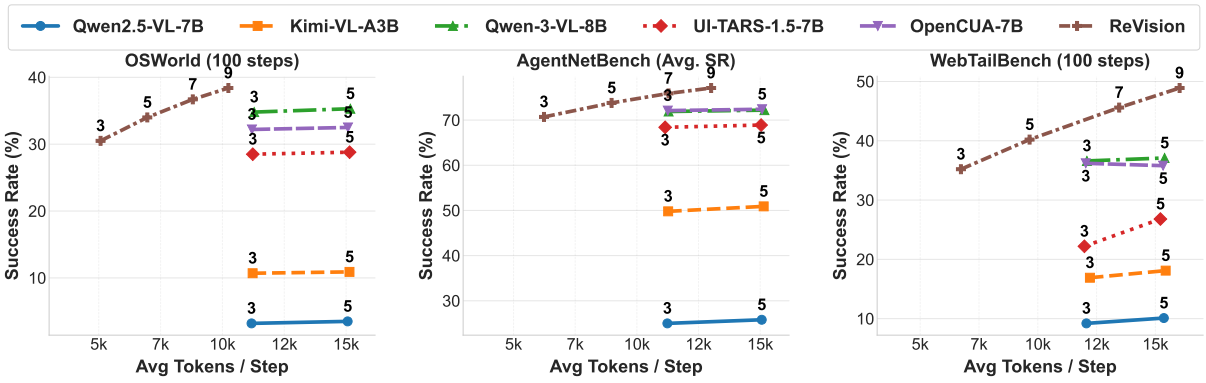


Figure 3: Success rate versus average tokens per step across OSWorld at 100 steps, AgentNetBench, and WebTailBench at 100 steps. REVISION consistently achieves high success rates at comparable or lower token budgets, effectively shifting the efficiency frontier. Detailed numerical results are provided in Tables 7, 8, and 9 in Appendix B. See Figure 7 in Appendix G for results on OSWorld at 15 steps, 50 steps, and WebTailBench at 50 steps.

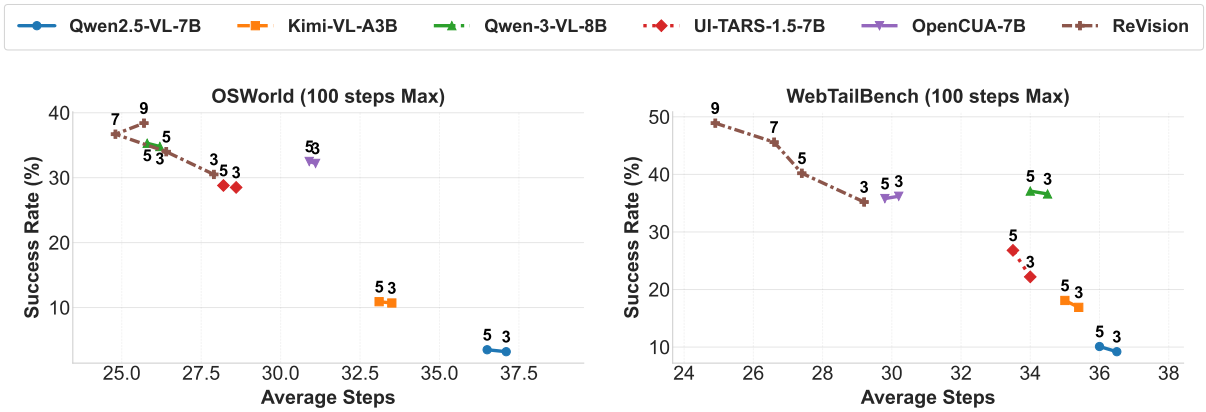


Figure 4: Success rate versus average number of steps for OSWorld and WebTailBench at 100 steps. REVISION achieves higher success rates with fewer steps, indicating more efficient decision-making. Detailed numerical results are provided in Tables 8 and 9 in Appendix B. See Figure 8 in Appendix G for results on OSWorld and WebTailBench at 50 steps.

4O (OpenAI, 2024)) to assess step-level correctness and compute SR. See Appendix A for more details on the benchmarks.

Baselines. We compare REVISION with general vision-language models, including QWEN-2.5-VL (Bai et al., 2025b), QWEN-3-VL (Team, 2025; Bai et al., 2025a), and KIMI-VL-A3B (Team et al., 2025), as well as specialized UI agents such as UI-TARS (Wang et al., 2025a; Qin et al., 2025) and OPENCUA (Wang et al., 2025b). *For these baselines, we provide all k history images without applying any patch-removal strategy. We do so because these models are trained to process full images, and naive token removal degrades their performance;* ***effective token reduction therefore requires model fine-tuning***. Accordingly, token counts are reported to show that REVISION can achieve strong performance with substantially fewer visual tokens, rather than as a direct efficiency comparison against baselines under token pruning. To isolate the effect of token removal, we include **REVISION No Drop**, which uses the same training setup as REVISION but disables patch removal at inference time. All baselines and REVISION variants receive the full history of reasoning traces and actions. Unless otherwise specified, all REVISION models, including REVISION No Drop, use QWEN2.5-VL-7B.

5.2 Efficiency-Performance Trade-offs

We analyze the trade-off between task performance and computational cost through two views: success rate versus token usage and success rate versus trajectory length. At each step, the agent receives the last k images along with the full textual context (reasoning and actions), where the history constraint applies only to images.

Performance vs. Token Usage. Figure 3

shows that adding more history images yields only marginal SR gains for baselines while substantially increasing token usage, suggesting that redundant information accumulates across consecutive GUI observations. REVISION scales differently: it uses far fewer tokens while consistently matching or outperforming baselines across all benchmarks. It improves SR by up to **7 points** on OSWorld and AgentNetBench and up to **14 points** on WebTailBench, while using **34% fewer tokens per image**. This efficiency enables **9-image histories** under a token budget comparable to **5-image** baselines. On WebTailBench, REVISION reaches nearly **50% SR** (**42% relative improvement**) versus below **30%** for strong baselines. These results show that many visual tokens in GUI trajectories are redundant, and that removing them can improve—not hurt—performance by producing a more compact and informative history. See Appendix J for a qualitative case study.

Performance vs. Trajectory Length. Figure 4 shows the relationship between success rate and trajectory length, measuring how efficiently models solve tasks as history is incorporated. REVISION consistently achieves higher success rates with fewer steps. On OSWorld, REVISION reduces average trajectory length by up to **4 steps** while improving success rate. On WebTailBench, the gains are stronger: REVISION reduces trajectories by up to **4 steps** and improves success rate by up to **14 points**. While strong baselines require 33–37 steps and still remain below **40%** SR, REVISION reaches nearly **50%** SR with only ∼**25–30 steps**. One exception is OSWorld with **9-image histories**, where REVISION shows a slight increase in steps despite strong performance, possibly due to over-reasoning from longer histories. Since this pattern does not appear at lower step budgets (SR@15 and SR@50) or on other benchmarks, it appears specific to longer-horizon OSWorld settings. Overall, these results show that redundancy-aware token filtering improves both task success and decision efficiency.

5.3 Effect of Using Different Token Selection

In Table 2, we compare REVISION with random, spiral, pixel-similarity, and embedding-similarity (QWEN2.5-VL-7B and DINOV2-BASE (Oquab et al., 2024)) token selection strategies. Naive dropping reduces tokens but consistently hurts performance, with aggressive removal (Random 90%) causing catastrophic failure. Pixel-based filtering compresses more but relies on noisy low-level similarity, while embedding-based methods better preserve performance but still do not outperform no dropping. In contrast, **RTS provides the best performance–efficiency trade-off**, improving success rate over no dropping (73.8 vs. 72.5 on AgentNet; 34.0 vs. 32.3 on OSWorld) while reducing tokens per step by **48%** on average. OmniParserV2 achieves the highest success rates but incurs **high latency** (> 550 ms), whereas REVISION obtains comparable gains with only ∼22 ms latency. These results highlight the importance of *semantic and temporal awareness* for effective token selection (see Appendix H).

Strategy	AgentNet		OSWorld		Lat. (ms)
	SR	Tok/Step	SR@100	Tok/Step	
No Drop	72.5	15076	32.3	15071	0
Random (50%)	67.9	9952	27.8	9788	0
Random (90%)	18.9	4234	4.6	4385	0
Spiral (50%)	69.4	9821	29.0	9662	0
Pixel	68.4	8213	28.6	6125	20
Qwen2.5-VL-7B (CosSim)	72.3	9424	32.1	7624	7
DINOv2-base (CosSim)	71.7	9682	31.4	7915	28.5
RTS + OmniParserV2	74.6	8420	35.2	6485	565
RTS (Ours)	73.8	8975	34.0	6963	22.5

Table 2: **Comparison of different token selection strategies.** Lat. is average latency. Moderate dropping reduces tokens but degrades performance, while aggressive removal (Random 90%) causes catastrophic failure. RTS achieves the best performance-efficiency trade-off with low average latency, whereas region-based methods (OmniParserV2) improve performance at significantly higher cost. See Appendix F for qualitative analysis on different removal strategies.

5.4 Visual History Scaling and Saturation

We analyze how increasing the number of history images affects performance and token usage for the *No Drop* baseline and REVISION. As shown in Figure 5, performance initially improves with longer histories but eventually saturates: the *No Drop* baseline peaks earlier, around 7 images, and then declines, while REVISION continues improving up to larger windows, around 11 images, before plateauing. Notably, saturation aligns more closely with total context length than with the number of images, occurring at approximately **23k tokens** across benchmarks. By removing redundant visual tokens, REVISION compresses the context and delays saturation, enabling more effective use of longer histories within the same token budget.

To better understand this behavior, we compare two token-removal directions in Table 3. The default REVISION uses *forward* removal, which re-

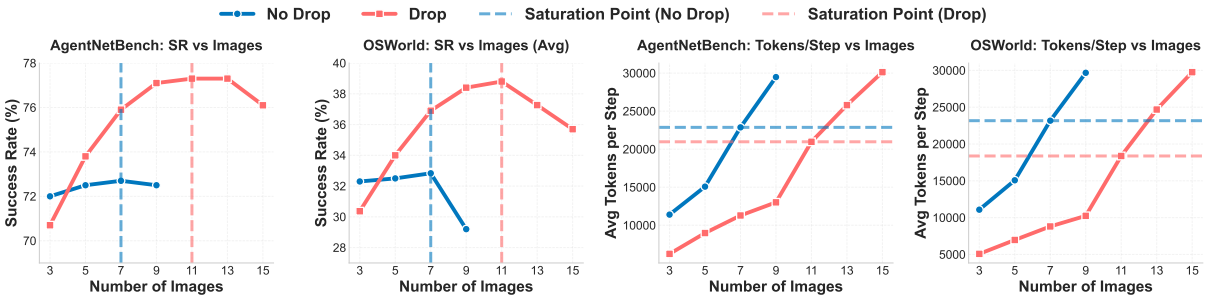


Figure 5: **Visual History Scaling.** As the number of history images increases, the No Drop baseline saturates early due to rising token usage, while REVISION removes redundant tokens, delaying saturation and achieving higher performance under a similar budget.

Benchmark	History	NoDrop	Forward	Backward
AgentNetBench	$H = 3$	**72.0**	70.7	71.4
	$H = 5$	72.5	**73.8**	70.2
	$H = 9$	72.5	**77.1**	68.9
WebTailBench	$H = 3$	**36.3**	35.2	35.6
	$H = 5$	36.0	**40.2**	33.8
	$H = 9$	35.1	**48.9**	31.6
OSWorld	$H = 3$	**32.2**	30.5	31.4
	$H = 5$	32.3	**34.0**	29.6
	$H = 9$	29.4	**38.4**	27.8

Table 3: Effect of forward vs. backward token selection across benchmarks and history lengths. Best results within each benchmark and history setting are bolded.

moves redundant patches from later screenshots, including the final observation. We also train a *backward* variant that keeps the final image intact and removes tokens from earlier history images. Forward removal performs best for $H = 5$ and $H = 9$ across all benchmarks, while backward removal degrades performance, especially with longer histories. This suggests that an overly complete final image can reduce the model's reliance on history; filtering it encourages the use of temporally distributed evidence. These results support our design choice and show that REVISION improves long-horizon reasoning by both reducing tokens and promoting better use of visual history. See Appendix E for window-size generalization analysis.

5.5 Ablations

REVISION Generalizes Across Models. We evaluate whether **REVISION** generalizes beyond a single backbone by comparing its performance across two model families: *Qwen2.5-VL-7B* and *Qwen3-VL-8B*. We report results for history window sizes of 3 and 5 images on *OSWorld* and *AgentNetBench*. As shown in Table 4, increasing the history size consistently improves performance for both model families while maintaining predictable scaling in token usage. Notably, the relative gains are consistent across benchmarks and architectures, indicating that REVISION generalizes effectively beyond a specific backbone. We observe that the improvement margins for *Qwen3-VL-8B* are slightly smaller, which may be attributed to its stronger baseline performance on computer-use tasks, leaving less room for improvement.

Base Model	History	OSWorld	AgentNetBench
Qwen2.5-VL-7B	3	30.5	70.7
	5	34.0	73.8
Qwen3-VL-8B	3	34.1	73.5
	5	36.7	76.0

Table 4: Generalization of REVISION across model families using 3 and 5 image history windows. We report SR@100 for OSWorld and average SR for AgentNetBench. Full results are provided in Appendix D.

Model	OSWorld-G	ScreenSpot-Pro	UI-Vision
Qwen2.5-VL-7B	31.3	27.8	0.85
Qwen3-VL-7B	57.8	55.3	27.6
REVISION (Qwen2.5-VL-7B)	31.1	27.6	0.83
REVISION (Qwen3-VL-8B)	57.5	55.6	27.2

Table 5: Performance comparison in the single-image setting across GUI grounding benchmarks. REVISION achieves comparable performance to the base models, confirming that training with temporal token filtering does not degrade single-image grounding ability.

REVISION Does not Hurt Performance with a Single Image. To verify that training with **REVISION** does not degrade performance in the standard single-image setting, we evaluate our models on four GUI grounding benchmarks: *OSWorld-G* (Xie et al., 2025), *ScreenSpot-Pro* (Li et al., 2025), and *UI-Vision* (Nayak et al., 2025). In this setting, no historical images are provided, and thus REVISION 's token filtering mechanism is effectively inactive

at inference time. This experiment isolates whether the modified training distribution, where redundant visual tokens are removed across trajectories, negatively impacts performance when only a single screenshot is available. As shown in Table 5, REVISION achieves performance comparable to the base models across all benchmarks, with only minor variations. These results indicate that training with filtered visual inputs does not harm the model's grounding ability in the single-image regime.

6 Conclusion

In this work, we introduced **ReVision**, a redundancy-aware history representation for computer-use agents that reduces unnecessary visual tokens by explicitly modeling temporal redundancy across consecutive screenshots. Our results show that a substantial portion of visual context in GUI trajectories is redundant, and that removing these tokens improves both efficiency and task performance. Across multiple benchmarks, ReVision consistently achieves higher success rates while using fewer tokens and shorter trajectories, demonstrating that better history representation can improve decision-making in long-horizon computer-use tasks. More broadly, our findings suggest that the key challenge in scaling visual reasoning is not simply the number of past images, but how much useful information can be preserved within a limited context budget. Looking forward, an important direction is to extend redundancy modeling beyond time to also capture spatial redundancy within screenshots, and to better understand the mechanisms behind performance saturation in long-context multimodal reasoning.

Limitations

REVISION has a few limitations worth noting. First, its effectiveness depends on the quality of the learned redundancy predictor. Although RTS is lightweight and trained to identify temporally redundant visual patches, errors in this classifier can remove patches that later become important for GUI reasoning, especially when small visual changes encode critical state transitions. This risk is particularly relevant for interfaces with subtle updates, dense text, small icons, or dynamically changing elements. Second, REVISION is designed around temporal redundancy in screenshot histories, and therefore provides the largest gains in long-horizon GUI tasks where consecutive screenshots contain substantial visual overlap. In settings with highly dynamic visual environments, rapid scene changes, or tasks that require detailed comparison of every frame, the achievable compression may be lower. Similarly, while preserving positional indices helps maintain compatibility with m-RoPE and spatial reasoning, token removal is still a lossy operation and may affect tasks requiring pixel-level precision. Finally, our experiments focus on GUI-based computer-use agents and vision-language models. While the core idea of removing temporally redundant visual tokens may generalize to other multimodal settings, such as video understanding or robotic perception, we do not claim that the current implementation directly transfers to those domains without additional adaptation.

Ethical Considerations

Potential Risks. REVISION is designed to improve the efficiency of multimodal computer-use agents by reducing redundant visual tokens from screenshot histories. More efficient agents can make long-horizon GUI automation cheaper and more accessible, but they may also amplify risks associated with autonomous computer-use systems. For example, if deployed without appropriate safeguards, such agents could perform unintended actions in user interfaces, mishandle sensitive information shown on screen, or interact with web services in ways that violate user expectations or platform policies. Since REVISION changes the visual context available to the agent, incorrect token pruning could also lead to mistaken actions when important interface details are removed. We therefore recommend that REVISION be used with human oversight, action confirmation mechanisms for sensitive operations, and careful evaluation before deployment in real-world environments.

Privacy and Data Considerations. REVISION operates on screenshot trajectories, which may contain sensitive information such as names, emails, account details, documents, or other private user content. Our work uses publicly available benchmarks and does not collect new private user data. However, future deployments should treat screenshots as sensitive data, avoid unnecessary storage of raw trajectories, and apply appropriate anonymization, access control, and data retention policies. Efficiency improvements should not be interpreted as eliminating the need for privacy-

preserving handling of visual interaction data.

Use of AI Assistance. During the preparation of this manuscript, we used large language models for paraphrasing and improving the writing quality of the paper, and for assisting with figure and caption refinement. All ideas, technical contributions, experimental design, implementation, and conclusions are entirely original and the work of the authors.

Reproducibility Statement. We describe the architecture of REVISION, the RTS training setup, token selection procedure, benchmarks, baselines, and evaluation settings in Section 5.1. We release our code to support reproducibility. All external benchmarks, datasets, models, and baseline artifacts used in this paper are publicly available, and our use of them is limited to research evaluation, consistent with their original licenses, terms of use, and intended access conditions. Our released artifacts are intended for research purposes and for reproducing the experiments reported in this paper.

Use of Existing Artifacts and Data. We use only publicly available benchmarks, datasets, models, and baseline artifacts. We do not collect new user data or attempt to identify individuals. Some benchmark trajectories may contain web-derived or interface-specific text from the original datasets; we follow the original preprocessing, licenses, and access conditions of those datasets. All benchmarks, datasets, and baselines used in this paper are openly available, and our use of them is consistent with research evaluation and reproducibility.

References

Reyna Abhyankar, Qi Qi, and Yiying Zhang. 2025. Osworld-human: Benchmarking the efficiency of computer-use agents. *Preprint*, arXiv:2506.16042.

Ahmed Awadallah, Yash Lara, Raghav Magazine, Hussein Mozannar, Akshay Nambi, Yash Pandya, Aravind Rajeswaran, Corby Rosset, Alexey Taymanov, Vibhav Vineet, Spencer Whitehead, and Andrew Zhao. 2025. Fara-7b: An efficient agentic model for computer use. *Preprint*, arXiv:2511.19663.

Shuai Bai, Yuxuan Cai, Ruizhe Chen, Keqin Chen, Xionghui Chen, Zesen Cheng, Lianghao Deng, Wei Ding, Chang Gao, Chunjiang Ge, and 1 others. 2025a. Qwen3-vl technical report. *arXiv preprint arXiv:2511.21631*.

Shuai Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Sibo Song, Kai Dang, Peng Wang, Shijie Wang, Jun Tang, Humen Zhong, Yuanzhi Zhu, Mingkun Yang, Zhaohai Li, Jianqiang Wan, Pengfei Wang, Wei Ding, Zheren Fu, Yiheng Xu, and 8 others. 2025b. Qwen2.5-vl technical report. *arXiv preprint arXiv:2502.13923*.

Rogerio Bonatti, Dan Zhao, Francesco Bonacci, Dillon Dupont, Sara Abdali, Yinheng Li, Justin Wagle, Kazuhito Koishida, Arthur Bucker, Lawrence Jang, and Zack Hui. 2024. Windows agent arena: Evaluating multi-modal os agents at scale.

Guoxin Chen, Zile Qiao, Xuanzhong Chen, Donglei Yu, Haotian Xu, Xin Zhao, Ruihua Song, Wenbiao Yin, Huifeng Yin, Liwen Zhang, Kuan Li, Minpeng Liao, Yong Jiang, Pengjun Xie, Fei Huang, and Jingren Zhou. 2026. Iterresearch: Rethinking long-horizon agents via markovian state reconstruction. In *The Fourteenth International Conference on Learning Representations*.

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. 2025. 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, Vienna, Austria. Association for Computational Linguistics.

Joonmyung Choi, Sanghyeok Lee, Jaewon Chu, Minhyuk Choi, and Hyunwoo J. Kim. 2024. vid-tldr: Training free token merging for light-weight video transformer. In *Conference on Computer Vision and Pattern Recognition*.

Rohan Choudhury, Guanglei Zhu, Sihan Liu, Koichiro Niinuma, Kris M. Kitani, and Laszlo Attila Jeni. 2024. Don't look twice: Faster video transformers with run-length tokenization. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

Boyu Gou, Zanming Huang, Yuting Ning, Yu Gu, Michael Lin, Botao Yu, Andrei Kopanev, Weijian Qi, Yiheng Shu, Jiaman Wu, Chan Hee Song, Bernal Jimenez Gutierrez, Yifei Li, Zeyi Liao, Hanane Nour Moussa, TIANSHU ZHANG, Jian Xie, Tianci Xue, Shijie Chen, and 7 others. 2025. Mind2web 2: Evaluating agentic search with agent-as-a-judge. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.

Hongliang He, Wenlin Yao, Kaixin Ma, Wenhao Yu, Yong Dai, Hongming Zhang, Zhenzhong Lan, and Dong Yu. 2024. WebVoyager: Building an end-to-end web agent with large multimodal models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 6864–6890, Bangkok, Thailand. Association for Computational Linguistics.

Yifei He, Pranit Chawla, Yaser Souri, Subhojit Som, and Xia Song. 2026. Webstar: Scalable data synthe-

sis for computer use agents with step-level filtering. *Preprint*, arXiv:2512.10962.

Wenyi Hong, Weihan Wang, Qingsong Lv, Jiazheng Xu, Wenmeng Yu, Junhui Ji, Yan Wang, Zihan Wang, Yuxiao Dong, Ming Ding, and Jie Tang. 2023. Cogagent: A visual language model for gui agents. *Preprint*, arXiv:2312.08914.

Mouxiao Huang, Borui Jiang, Dehua Zheng, Hailin Hu, Kai Han, and Xinghao Chen. 2026a. PPE: Positional preservation embedding for token compression in multimodal large language models. In *The Fourteenth International Conference on Learning Representations*.

Yihong Huang, Fei Ma, Yihua Shao, Jingcai Guo, Zitong YU, Laizhong Cui, and Qi Tian. 2026b. Nüwa: Mending the spatial integrity torn by VLM token pruning. In *The Fourteenth International Conference on Learning Representations*.

Imene Kerboua, Sahar Omidi Shayegan, Megh Thakkar, Xing Han Lù, Léo Boisvert, Massimo Caccia, Jérémy Espinas, Alexandre Aussem, Véronique Eglin, and Alexandre Lacoste. 2025. Focusagent: Simple yet effective ways of trimming the large context of web agents. *Preprint*, arXiv:2510.03204.

Jing Yu Koh, Robert Lo, Lawrence Jang, Vikram Duvvur, Ming Lim, Po-Yu Huang, Graham Neubig, Shuyan Zhou, Russ Salakhutdinov, and Daniel Fried. 2024a. VisualWebArena: Evaluating multimodal agents on realistic visual web tasks. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 881–905, Bangkok, Thailand. Association for Computational Linguistics.

Jing Yu Koh, Robert Lo, Lawrence Jang, Vikram Duvvur, Ming Lim, Po-Yu Huang, Graham Neubig, Shuyan Zhou, Russ Salakhutdinov, and Daniel Fried. 2024b. VisualWebArena: Evaluating multimodal agents on realistic visual web tasks. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 881–905, Bangkok, Thailand. Association for Computational Linguistics.

Bruno Korbar, Du Tran, and Lorenzo Torresani. 2019. Scsampler: Sampling salient clips from video for efficient action recognition. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 6232–6242.

Kaixin Li, Meng Ziyang, Hongzhan Lin, Ziyang Luo, Yuchen Tian, Jing Ma, Zhiyong Huang, and Tat-Seng Chua. 2025. Screenspot-pro: GUI grounding for professional high-resolution computer use. In *Workshop on Reasoning and Planning for Large Language Models*.

Kevin Qinghong Lin, Linjie Li, Difei Gao, Zhengyuan Yang, Shiwei Wu, Zechen Bai, Weixian Lei, Lijuan Wang, and Mike Zheng Shou. 2024. Showui: One vision-language-action model for gui visual agent. *Preprint*, arXiv:2411.17465.

Zhaoyang Liu, Jingjing Xie, Zichen Ding, Zehao Li, Bowen Yang, Zhenyu Wu, Xuehui Wang, Qiushi Sun, Shi Liu, Weiyun Wang, Shenglong Ye, Qingyun Li, Xuan Dong, Yue Yu, Chenyu Lu, YunXiang Mo, Yao Yan, Zeyue Tian, Xiao Zhang, and 11 others. 2025. Scalecua: Scaling open-source computer use agents with cross-platform data. *arXiv preprint arXiv:2509.15221*. Preprint.

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

Shravan Nayak, Xiangru Jian, Kevin Qinghong Lin, Juan A. Rodriguez, Montek Kalsi, Rabiul Awal, Nicolas Chapados, M. Tamer Özsu, Aishwarya Agrawal, David Vazquez, Christopher Pal, Perouz Taslakian, Spandana Gella, and Sai Rajeswar. 2025. Ui-vision: A desktop-centric gui benchmark for visual perception and interaction. *Preprint*, arXiv:2503.15661.

OpenAI. 2024. Gpt-4o.

Maxime Oquab, Timothée Darcet, Théo Moutakanni, Huy Vo, Marc Szafraniec, Vasil Khalidov, Pierre Fernandez, Daniel Haziza, Francisco Massa, Alaaeldin El-Nouby, Mahmoud Assran, Nicolas Ballas, Wojciech Galuba, Russell Howes, Po-Yao Huang, Shang-Wen Li, Ishan Misra, Michael Rabbat, Vasu Sharma, and 7 others. 2024. Dinov2: Learning robust visual features without supervision. *Preprint*, arXiv:2304.07193.

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

Yujia Qin, Yining Ye, Junjie Fang, Haoming Wang, Shihao Liang, Shizuo Tian, Junda Zhang, Jiahao Li, Yunxin Li, Shijue Huang, Wanjun Zhong, Kuanye Li, Jiale Yang, Yu Miao, Woyu Lin, Longxiang Liu, Xu Jiang, Qianli Ma, Jingyu Li, and 16 others. 2025. Ui-tars: Pioneering automated gui interaction with native agents. *Preprint*, arXiv:2501.12326.

Christopher Rawles, Sarah Clinckemaillie, Yifan Chang, Jonathan Waltz, Gabrielle Lau, Marybeth Fair, Alice Li, William Bishop, Wei Li, Folawiyo Campbell-Ajala, Daniel Toyama, Robert Berry, Divya Tyamagundlu, Timothy Lillicrap, and Oriana Riva. 2024. Androidworld: A dynamic benchmarking environment for autonomous agents. *Preprint*, arXiv:2405.14573.

Pascal J. Sager, Benjamin Meyer, Peng Yan, Rebekka Von Wartburg-Kottler, Layan Etaiwi, Aref Enayati, Gabriel Nobel, Ahmed Abdulkadir, Benjamin F. Grewe, and Thilo Stadelmann. 2026. A comprehensive survey of agents for computer use: Foundations, challenges, and future directions. *Journal of Artificial Intelligence Research*, 85.

Keda Tao, Can Qin, Haoxuan You, Yang Sui, and Huan Wang. 2025. Dycoke: Dynamic compression of tokens for fast video large language models. In *Pro-*

ceedings of the Computer Vision and Pattern Recognition Conference, pages 18992–19001.

Kimi Team, Angang Du, Bohong Yin, Bowei Xing, Bowen Qu, Bowen Wang, Cheng Chen, Chenlin Zhang, Chenzhuang Du, Chu Wei, Congcong Wang, Dehao Zhang, Dikang Du, Dongliang Wang, Enming Yuan, Enzhe Lu, Fang Li, Flood Sung, Guangda Wei, and 73 others. 2025. Kimi-VL technical report. *Preprint*, arXiv:2504.07491.

Qwen Team. 2025. Qwen3 technical report. *Preprint*, arXiv:2505.09388.

Enwei Tong, Yuanchao Bai, Yao Zhu, Junjun Jiang, and Xianming Liu. 2026. Focus-scan-refine: From human visual perception to efficient visual token pruning. *Preprint*, arXiv:2602.05809.

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

Xinyuan Wang, Bowen Wang, Dunjie Lu, Junlin Yang, Tianbao Xie, Junli Wang, Jiaqi Deng, Xiaole Guo, Yiheng Xu, Chen Henry Wu, Zhennan Shen, Zhuokai Li, Ryan Li, Xiaochuan Li, Junda Chen, Boyuan Zheng, Peihang Li, Fangyu Lei, Ruisheng Cao, and 23 others. 2025b. Opencua: Open foundations for computer-use agents. *Preprint*, arXiv:2508.09123.

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. 2025. Scaling computer-use grounding via user interface decomposition and synthesis. *Preprint*, arXiv:2505.13227.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, Yitao Liu, Yiheng Xu, Shuyan Zhou, Silvio Savarese, Caiming Xiong, Victor Zhong, and Tao Yu. 2024. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments. *Preprint*, arXiv:2404.07972.

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

Linli Yao, Yicheng Li, Yuancheng Wei, Lei Li, Shuhuai Ren, Yuanxin Liu, Kun Ouyang, Lean Wang, Shicheng Li, Sida Li, Lingpeng Kong, Qi Liu, Yuanxing Zhang, and Xu Sun. 2025. Timechat-online: 80% visual tokens are naturally redundant in streaming videos. In *Proceedings of the 33rd ACM International Conference on Multimedia*, MM '25, page 10807–10816, New York, NY, USA. Association for Computing Machinery.

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. 2022. Webshop: Towards scalable real-world web interaction with grounded language agents. In *Advances in Neural Information Processing Systems*, volume 35, pages 20744–20757. Curran Associates, Inc.

Boyuan Zheng, Boyu Gou, Jihyung Kil, Huan Sun, and Yu Su. 2024. Gpt-4v(ision) is a generalist web agent, if grounded. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org.

Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Yonatan Bisk, Daniel Fried, Uri Alon, and 1 others. 2023. Webarena: A realistic web environment for building autonomous agents. *arXiv preprint arXiv:2307.13854*.

A Benchmark Details

We evaluate REVISION on three computer-use benchmarks: OSWorld, WebTailBench, and AgentNetBench. These benchmarks cover complementary settings, including interactive desktop control, web-based long-horizon tasks, and offline trajectory-based evaluation. Table 6 summarizes their main characteristics.

OSWorld (Xie et al., 2024) evaluates agents in realistic desktop environments with tasks involving web browsing, file manipulation, office applications, system settings, and multi-application workflows. Each task provides an initial environment state and an instruction, and the agent must interact with the GUI until the task is completed or the step budget is exhausted. We report success rate under multiple step budgets, SR@15, SR@50, and SR@100, to measure both short-horizon and longer-horizon task completion. This benchmark is particularly useful for evaluating REVISION because many tasks require reasoning over repeated screenshots where large portions of the interface remain unchanged across steps.

WebTailBench (Awadallah et al., 2025) focuses on web-based computer-use tasks and contains longer-tailed task categories that are underrepresented in existing GUI-agent benchmarks. We run WebTailBench in the OSWorld-style interactive environment and report SR@50 and SR@100. Since WebTailBench tasks often require navigating multi-step web workflows, they benefit from maintaining useful visual history. At the same time, consecutive browser screenshots contain substantial redundancy, making the benchmark a strong testbed for

Benchmark	Setting	# Tasks	Evaluation	Reported Metric
OSWorld	Desktop GUI	369	Interactive execution	SR@15 / SR@50 / SR@100
WebTailBench	Web GUI	609	Interactive execution	SR@50 / SR@100
AgentNetBench	Desktop GUI	1091	Offline trajectory evaluation	Func. SR / Content SR / Coord. SR / Avg. SR

Table 6: Summary of the benchmarks used in our evaluation. OSWorld and WebTailBench are evaluated interactively under different step budgets, while AgentNetBench is an offline benchmark with fixed trajectories. SR denotes success rate.

evaluating whether REVISION can preserve temporally useful evidence while reducing unnecessary visual tokens.

AgentNetBench (Wang et al., 2025b) is an offline benchmark derived from held-out AgentNet trajectories. Unlike OSWorld and WebTailBench, it does not require live environment interaction during evaluation; instead, models are evaluated on fixed trajectory states and asked to predict appropriate actions. This setting enables faster and more reproducible evaluation across model variants. We report the average success rate across task categories. AgentNetBench complements the interactive benchmarks by isolating the model's ability to reason over visual histories without confounding factors from environment execution.

B Detailed Results Tables

We provide the full numerical results corresponding to the efficiency–performance trade-off analysis in Section 5.2.

OSWorld. Table 7 reports detailed results on OSWorld across general VLMs, UI agents, and ReVision. Across all model families, increasing the number of history images leads to consistent but diminishing improvements in success rate. Moving from one to three images provides the largest gains, while increasing to five images yields only marginal improvements despite a substantial increase in token usage (e.g., from $\sim$4k to $\sim$15k tokens per step). This trend suggests that short-term visual context is beneficial, but additional history quickly becomes redundant. A similar pattern is observed in trajectory length, where increasing history results in only minor reductions in the number of steps, indicating limited improvements in action efficiency.

ReVision exhibits a different scaling behavior. Without token dropping, it matches the performance of the corresponding baselines, confirming that the training procedure does not degrade performance. When redundant visual tokens are removed, ReVision consistently improves both efficiency and performance. For example, with Qwen2.5-VL-7B and a 5-image history, ReVision improves 50-step success rate from 34.5 to 35.9 while reducing tokens per step by more than 2$\times$ (15,071 $\rightarrow$ 6,963) and decreasing the average number of steps (22.7 $\rightarrow$ 19.8). Similar trends are observed across step budgets and backbones. These results indicate that a significant portion of visual tokens is redundant, and that removing them not only reduces computational cost but also leads to more efficient decision-making by enabling the model to better focus on relevant actions and observations.

AgentNetBench. Table 8 reports results on AgentNetBench across general VLMs, UI agents, and ReVision. Similar to OSWorld, increasing the number of history images leads to consistent improvements across all models, with gains saturating beyond 3 to 5 images. This trend holds across coordinate, content, and functional success rates, indicating that additional short-term visual context is beneficial but provides diminishing returns as history grows. Compared to OSWorld, the improvements from longer history are more stable and less sensitive, reflecting the offline nature of AgentNetBench where trajectories are fixed and less prone to compounding errors.

ReVision again exhibits a different scaling behavior. Without token dropping, it matches the corresponding baselines, confirming that the training setup does not degrade performance. When redundant visual tokens are removed, ReVision consistently improves performance across all metrics. For example, with Qwen2.5-VL-7B and a 5-image history, ReVision improves average success rate from 72.5 to 73.8, with gains observed across coordinate, content, and functional metrics. Similar improvements are observed with stronger backbones, such as Qwen-3-VL-8B. These results suggest that removing redundant visual tokens does not harm any specific aspect of the task and instead

enables more effective use of the available context. Overall, the improvements are more modest compared to OSWorld, but remain consistent, indicating that redundancy-aware token filtering provides reliable gains even in less challenging, offline evaluation settings.

WebTailBench. Table 9 reports results on WebTailBench, a long-horizon benchmark designed to evaluate performance on complex multi-step tasks. Compared to OSWorld and AgentNetBench, improvements from increasing history are limited for standard baselines. While moving from one to three images provides moderate gains, further increasing to five images yields marginal or even diminishing returns despite a substantial increase in token usage. For example, OpenCUA improves from 25.8 to 29.5 at 50-step success rate when increasing history from one to three images, but slightly drops to 29.1 at five images while token cost continues to grow significantly. This pattern indicates that longer visual histories introduce redundancy that models struggle to utilize effectively.

ReVision demonstrates a markedly different behavior in this setting. When redundant visual tokens are removed, performance improves consistently as the history length increases, while maintaining significantly lower token usage. For instance, with Qwen2.5-VL-7B, ReVision improves from 28.4 to 40.8 at 50-step success rate as history increases from three to nine images, and from 35.2 to 48.9 at 100 steps. At the same time, it reduces the average number of steps required to complete tasks (e.g., from 27.9 to 23.6 at 5 images), indicating more efficient decision-making. These results show that, unlike baselines, ReVision is able to effectively leverage longer histories by removing redundant visual information. As a result, longer context becomes beneficial rather than detrimental, highlighting that redundancy-aware token filtering is critical for scaling performance in long-horizon computer-use tasks.

C Error Analysis

To evaluate the stability of our results, we run each model three times and report the standard deviation of success rate in Table 10. Across all benchmarks and history lengths, the standard deviations remain small, indicating that the observed trends are robust across runs. In particular, REVISION maintains low variance even with longer histories ($H = 7$ and $H = 9$), where the number of visual tokens and reasoning context increase substantially. This suggests that the gains from redundancy-aware token filtering are not caused by run-specific fluctuations, but reflect a consistent improvement in long-horizon decision-making.

D Generalization Across Different Models

We provide the full results for ReVision across different history window sizes (3, 5, 7, and 9 images) and model families. These results extend Table 4 in the main paper.

E Training on a Fixed Context Window Generalizes to Other Window Sizes.

We analyze whether models trained with **ReVision** under a fixed visual context window generalize to different context sizes at inference time. Although ReVision is trained with a fixed number of history images, agents in practice may operate under varying context budgets. To study this, we train models with window sizes $w \in 3, 5$ and evaluate them under both matched and mismatched inference windows. Table 12 reports results on *OSWorld* (100-step success rate) and *AgentNetBench* (average success rate), along with average tokens per step. Performance is best when the training and inference window sizes match, but the drop under mismatched settings is modest. This suggests that ReVision-trained models are robust to changes in the number of visual context images and generalize beyond their training configuration without substantial loss in performance.

F Qualitative Comparison of Token Selection Strategies

Figure 6 provides a qualitative comparison of different token selection strategies on a representative trajectory step. We visualize which patches are retained across methods by overlaying the patch grid on the screenshots, where removed regions are suppressed and retained patches are highlighted.

Naive strategies such as **Random** and **Spiral** dropping remove patches without considering semantic consistency across time. As a result, they often discard important regions (e.g., UI elements relevant to the task) while retaining redundant background content. This leads to fragmented visual context, where critical information may be missing or partially preserved.

Pixel-based similarity performs more structured filtering by removing patches with low pixel-level changes. However, it is sensitive to small visual variations (e.g., rendering noise, cursor movement), which causes it to either retain redundant regions or remove semantically important details. Consequently, although it achieves stronger token reduction, it often harms downstream reasoning.

Embedding-based methods (DINO and Qwen) provide improved consistency by comparing patch embeddings. These approaches better preserve semantically meaningful regions, but they still struggle to precisely localize task-relevant changes. In particular, they may retain large redundant areas or fail to capture fine-grained updates in the interface.

In contrast, **ReVision** explicitly models temporal redundancy between corresponding patches across consecutive images. As shown in the figure, it effectively removes unchanged regions while preserving newly updated and task-relevant content. This results in a cleaner and more focused visual representation, where the model can rely on previous images for redundant information while attending to only the necessary updates in the current step.

G Additional Efficiency Results

Figures 7 and 8 provide additional views of the efficiency-performance trade-offs across benchmarks and step budgets. Consistent with the main results, **ReVision** achieves a more favorable trade-off, reaching higher success rates with substantially fewer tokens per step.

H Region-Level Grouping and Learned Filtering

Learned filtering vs. cosine similarity. We first analyze the impact of replacing cosine similarity with a lightweight classifier for redundancy detection. Across both Qwen and DINOv2 embeddings, classifier-based filtering consistently improves performance (e.g., Qwen: 72.3 $\rightarrow$ 72.9 SR; DINOv2: 71.7 $\rightarrow$ 72.1), while slightly reducing token usage. This improvement stems from the classifier's ability to learn an adaptive decision boundary, in contrast to cosine similarity which relies on a fixed threshold (set to 0.95 in our experiments). These results suggest that even simple learned filtering can better capture context-dependent redundancy.

Effect of region-level grouping (OmniParser). We next evaluate the effect of incorporating region-level structure using OmniParser. By grouping semantically coherent UI elements, OmniParser enables more structured redundancy detection, leading to improved performance and stronger token reduction (**74.6 SR** on AgentNetBench and **35.2 SR@100** on OSWorld, with lower visual token ratios). However, this improvement comes at a significant computational cost. OmniParser introduces substantial latency ($\sim$**558–572 ms**), which is an order of magnitude higher than embedding-based approaches. In contrast, ReVision (RTS) avoids region parsing at inference time and maintains low latency ($\sim$**22–23 ms**) while achieving competitive performance (**73.8 SR**). These results highlight a key trade-off: while structured region-level grouping can improve redundancy detection, it incurs high overhead that may limit practical deployment. Our approach instead leverages such structure during training, enabling efficient inference without sacrificing performance.

I Training and Implementation Details

Training setup. We follow the OpenCUA framework (Wang et al., 2025b) and use the same training configuration for fair comparison. All models are trained on a cluster of $8\times$ NVIDIA H200 GPUs. We use standard autoregressive next-token prediction, where the loss is applied only to text tokens (reasoning and actions), while visual tokens are used as conditioning input. For each history window size k, we train a separate model using trajectory segments with up to k images, ensuring that the training distribution matches the inference setting.

Data construction. Training samples are constructed from AgentNet trajectories by applying a sliding window over interaction sequences. At each step t, we form a sample consisting of the most recent k images $\mathcal{I}_{t-k+1:t}$ together with all previous reasoning and actions. This converts trajectories into step-level supervision and increases the number of training samples.

Token filtering. During training, we apply the same ReVision token filtering pipeline used at inference time (Algorithm 1). For each pair of consecutive images, RTS compares corresponding patches and produces a binary mask to remove redundant tokens. The first image in each window is kept intact, while subsequent images retain only non-redundant patches. This ensures that the model learns to operate under partially observed visual

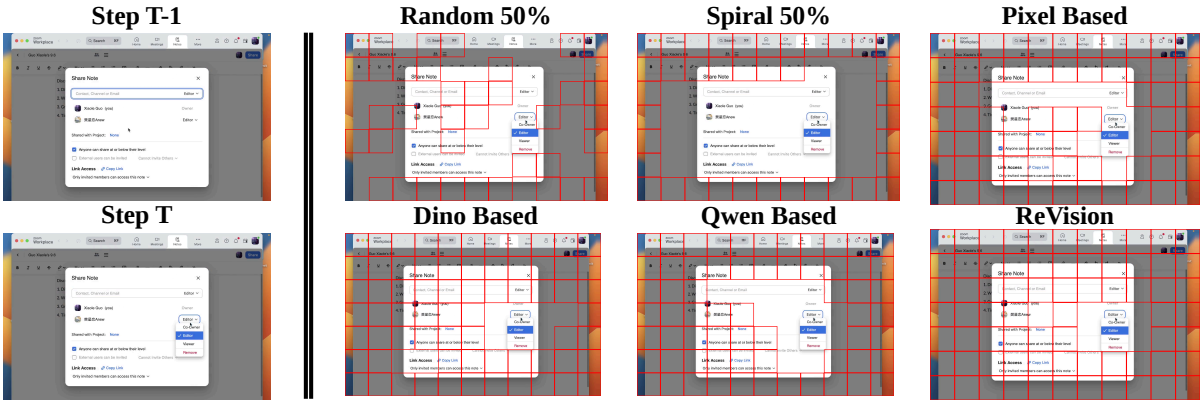


Figure 6: **Qualitative comparison of token selection strategies.** We show patch retention across different methods for two consecutive steps ($t-1$ and t). For visualization purposes, we use lower-resolution images, resulting in fewer patches and clearer overlays. Random and spiral strategies remove patches indiscriminately, often discarding important UI elements. Pixel-based similarity removes more patches but fails to preserve fine-grained semantic details. Embedding-based methods (DINO and Qwen) improve consistency but still retain redundant regions. In contrast, ReVision selectively removes temporally redundant patches while preserving task-relevant updates, leading to a more informative and compact visual representation.

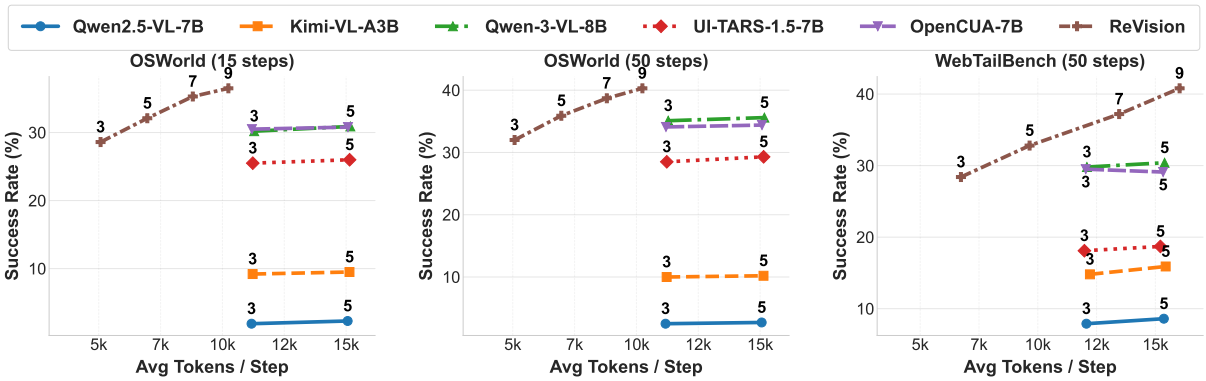


Figure 7: Success rate versus average tokens per step across OSWorld at 15 steps, 50 steps, WebTailBench at 50 steps. Detailed numerical results are provided in Tables 7, 8, and 9 in Appendix B.

inputs.

Decoding and evaluation. During inference, we use a fixed decoding temperature of T=0.0 to ensure deterministic behavior and isolate the effect of token filtering. All reported results are averaged over three runs. For WebTailBench, we adopt an LLM-as-a-judge protocol using GPT-4o to evaluate step-level correctness and compute final success rates.

Metrics. We report success rate (SR) as the primary metric across all benchmarks. For OSWorld and WebTailBench, we additionally report results under different step budgets to capture long-horizon performance. To better understand efficiency, we also analyze the relationship between success rate, token usage, and the average number of interaction steps required to complete tasks.

J Case Study

To provide a concrete illustration of how **ReVision** improves efficiency during sequential decision-making, we present a step-by-step case study on OSWorld in Table 14. The task requires removing tracking data from Amazon by navigating browser settings and disabling privacy-related options. As shown in the table, consecutive screenshots contain substantial visual overlap, particularly in static UI regions such as navigation bars, menus, and background areas. Without token filtering, all patches are retained across steps, leading to significant redundancy in the visual context.

ReVision addresses this by selectively removing visually redundant patches between consecutive frames while preserving regions that are relevant to the current action. This results in a progressively more compact representation, as seen in the "After

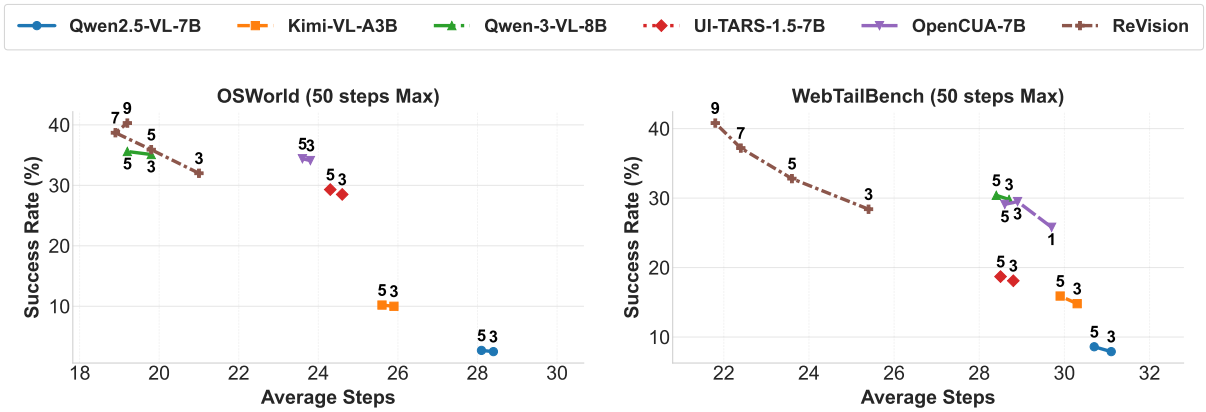


Figure 8: Success rate versus average trajectory length (number of steps) for WebTailBench and OSWorld at 50 steps. Detailed numerical results are provided in Tables 8 and 9 in Appendix B.

Token Removal" column, where large portions of unchanged interface elements are omitted. Importantly, despite this reduction, the agent continues to produce correct actions and coherent reasoning at each step. For example, the model successfully identifies and navigates to browser settings, selects the appropriate privacy options, and proceeds to disable ad personalization, all while operating on a reduced visual context.

This example highlights two key advantages of ReVision: (i) it effectively eliminates temporal redundancy without disrupting spatial alignment or task-relevant information, and (ii) it enables the model to rely on temporally distributed evidence, where previously observed content can be implicitly recalled rather than repeatedly re-encoded. Overall, the case study demonstrates that substantial token savings can be achieved in realistic multi-step interactions without sacrificing decision quality.

Model	History	Drop	15 Steps		50 Steps		100 Steps		Avg Tokens / Step
			SR	Avg Steps	SR	Avg Steps	SR	Avg Steps	
General VLMs									
Qwen2.5-VL-7B	1	✗	1.6	14.5	2.2	29.6	2.7	38.2	4,013
	3	✗	1.9	13.6	2.5	28.4	3.2	37.1	11,176
	5	✗	2.3	13.2	2.7	28.1	3.5	36.5	15,062
Qwen2.5-VL-32B	1	✗	2.5	13.8	3.3	28.5	3.9	36.7	4,067
	3	✗	2.9	12.7	3.6	27.4	4.2	35.6	11,238
	5	✗	3.0	12.5	3.8	27.0	4.4	35.2	15,087
Qwen2.5-VL-72B	1	✗	3.6	13.4	4.4	28.1	5.0	36.2	4,109
	3	✗	3.9	12.6	4.7	27.3	5.3	35.4	11,307
	5	✗	4.1	12.4	4.9	27.0	5.5	35.0	15,183
Kimi-VL-A3B	1	✗	8.5	12.6	9.7	26.5	10.3	34.1	4,094
	3	✗	9.2	11.8	10.0	25.9	10.7	33.5	11,221
	5	✗	9.5	11.6	10.2	25.6	10.9	33.1	15,136
Qwen-3-VL-8B	1	✗	29.8	12.8	34.2	20.7	33.9	27.3	4,061
	3	✗	30.2	11.9	35.1	19.8	34.8	26.2	11,284
	5	✗	30.9	11.6	35.6	19.2	35.3	25.8	15,171
Qwen-3-VL-32B	1	✗	33.2	10.4	41.1	21.4	41.0	28.4	4,133
	3	✗	39.8	9.3	42.7	19.6	42.4	27.8	11,346
	5	✗	40.3	9.1	42.8	18.9	42.5	27.3	15,229
Qwen3-VL-30B-A3B	1	✗	27.9	10.6	32.4	23.6	31.0	30.8	4,117
	3	✗	35.2	9.5	39.8	22.6	38.6	30.0	11,318
	5	✗	35.8	9.3	40.4	22.2	39.2	29.6	15,198
UI Agents									
OpenCUA	1	✗	23.8	11.2	27.4	24.6	26.0	31.8	3,994
	3	✗	30.5	10.1	34.1	23.8	32.2	31.1	11,193
	5	✗	30.8	10.0	34.4	23.6	32.5	30.9	15,075
UI-TARS-72B-DPO	1	✗	20.7	12.2	23.7	25.6	23.5	32.9	3,967
	3	✗	20.9	11.7	24.4	25.0	24.2	32.6	11,159
	5	✗	21.1	11.4	24.9	24.6	24.6	31.4	15,032
UI-TARS-1.5-7B	1	✗	24.1	11.4	27.3	25.2	27.1	29.1	4,058
	3	✗	25.5	10.9	28.5	24.6	28.5	28.6	11,227
	5	✗	26.0	10.7	29.3	24.3	28.8	28.2	15,141
ReVision									
ReVision	3	✗	30.6	9.8	34.1	22.9	32.2	29.8	11,078
	3	✓	28.6	8.9	32.0	21.0	30.5	27.9	5,074
	5	✗	30.7	9.7	34.5	22.7	32.3	29.7	15,071
	5	✓	32.1	8.3	35.9	19.8	34.0	26.4	6,963

Table 7: OSWorld results across general VLMs, UI agents, and ReVision. For ReVision, we only report settings where the training window size matches the number of history images. A checkmark indicates redundant history token dropping, while a cross indicates no dropping.

Model	History	Drop	Coord. SR	Content SR	Func. SR	Avg. SR
General VLMs						
Qwen2.5-VL-7B	1	✗	25.1	18.3	11.9	23.4
Qwen2.5-VL-7B	3	✗	26.4	19.5	12.7	25.0
Qwen2.5-VL-7B	5	✗	27.0	20.1	13.3	25.8
Qwen2.5-VL-32B	1	✗	38.5	28.2	18.4	35.6
Qwen2.5-VL-32B	3	✗	40.2	29.8	19.5	37.9
Qwen2.5-VL-32B	5	✗	41.0	30.5	20.2	38.7
Qwen2.5-VL-72B	1	✗	42.1	30.5	20.2	38.7
Qwen2.5-VL-72B	3	✗	43.6	31.8	21.0	40.8
Qwen2.5-VL-72B	5	✗	44.4	32.4	21.7	41.5
Kimi-VL-A3B	1	✗	51.6	37.8	25.9	47.2
Kimi-VL-A3B	3	✗	54.0	39.5	27.2	49.8
Kimi-VL-A3B	5	✗	55.2	40.4	28.1	50.9
Qwen-3-VL-8B	1	✗	73.4	54.9	40.2	68.7
Qwen-3-VL-8B	3	✗	74.6	58.8	43.1	71.9
Qwen-3-VL-8B	5	✗	75.2	59.3	43.4	72.2
Qwen-3-VL-72B	1	✗	79.2	64.5	48.1	76.9
Qwen-3-VL-72B	3	✗	82.5	67.4	50.8	79.3
Qwen-3-VL-72B	5	✗	83.6	68.3	51.9	80.2
Qwen3-VL-30B-A3B	1	✗	76.5	61.3	44.6	74.1
Qwen3-VL-30B-A3B	3	✗	80.4	65.1	48.2	77.0
Qwen3-VL-30B-A3B	5	✗	81.3	66.0	49.1	77.8
UI Agents						
OpenCUA	1	✗	72.0	55.6	39.8	68.4
OpenCUA	3	✗	75.8	59.4	42.5	72.1
OpenCUA	5	✗	76.1	59.7	42.8	72.4
UI-TARS-72B-DPO	1	✗	66.2	48.1	34.7	62.3
UI-TARS-72B-DPO	3	✗	67.1	49.0	35.5	63.2
UI-TARS-72B-DPO	5	✗	67.8	49.6	36.0	63.8
UI-TARS-1.5-7B	1	✗	70.4	52.3	38.8	66.9
UI-TARS-1.5-7B	3	✗	71.8	53.8	40.2	68.4
UI-TARS-1.5-7B	5	✗	72.3	54.5	40.6	68.9
ReVision						
ReVision	3	✗	75.9	59.3	42.4	72.0
ReVision	3	✓	74.4	58.1	41.7	70.7
ReVision	5	✗	76.2	59.8	42.9	72.5
ReVision	5	✓	77.0	61.0	44.2	73.8

Table 8: AgentNetBench results across general VLMs, UI agents, and ReVision. For ReVision, we only report settings where the training window size matches the number of history images.

Model	History	Drop	SR@50	Avg. Steps@50	SR@100	Avg. Steps@100	Avg Tokens / Step
General VLMs							
Qwen2.5-VL-7B	1	✗	6.5	31.8	7.8	37.2	4,213
Qwen2.5-VL-7B	3	✗	7.9	31.1	9.2	36.5	12,067
Qwen2.5-VL-7B	5	✗	8.6	30.7	10.1	36.0	15,361
Qwen2.5-VL-32B	1	✗	6.2	32.1	7.1	37.4	4,287
Qwen2.5-VL-32B	3	✗	6.8	31.4	7.9	36.8	12,183
Qwen2.5-VL-32B	5	✗	7.5	31.0	8.6	36.1	15,472
Qwen2.5-VL-72B	1	✗	7.5	31.7	8.6	36.9	4,349
Qwen2.5-VL-72B	3	✗	8.2	31.0	9.4	36.2	12,244
Qwen2.5-VL-72B	5	✗	9.0	30.6	10.1	35.7	15,561
Kimi-VL-A3B	1	✗	13.2	31.0	15.0	36.1	4,318
Kimi-VL-A3B	3	✗	14.8	30.3	16.9	35.4	12,217
Kimi-VL-A3B	5	✗	15.9	29.9	18.1	35.0	15,438
Qwen-3-VL-8B	1	✗	26.0	26.2	32.8	35.3	4,266
Qwen-3-VL-8B	3	✗	29.8	28.7	36.6	34.5	12,089
Qwen-3-VL-8B	5	✗	30.4	28.4	37.1	34.0	15,392
Qwen-3-VL-32B	1	✗	34.8	27.5	41.9	34.3	4,331
Qwen-3-VL-32B	3	✗	39.6	29.6	46.8	33.4	12,214
Qwen-3-VL-32B	5	✗	40.2	29.2	47.3	32.9	15,487
Qwen3-VL-30B-A3B	1	✗	32.5	27.1	39.2	34.8	4,309
Qwen3-VL-30B-A3B	3	✗	37.4	29.1	44.3	34.0	12,173
Qwen3-VL-30B-A3B	5	✗	38.0	28.8	44.9	33.5	15,456
UI Agents							
OpenCUA	1	✗	25.8	29.7	32.4	26.4	4,237
OpenCUA	3	✗	29.5	28.9	36.2	30.2	12,041
OpenCUA	5	✗	29.1	28.6	35.8	29.8	15,329
UI-TARS-72B-DPO	1	✗	15.3	26.8	17.4	34.6	4,248
UI-TARS-72B-DPO	3	✗	16.9	28.4	18.2	33.7	12,107
UI-TARS-72B-DPO	5	✗	17.4	28.1	19.5	33.2	15,376
UI-TARS-1.5-7B-DPO	1	✗	16.5	26.6	20.4	35.0	4,169
UI-TARS-1.5-7B-DPO	3	✗	18.1	28.8	22.2	34.0	11,982
UI-TARS-1.5-7B-DPO	5	✗	18.7	28.5	26.8	33.5	15,214
ReVision							
ReVision	3	✗	29.6	28.1	36.3	30.3	11,907
ReVision	3	✓	28.4	25.4	35.2	29.2	6,731
ReVision	5	✗	29.3	27.9	36.0	30.0	15,362
ReVision	5	✓	32.8	23.6	40.2	27.4	9,651

Table 9: WebTailBench results on long-horizon tasks. We report success rates at 50 and 100 steps, average trajectory length, and average tokens per step. For ReVision, we report configurations with matched training window and history size, together with the corresponding no-drop controls when available.

Model	Hist.	OSWorld	AgentNetBench	WebTailBench
Qwen2.5-VL-7B	$H = 3$	0.8	0.7	0.9
	$H = 5$	0.9	0.8	1.0
Kimi-VL-A3B	$H = 3$	1.0	0.9	1.1
	$H = 5$	1.1	0.8	1.2
Qwen3-VL-8B	$H = 3$	0.7	0.6	0.8
	$H = 5$	0.8	0.7	0.9
UI-TARS-1.5-7B	$H = 3$	0.9	0.8	1.0
	$H = 5$	1.0	0.9	1.1
OpenCUA-7B	$H = 3$	0.8	0.7	0.9
	$H = 5$	0.9	0.8	1.0
REVISION	$H = 3$	0.7	0.6	0.8
	$H = 5$	0.8	0.6	0.9
	$H = 7$	0.9	0.7	1.0
	$H = 9$	1.0	0.8	1.1

Table 10: Standard deviation of success rate over three evaluation runs. We report SR@100 for OSWorld and WebTailBench, and average SR for AgentNetBench. The small standard deviations indicate that the observed trends are stable across runs.

ReVision Base Model	Hist.	WebTailBench		OSWorld		AgentNetBench	
		SR@100	Avg. Tok/Step	SR@100	Avg. Tok/Step	Avg SR	Avg. Tok/Step
Qwen2.5-VL-7B	3	35.2	6,731	30.5	5,074	70.7	6,235
	5	40.2	9,651	34.0	6,963	73.8	8,975
	7	45.6	13,528	36.7	8,802	75.9	11,284
	9	48.9	15,283	38.4	10,241	77.1	13,012
Qwen3-VL-8B	3	42.1	7,209	34.1	5,396	73.5	6,654
	5	46.6	10,941	36.7	7,258	76.0	9,218
	7	49.8	13,462	40.4	8,994	78.6	11,759
	9	52.4	16,031	41.6	10,723	80.2	13,921

Table 11: Generalization results of ReVision across model families and history sizes.

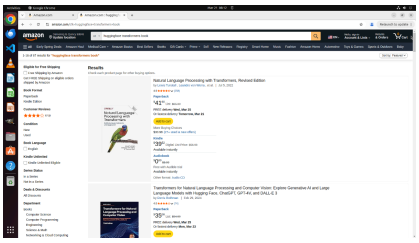
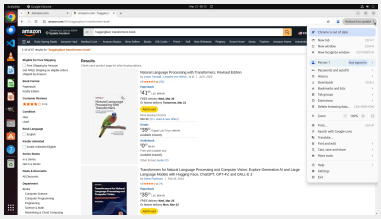
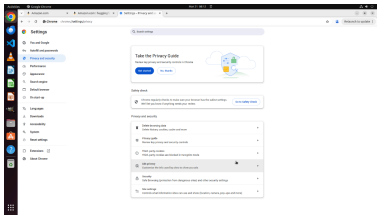
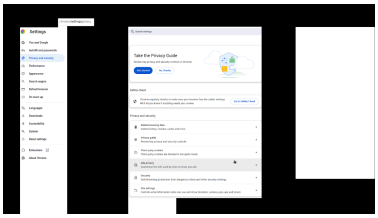
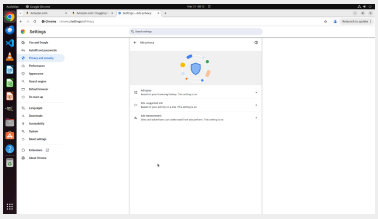
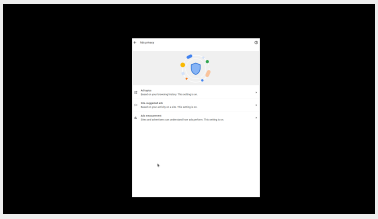
Train Window Size	Inference Window Size	OSWorld		AgentNetBench	
		SR@100	Tok/Step	Avg SR	Tok/Step
3	3	30.5	5,074	70.7	6,235
3	5	29.1	6,755	69.7	8,437
5	3	29.4	5,394	69.9	6,959
5	5	34.0	6,963	73.8	8,975

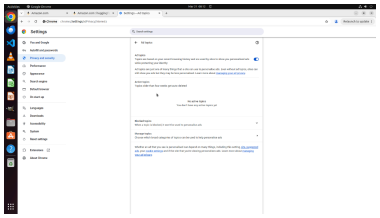
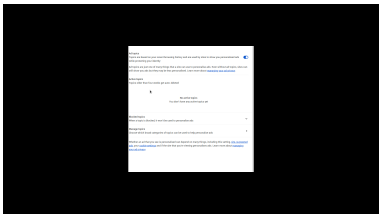
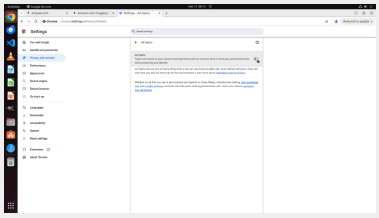
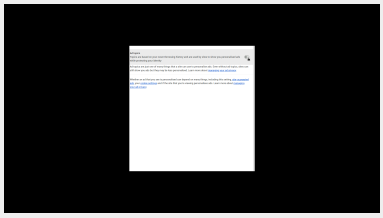
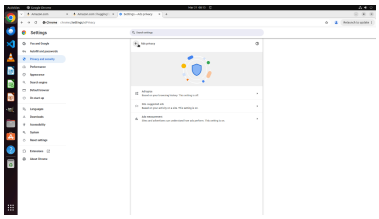
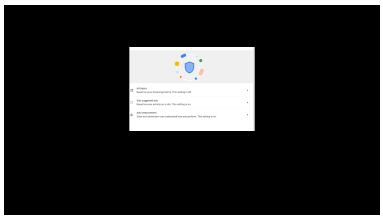
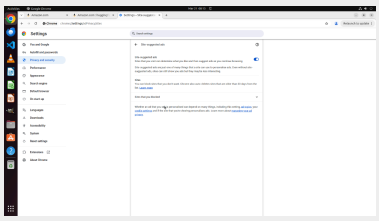
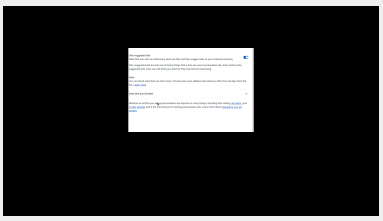
Table 12: Cross-window generalization of ReVision on **Qwen2.5-VL-7B**. Models perform best when training and inference window sizes match, but maintain competitive performance under mismatched settings, indicating robustness to varying visual context sizes.

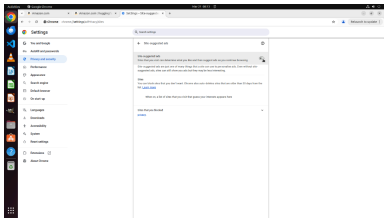
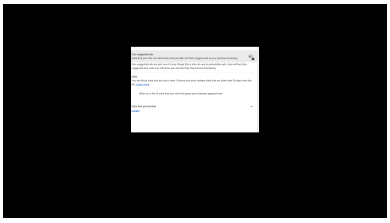
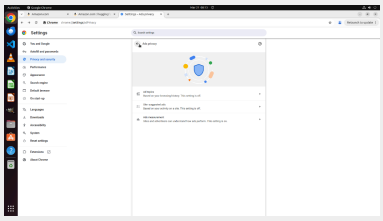
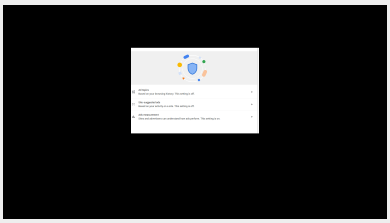
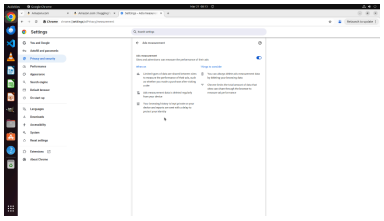
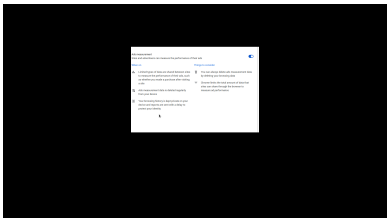
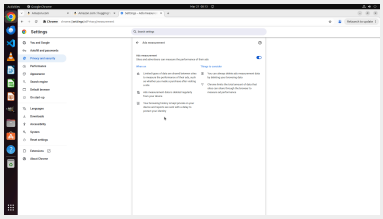
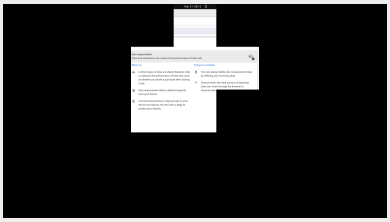
Strategy	AgentNetBench				OSWorld			
	SR	**Avg Tok/Step**	**Vis Tok%**	**Latency (ms)**	**SR@100**	**Avg Tok/Step**	**Vis Tok%**	**Latency (ms)**
No Drop	72.5	15076	92.9	0	32.3	15071	92.9	0
Qwen2.5-VL-7B (CosSim)	72.3	9424	85.6	6	32.1	7624	84.1	8
Qwen2.5-VL-7B (Classifier)	72.9	9301	85.2	10	32.6	7480	83.8	13
DINOv2-base (CosSim)	71.7	9682	86.2	26	31.4	7915	84.9	31
DINOv2-base (Classifier)	72.1	9520	85.9	29	31.9	7768	84.5	36
RTS + OmniParser	74.6	8420	79.8	558	35.2	6485	78.9	572
RTS (Ours)	73.8	8975	83.4	23	34.0	6963	82.9	22

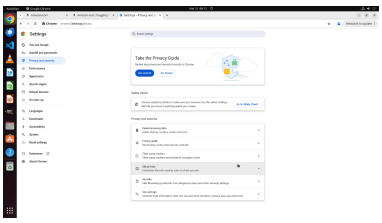
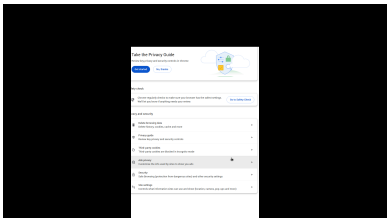
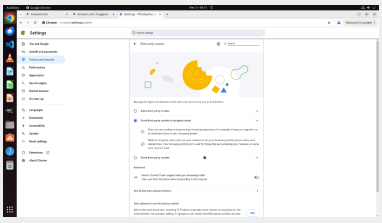
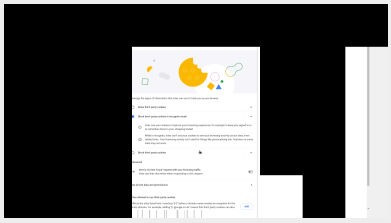
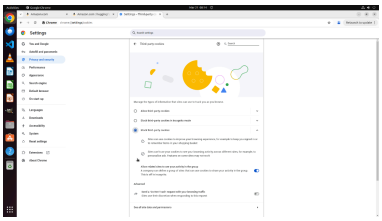
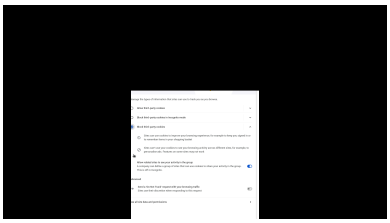
Table 13: **Ablation on learned filtering and region-level grouping.** We compare cosine similarity and classifier-based filtering across embedding backbones, as well as region-based grouping with OmniParser. Classifier-based methods provide consistent improvements over cosine similarity by learning adaptive thresholds. Incorporating OmniParser further improves performance and reduces visual redundancy, but introduces substantial latency overhead. ReVision (RTS) achieves a strong balance, maintaining competitive performance with significantly lower inference cost.

Table 14: Case study of ReVision on OSWorld.

Step	Before Token Removal	After Token Removal
Task: Can you help me clean up my computer by getting rid of all the tracking things that Amazon might have saved? I want to make sure my browsing is private and those sites don't remember me.		
1		
Action: pyautogui.click(1893, 139) **Reasoning:** The browser is open on Amazon search results. The next step toward removing Amazon tracking data is to open Chrome's main menu so I can get to browsing data/site settings and clear Amazon cookies and stored data.		
2		
Action: pyautogui.click(1702, 524) **Reasoning:** The Chrome menu is open, and there is a direct "Delete browsing data..." option visible, which is the quickest path to clearing Amazon cookies, cache, and other tracking data so the site won't remember the user.		
3		
Action: pyautogui.click(1702, 524) **Reasoning:** To reduce tracking and site-based remembering, the next useful step is to open Chrome's ad privacy controls and disable ad-related site personalization. To start, we start with changing the setting for add privacy. We can use the "Add Privacy" button to go the related option.		
4		
Action: pyautogui.click(784, 456) **Reasoning:** The Ads privacy page shows ad-related personalization features are still on. The next step is to open "Ad topics" first and turn it off to reduce tracking and personalized ads.		

Step	Before Token Removal	After Token Removal
5		
Action: pyautogui.click(1300, 284) **Reasoning:** The “Ad topics” setting is currently on. To improve privacy and reduce tracking-based personalization, the next step is to toggle it off.		
6		
Action: pyautogui.click(683, 197) **Reasoning:** Ad topics is now turned off. The next step is to go back to the Ads privacy page and disable the other ad-tracking related settings, starting with Site-suggested ads.		
7		
Action: pyautogui.click(824, 510) **Reasoning:** Ad topics is now off, while site-suggested ads and Ads measurement are still on. The next step is to open Site-suggested ads and disable it.		
8		
Action: pyautogui.click(1300, 274) **Reasoning:** Site-suggested ads is currently on. The next step is to toggle it off to reduce tracking and ad personalization.		

Step	Before Token Removal	After Token Removal
9		
Action: pyautogui.click(682, 197) **Reasoning:** Site-suggested ads is now off. The next step is to go back to Ads privacy and disable Ads measurement as well.		
10		
Action: pyautogui.click(815, 574) **Reasoning:** Ad topics and Site-suggested ads are now off, while Ads measurement is still on. The next step is to open Ads measurement and turn it off.		
11		
Action: pyautogui.click(1301, 273) **Reasoning:** Ads measurement is still enabled. The next step is to toggle it off to further reduce tracking.		
12		
Action: pyautogui.hotkey('ctrl', 'shift', 'delete') **Reasoning:** All Chrome Ads privacy tracking settings shown are now off. To make Amazon not remember the user, the next step is to clear Amazon cookies/site data from the open Amazon tab.		

Step	Before Token Removal	After Token Removal
13		
	Action: pyautogui.click(1018, 764) **Reasoning:** Browsing data has been cleared, so the next privacy step is to tighten cookie tracking by opening Third-party cookies and blocking them more broadly.	
14		
	Action: pyautogui.click(1018, 764) **Reasoning:** The third-party cookies page is open and currently set to block only in Incognito. To improve privacy and reduce tracking, the next step is to select “Block third-party cookies.”	
15		
	Action: pyautogui.click(1295, 844) **Reasoning:** Third-party cookies are now blocked. I confirm the task has ended now.	