DON'T LOOK TWICE: LEVERAGING VISUAL EXPERIENCE FOR TOKEN-EFFICIENT GUI AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

GUI agents powered by multimodal large language models (MLLMs) repeatedly process high-resolution screenshots throughout multi-step interactions, introducing substantial visual computation. Existing efficiency methods primarily reduce redundancy within individual screenshots or interaction trajectories, overlooking reusable visual experience accumulated across tasks. We introduce PAVE, a framework that leverages historical interactions for efficient GUI inference. PAVE retrieves relevant experience from previously completed tasks and uses it as a prior for a lightweight controller that adaptively selects visual tokens from the current screen, while keeping the underlying action model unchanged. Experiments on offline GUI benchmarks and online in-the-wild environments show that PAVE reduces visual-token computation by up to 17.4% and 8.3%, while retaining the original task performance. We open source our code and experiment to facilitate further research and application.

1 INTRODUCTION

Multimodal large language models (MLLMs) are increasingly deployed as autonomous agents that perceive and interact with graphical user interfaces (GUIs) to accomplish everyday tasks (Hong et al., 2024; Wang et al., 2024; Cheng et al., 2024). Unlike conventional vision-language tasks that typically require processing only a few images, GUI agents operate over long interaction trajectories, repeatedly processing high-resolution screenshots at each step (Rawles et al., 2025; Xie et al., 2024; Huang et al., 2025; Wang et al., 2026). Each screenshot can introduce a large number of visual tokens, making visual processing a substantial source of computational overhead. As task horizons grow, this cost accumulates across interaction steps, making GUI agent inference increasingly expensive.

To mitigate the growing inference overhead, recent studies have explored efficient GUI agents by reducing redundant computation across both individual screenshots and interaction trajectories. Some methods develop efficient visual architectures or representations for high-resolution GUI understanding (Hong et al., 2024; Lin et al., 2025), while others exploit the sequential nature of GUI interaction to prune redundant observations, historical frames, or KV-cache entries (Abaskohi et al., 2026; Huang et al., 2025; Wang et al., 2026; Zhou et al., 2026). In parallel, general visual token reduction provides a complementary approach by selectively pruning or merging redundant tokens before or during MLLM inference, such as FastV (Chen et al., 2024), DivPrune (Alvar et al., 2025), CDPruner (Zhang et al., 2025), and PruMerge+ (Shang et al., 2025).

Despite their effectiveness on general vision-language benchmarks, existing visual token pruning methods largely treat each observation in isolation, overlooking a unique opportunity in GUI agents: visual experience accumulated through historical interactions. As GUI agents continuously interact with interfaces and complete tasks, they repeatedly encounter semantically related visual elements, layouts, and interaction patterns across episodes. For example, after learning to send a message to one contact, an agent later messaging another contact will encounter largely the same interface, making much of the visual information already familiar. However, existing methods typically determine token importance solely from the current observation, without leveraging such accumulated experience, and thus repeatedly devote computation to visual information that may already be familiar from previous interactions. This motivates a different paradigm for efficient GUI agents: rather than deciding which visual tokens are important from each screen independently, an agent can leverage its historical

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.

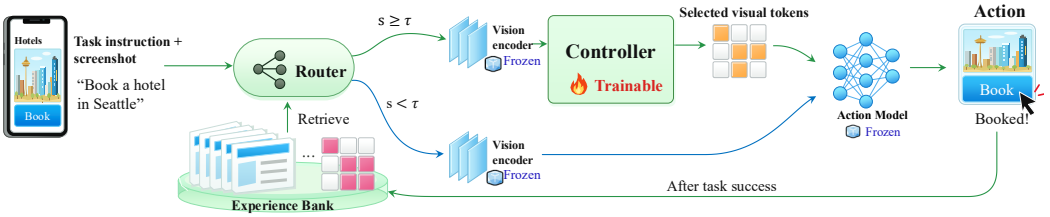


Figure 1: Overview of PAVE. Given the current task and screen, a router first retrieves relevant visual experience from previously completed interactions. Conditioned on the retrieved experience, a lightweight trained controller then identifies and retains task-relevant visual tokens from the current screen, reducing redundant visual computation while preserving the action model's performance.

interactions to assess what visual information is already familiar or reusable for the current task, and selectively allocate computation to tokens that require further processing.

Inspired by how humans leverage prior experience to perform extended, repetitive tasks with reduced cognitive effort, we propose **PAVE**, a **P**ersistent **A**cross-episode **V**isual **E**xperience framework for efficient GUI agents. As illustrated in Figure 1, rather than processing every screen from scratch, PAVE exploits information accumulated throughout the interaction trajectory to allocate visual computation selectively. Specifically, PAVE first measures the visual-semantic relationship between the current task and historical interactions to identify reusable information. Conditioned on this interaction history, a lightweight trained controller then determines which visual tokens require further processing and which can be safely pruned or reused. The resulting compact visual representation is finally passed to the MLLM to predict the next action.

We first evaluate PAVE on two widely used GUI benchmarks, GUIOdyssey and AndroidControl. PAVE achieves strong performance while substantially reducing visual-token computation, reaching 43.35% and 66.86% accuracy on GUIOdyssey and AndroidControl, respectively, while reducing visual-token usage by 14.93% and 17.40%. Beyond offline benchmarks, we further evaluate the real-world effectiveness of PAVE through in-the-wild experiments on AndroidWorld. In this more challenging setting, PAVE achieves a 48.2% task success rate while further reducing visual-token usage by 8.29%. These results demonstrate that leveraging cross-episode visual experience can effectively reduce redundant visual processing without substantially compromising task performance.

- We introduce a new perspective on efficient GUI agents by exploiting visual experience accumulated across interaction episodes. That is, the interaction history provides valuable cues for identifying redundant visual information and reducing repeated computation over long-horizon tasks.
- We propose PAVE, a history-aware visual token pruning framework for efficient GUI agents. PAVE captures visual-semantic relationships between historical and current task and employs a lightweight learned controller to adaptively determine which visual tokens require further processing.
- We conduct extensive experiments on established GUI-agent benchmarks together with in-the-wild evaluations. The results demonstrate that PAVE substantially reduces visual token computation while maintaining competitive task performance, highlighting its effectiveness and practical utility for real-world GUI agents.

2 Related Work

Efficient GUI Agents. GUI agents continuously process high-resolution screenshots throughout multi-step interactions, resulting in substantial visual computation and memory overhead (Huang et al., 2025; Wang et al., 2026; Zhou et al., 2026). Existing efforts improve inference efficiency at different levels of GUI perception and interaction. One line of work develops more efficient visual representations or architectures. For example, CogAgent (Hong et al., 2024) combines a low-resolution visual backbone with a high-resolution cross-attention branch to retain fine-grained information required for GUI grounding. Other methods reduce the redundancy contained within in-

dividual GUI screens. AQuaUI (Li et al., 2026b), for instance, adaptively partitions screenshots using quadtrees, while general visual token reduction techniques prune or consolidate tokens according to attention, feature diversity, instruction relevance, or visual redundancy (Chen et al., 2024; Alvar et al., 2025; Zhang et al., 2025; Shang et al., 2025). Beyond optimizing individual screens, recent work exploits the sequential nature of GUI interaction to reduce redundancy within an interaction trajectory. ReVision (Abaskohi et al., 2026) learns to reduce temporally redundant visual information across interaction steps, while other approaches use spatial and temporal cues to prune high-resolution screenshots or historical frames (Xu et al., 2026b; Huang et al., 2025; Wang et al., 2026; Zhou et al., 2026; Li et al., 2026a).

However, existing approaches primarily optimize visual computation *within the current screens or interaction episode*; once an episode ends, the visual experience acquired during that interaction is typically discarded rather than reused to facilitate subsequent tasks. MementoGUI (Zeng et al., 2026) instead retrieves cross-episode memory as textual and visual context for action prediction. In contrast, PAVE investigates *persistent cross-episode visual experience*, where historical interactions are retained and related to the current task to guide adaptive visual token processing.

Visual Token Pruning. Visual token pruning improves MLLM inference efficiency by reducing redundant visual tokens, with existing approaches broadly categorized into pruning-based and merging-based methods. Pruning-based methods directly identify and remove less informative visual tokens. FastV (Chen et al., 2024) estimates token importance using attention statistics within the language model, while DivPrune (Alvar et al., 2025) promotes feature diversity to eliminate redundant visual representations. CDPruner (Zhang et al., 2025) further considers both feature diversity and instruction relevance when selecting tokens. Merging-based methods, instead of simply discarding redundant tokens, aggregate their information into a smaller set of compact representations. VisionZip (Yang et al., 2025) identifies informative visual tokens and merges contextual tokens to construct a compact visual representation, while PruMerge+ (Shang et al., 2025) combines token pruning and merging to adaptively reduce the visual token count. Despite their effectiveness, existing methods determine token redundancy primarily from the current visual input, overlooking reusable experience from prior interactions. In contrast, PAVE leverages cross-episode visual experience to adaptively allocate visual computation based on accumulated interaction history.

3 METHOD

3.1 PROBLEM FORMULATION

In multi-step GUI interaction, an agent repeatedly observes the interface and predicts an action conditioned on the current screen and task instruction. At step t, a frozen vision encoder maps the current screenshot into N_t post-encoder visual tokens,

$$V_t = \{v_{t,i}\}_{i=1}^{N_t}, \qquad v_{t,i} \in \mathbb{R}^{d_v}. \tag{1}$$

where d_v denotes the visual feature dimension. Each token is associated with its normalized spatial center, denoted by $X_t = \{x_{t,i}\}_{i=1}^{N_t}$. Let g_t denote the original task prompt provided to the action model. Under standard inference, the action model consumes all N_t visual tokens at every step, repeatedly incurring the cost of processing the full screen.

Our goal is to reduce this visual computation by leveraging experience accumulated from previously completed episodes. Let $\mathcal{B}_t$ denote the experience bank available before the current episode, and let $q_t = z_t^q$ denote a frozen task embedding used to relate the current task to this historical experience. Importantly, q_t serves only as a retrieval and selection signal and is not directly provided to the action model. Conditioned on the current observation, task, and available experience, PAVE predicts a subset of token indices

$$K_t \subseteq \{1, \ldots, N_t\}, \tag{2}$$

and passes only the selected visual tokens, together with their original spatial positions, to the unchanged action model:

$$\hat{a}_t = A\left(g_t, V_t[K_t], X_t[K_t]\right). \tag{3}$$

When $K_t = \{1, \ldots, N_t\}$,, this formulation recovers the original backbone, which processes the complete visual input. The objective is therefore to minimize the fraction of visual tokens delivered to the action model, $|K_t|/N_t$, while preserving its ability to execute the task successfully.

3.2 PERSISTENT VISUAL EXPERIENCE

Standard GUI agents treat each task largely as an independent episode, discarding the visual information acquired once the interaction terminates. However, GUI tasks often revisit related interfaces, layouts, and functional elements, making previously encountered screens potentially useful for reducing visual processing in subsequent tasks. We therefore introduce *Persistent Visual Experience (PVE)*, which maintains a lightweight experience bank across episodes and makes successful historical interactions available to future tasks.

In details, we maintain an experience bank $\mathcal{B}$ across task episodes. The bank is initialized as empty, $\mathcal{B}_0 = \varnothing$, so the first task is processed without prior experience. After completing an episode e, we update the bank only when the task is successfully completed. Restricting admission to successful episodes provides a simple quality-control mechanism for persistent experience. An unsuccessful trajectory may contain screens reached through erroneous actions or states irrelevant to the intended task, making its visual history an unreliable reference for future computation. We therefore use success as the admission criterion. As more tasks are completed, $\mathcal{B}$ progressively captures visual experience acquired from prior interactions.

3.3 CROSS-EPISODE EXPERIENCE RETRIEVAL

As the experience bank grows, only a small fraction of historical interactions are likely to be relevant to the current task. Retrieving all accumulated experience would introduce unnecessary computation and, more importantly, unrelated historical context. We therefore retrieve only experiences that are sufficiently similar to the current interaction.

Task instructions alone may be insufficient to identify reusable experience, as similar tasks can occur on visually different interfaces, while different instructions may refer to the same GUI. We therefore characterize an interaction from three complementary signals: the *task instruction*, the *current screenshot*, and the *on-screen text*. Specifically, we use a frozen CLIP ViT-B/32 (Radford et al., 2021) to embed the task instruction and screenshot, and apply OCR to extract textual content visible on the interface. The same representations are maintained for each experience in the bank, allowing the current interaction to be compared with previously completed tasks in terms of both task semantics and interface content.

For each historical experience $e \in \mathcal{B}$, we compute a multimodal similarity score $s(t, e)$ between its stored metadata and the current interaction. We consider an experience reusable only when

$$s(t, e) \geq \tau, \tag{4}$$

where τ is a similarity threshold. When no historical experience reaches τ, we treat the current interaction as having no reusable experience and process the current screenshot without historical conditioning. Otherwise, the retrieved experiences are passed to the subsequent experience-conditioned controller to guide visual token pruning.

3.4 EXPERIENCE-CONDITIONED CONTROLLER

The retrieved experiences indicate that the current task resembles interactions encountered previously, but they do not directly determine which tokens should be retained from the current screen. We therefore introduce an experience-conditioned controller that uses retrieved experience as a prior while making token-level decisions based on the current screenshot and task, as shown in Figure 2.

Experience-derived Spatial Prior. Among the retrieved experiences, some contain sparse token masks that record which visual regions were retained during their previous interactions. We transfer these masks to the current visual-token lattice through bilinear interpolation, yielding $\tilde{m}_j(i)$ for retrieved experience j at current token position i. Since more similar experiences should provide stronger evidence, we normalize their retrieval similarities s_j into weights $\alpha_j = \mathrm{softmax}(s_j/T)$, where T controls the concentration of the weighting.

We summarize the retrieved masks at each position into a three-channel spatial prior,

$$P_t(i) = \left[\sum_{j \in S_t} \alpha_j \tilde{m}_j(i), ; \mathrm{Var}_\alpha \left(\tilde{m}_j(i) \right), |S_t|/K \right], \tag{5}$$

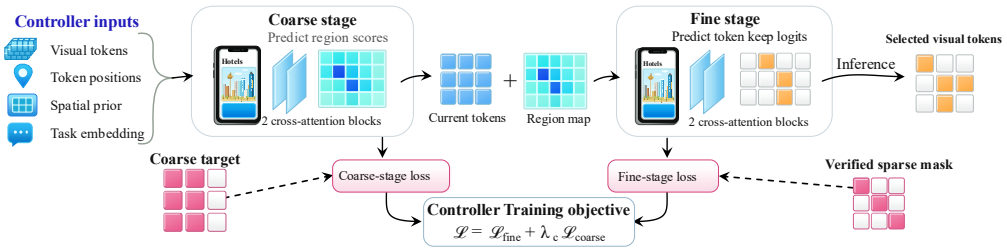


Figure 2: Controller training and inference, where the coarse stage predicts region-level relevance scores and the fine stage predicts token-level logits.

where S_t denotes the subset of retrieved experiences that contain sparse masks and K is the maximum number of retrieved experiences. The three channels respectively capture *support*, indicating how strongly prior experience favors a location; *disagreement*, indicating how consistently the retrieved experiences agree on that location; and *coverage*, indicating how much spatial evidence is available.

Controller. Given the experience-derived spatial prior, we next determine which visual tokens from the current screen should be retained. We employ a lightweight two-stage Transformer (Vaswani et al., 2017), with two cross-attention blocks per stage, a hidden dimension of 256, and four attention heads.

$$h_{t,i} = W_v v_{t,i} + \phi_x(x_{t,i}) + \phi_p(P_t(i)), \qquad h_t^q = \phi_q(q_t). \tag{6}$$

where q_t is the frozen CLIP task embedding and ϕ_x, ϕ_p, and ϕ_q project their respective inputs into the controller's hidden space.

To avoid making aggressive token-level decisions directly, the controller follows a coarse-to-fine design. The coarse stage first divides the visual lattice into 4×4 spatial anchors and predicts high-recall context scores ℓ^c, identifying regions that may contain useful information. These soft predictions then condition the fine stage, which produces a keep logit ℓ_i^f for every visual token. The final decision therefore remains grounded in the current screen and task, while retrieved experience provides additional spatial evidence.

Controller Supervision. Existing GUI datasets provide action annotations but do not specify which visual tokens are necessary for correct action prediction. We therefore construct supervision through controlled interventions on the frozen action model. Specifically, we consider states where inference with the full visual input correctly grounds the target element, and rerun the model using each candidate mask from a predefined family $\mathcal{C}t$. We select the sparsest candidate that preserves correct grounding:

$$m_t^\star = \arg\min_{m \in \mathcal{C}_t} |m|_0 \quad \text{s.t.} \quad A(g_t, V_t[m], X_t[m]) \in B_t^{\text{GT}}, \tag{7}$$

where B_t^{GT} denotes the ground-truth bounding box and $|m|_0$ measures the number of retained tokens. The resulting $m_t^\star$ serves as a sparse sufficient target for training the controller, indicating a minimal token subset within $\mathcal{C}_t$ that preserves the frozen model's correct prediction. We supervise the controller only with such verified sparse masks; if correct grounding requires the candidate containing all visual tokens, we treat the state as unresolved rather than assigning an all-one target, since it provides no reliable evidence for token-level sparsity. Further details on the candidate-mask family and training-data construction are provided in Appendix A.

Training Objective. The supervision masks are typically sparse, with only a small fraction of visual tokens labeled for retention. A standard token-wise binary cross-entropy (BCE) loss would therefore be dominated by negative tokens and could bias the controller toward excessive pruning. To address this imbalance, we combine class-balanced BCE with a keep-rate matching term that encourages the predicted retention ratio to follow that of the supervision mask:

$$\mathcal{L}_{\text{mask}}(\ell, m; \lambda_b) = \text{BCE}_{\text{bal}}(\ell, m) + \lambda_b \left| \text{mean}\, \sigma(\ell) - \text{mean}(m) \right|. \tag{8}$$

where the first term balances positive and negative token supervision, while the second penalizes deviations between the predicted and target keep rates.

We apply this objective at both stages of the coarse-to-fine controller. The fine-stage target is the verified sparse mask $m_t^\star$. To construct the coarse-stage target $c_t^\star$, we mark a 4×4 anchor cell as positive whenever it contains at least one positive token in $m_t^\star$, providing a higher-recall spatial target for the coarse stage. The overall training objective is

$$\mathcal{L} = \mathcal{L}_{\text{fine}} + \lambda_c \mathcal{L}_{\text{coarse}} \tag{9}$$

where $\mathcal{L}_{\text{fine}}$ supervises the final token-level prediction and $\mathcal{L}_{\text{coarse}}$ provides auxiliary supervision for coarse context localization. Throughout training, all components except the controller remain frozen, and model checkpoints are selected according to validation loss. Optimization details and loss coefficients are provided in Appendix A.4.

4 Experimental Setup

Backbone and Implementation Details. We adopt GUI-Owl-1.5-8B-Instruct (Xu et al., 2026a) as the backbone action model, owing to its strong GUI understanding and interaction capabilities across multiple platforms. Throughout our experiments, the backbone remains frozen, and only the lightweight experience-conditioned controller is trained. We train the controller for 50 epochs with a batch size of 2 using a single NVIDIA L40S GPU. For retrieval, we rank eligible records and retain at most the top three; the highest similarity is compared with τ to determine whether to invoke the controller. Restricting retrieval to a small set keeps the historical context compact and reduces interference from weakly related experiences. Additional training hyperparameters and implementation details are provided in Appendix A.

Offline Benchmark Evaluation. We first conduct offline experiments on two mobile GUI benchmarks, GUIOdyssey (Lu et al., 2025) and AndroidControl (Li et al., 2024), to evaluate the effectiveness and efficiency of PAVE under controlled settings. These benchmarks contain diverse mobile interaction trajectories and allow us to evaluate whether historical visual experience can reduce the number of visual tokens processed by the backbone while preserving its original action-prediction capability. For each benchmark, we construct a fixed experience bank from training-only records and evaluate on held-out test states. We report both action accuracy, measured by whether the agent correctly predicts the target, and visual-token reduction to evaluate the trade-off between task performance and inference efficiency. More details are provided in Appendix B.

Online In-the-Wild Evaluation. Beyond offline benchmarks, we conduct in-the-wild evaluation in AndroidWorld (Rawles et al., 2025), where the agent observes and interacts with a live Android environment. In contrast to offline evaluation on prerecorded interaction data, AndroidWorld requires the agent to execute multi-step actions and respond to dynamically changing interface states, providing a more realistic assessment of whether cross-episode visual experience remains effective during end-to-end deployment. We evaluate PAVE on 103 AndroidWorld tasks following the standard evaluation protocol and evaluate its task success rate and visual-token reduction. More details are provided in Appendix B.

5 Results

5.1 Offline Benchmark Evaluation

Effect of experience retrieval threshold. The retrieval threshold τ plays an important role in PAVE by controlling when historical experience is considered sufficiently relevant to guide visual-token reduction. We therefore first investigate the effect of different retrieval thresholds on GUIOdyssey and AndroidControl, as shown in Figure 3. Overall, lowering τ allows more historical experiences to participate in visual-token reduction, leading to greater token savings but with a modest impact on accuracy. For instance, decreasing τ from 0.90 to 0.75 increases token reduction from 11.91% to 18.04% on GUIOdyssey and from 5.89% to 17.40% on AndroidControl, while accuracy varies within a relatively narrow range. We adopt $\tau = 0.85$ on GUIOdyssey and $\tau = 0.75$ on AndroidControl as balanced operating points for subsequent experiments. The thresholds are selected on held-out validation data before final test evaluation. At these thresholds, PAVE achieves 43.35% accuracy

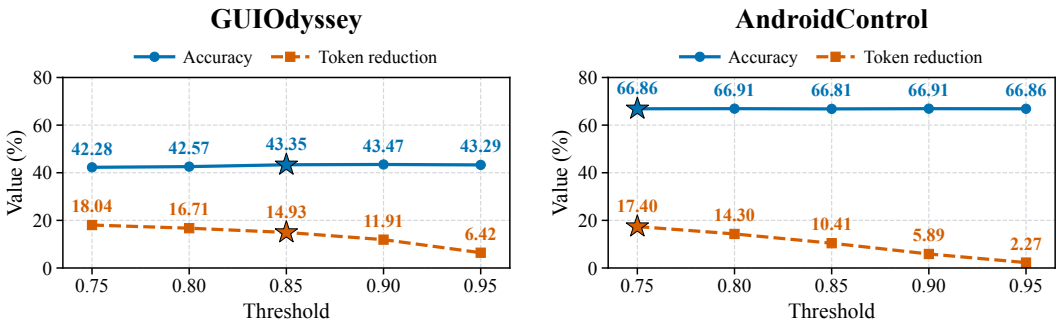


Figure 3: Performance of the experience retrieval threshold τ. Stars indicate the selected operating points.

Table 1: Accuracy comparison with state-of-the-art baselines.

Method	GUIOdyssey	AndroidControl
Uniform Grid	41.74%	65.91%
Random Selection	41.14%	64.31%
Random History	40.90%	62.13%
FastV	40.12%	62.51%
VisionZip	39.64%	62.35%
PAVE	**43.35%**	**66.86%**

with 14.93% token reduction on GUIOdyssey, and 66.86% accuracy with 17.40% token reduction on AndroidControl. Compared with the more conservative $\tau = 0.90$, these settings substantially increase token savings while maintaining comparable task performance. These results demonstrate that the retrieval threshold provides a direct mechanism for controlling the trade-off between exploiting cross-episode experience and preserving the visual information available to the action model.

Comparison with state-of-the-art baselines. We compare PAVE against five representative baselines, including three ablation strategies (*Uniform Grid*, *Random Selection*, and *Random-History Retrieval*), and two widely used visual token reduction methods, FastV (Chen et al., 2024) and VisionZip (Yang et al., 2025). Uniform Grid retains spatially distributed tokens across the screen, Random Selection uniformly samples tokens from the current screenshots, and Random-History Retrieval replaces relevance-based experience retrieval with randomly selected historical interactions. To ensure a controlled comparison, we evaluate all methods under the same visual-token budget. Comparing methods at their default retention rates can confound *selection quality* with token count. Fixing the token budget allows us to evaluate how effectively each method preserves task-relevant visual information.

As shown in Table 1, PAVE consistently achieves the strongest performance. This suggests that reducing visual tokens is not merely a matter of preserving spatial coverage: although Uniform Grid retains information broadly across the interface, it cannot distinguish between task-relevant regions and visually redundant content. In contrast, PAVE leverages relevant experience from previous interactions together with the current screen and task to allocate its limited token budget more selectively. Interestingly, general-purpose visual token reduction methods such as FastV and VisionZip do not perform as strongly, achieving 40.12% (GUIOdyssey) / 62.51% (AndroidControl) and 39.64% (GUIOdyssey) / 62.35% (AndroidControl) accuracy, respectively. We attribute this behavior primarily to two factors. First, these methods are designed for general vision-language inputs rather than GUI screenshots, whose fine-grained text, icons, and small interactive elements can be particularly sensitive to token removal. Second, they determine token importance primarily from the current input, without exploiting cross-episode interaction experience. Consequently, under a constrained token budget, they may discard visually small but interaction-critical information that PAVE can preserve using experience from related tasks. PAVE also maintains accuracy comparable to the full-token backbone (43.23% on GUIOdyssey; 66.86% on AndroidControl) while reducing visual tokens. As shown in Figure 3, PAVE selectively preserves tokens around task-relevant interface elements while aggressively reducing tokens from familiar or less informative regions. More examples are shown in Appendix G.

Generalization across backbones. While our main experiments use GUI-Owl-1.5-8B-Instruct as the backbone, we further investigate whether the effectiveness of PAVE generalizes to different vision-



Figure 4: Example visualization of retained visual tokens. Yellow regions indicate the tokens selected for processing.

language models. To this end, we additionally evaluate PAVE with Qwen2.5-VL-7B-Instruct (Bai et al., 2025) as an alternative backbone. We choose it because the Qwen models have been widely adopted as the foundation model for recent GUI agents (Bai et al., 2025). As shown in Table 3 in Appendix D, PAVE consistently reduces visual-token computation while maintaining competitive task accuracy when applied to Qwen. These results suggest that the benefits of persistent cross-episode visual experience are not specific to GUI-Owl, demonstrating the potential of PAVE as a general approach for improving the inference efficiency of different vision-language backbones in GUI tasks.

5.2 Online In-the-Wild Evaluation

Performance of PAVE. We conduct online experiments on AndroidWorld to evaluate PAVE under realistic end-to-end GUI interaction, where the agent observes changing screens and executes multi-step tasks. Appendix E reports the results. Under our default setting, PAVE achieves a task success rate of 48.2%, closely matching the 49.0% success rate of the original backbone without token reduction. Meanwhile, PAVE reduces the average number of visual tokens processed per screen by 8.29%. Although this reduction may appear moderate at the individual-screen level, the savings accumulate throughout a multi-step trajectory. We observe that the token reduction in online experiments is smaller than that achieved on the offline benchmarks. We attribute this mainly to two factors. First, live interaction introduces substantially more diverse and previously unseen interface states, making relevant historical experience less likely to satisfy the retrieval threshold and consequently triggering more conservative visual processing. Second, online trajectories evolve according to the agent's own actions and may deviate from previously observed interaction paths, particularly over longer tasks. As the current state becomes less aligned with retrieved experience, PAVE conservatively retains more visual information rather than relying on potentially irrelevant historical evidence. Despite these challenges, the consistent token savings with only a small change in task success demonstrate the practical potential of cross-episode visual experience for real-world GUI agents.

Effect of long-running GUI agent. A key property of PAVE is that its persistent experience grows as the agent completes more tasks. Consequently, a long-running GUI agent may encounter

increasingly familiar interfaces and leverage accumulated experience to further reduce redundant visual computation.

To investigate this behavior, we conduct a simulated long-running experiment in which the agent sequentially performs related tasks within the same application, while experiences from previously completed tasks are progressively added to the repository. Figure 5 shows how visual-token reduction changes as the agent accumulates more interaction experience. At the beginning, when little relevant experience is available, PAVE reduces visual tokens by only 0.71%. As more related tasks are completed and stored, the agent gains increasingly useful experience for subsequent interactions, with token reduction reaching 13.82% by the eighth task and continuing to increase as additional experience becomes available. This trend highlights an appealing property of PAVE: its efficiency can improve naturally with continued use, without modifying the underlying action model. Such behavior is particularly valuable for long-running real-world GUI agents. As interaction history accumulates, previously completed tasks can provide reusable visual experience for future tasks, allowing the computational cost of familiar GUI interactions to be progressively amortized over time.

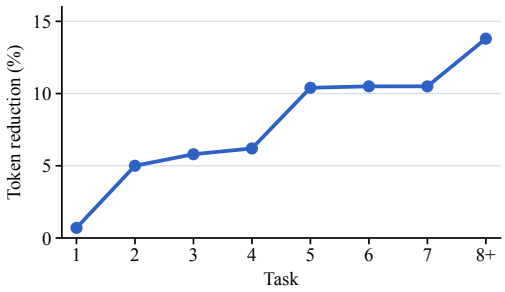


Figure 5: Token reduction within the same app.

6 Discussion

Our experiments demonstrate that PAVE substantially reduces visual-token computation while largely preserving agent effectiveness across both offline benchmarks and online in-the-wild evaluation. This suggests two promising applications. **Personal GUI assistants** repeatedly interact with a relatively stable set of applications, websites, and interface patterns on behalf of the same user (Wang et al., 2024; Xu et al., 2026a). In such settings, processing every interface entirely from scratch is potentially wasteful, since many visual elements and interaction patterns have already been encountered in previous tasks. By retaining experience from successful interactions, PAVE enables an agent to progressively exploit familiar visual information when handling subsequent tasks, making visual computation more efficient as experience accumulates. This application is particularly relevant to emerging on-device GUI agents (Xu et al., 2026a), such as Doubao AI Phone, where reducing visual computation may help alleviate the latency, memory, and energy constraints of continuously running capable multimodal models. Another promising application is **industry-scale GUI testing**, where large numbers of automated test cases are repeatedly executed across applications, devices, and software versions (Rawles et al., 2025; Xie et al., 2024). Such workloads frequently revisit similar interfaces, components, and interaction flows, causing GUI agents to repeatedly process highly redundant visual information. By accumulating experience from previously completed test cases, PAVE can reuse familiar visual patterns to reduce redundant computation in subsequent runs. As testing workloads scale, cross-episode visual reuse provides a potential way to amortize visual processing across test cases, enabling more efficient deployment of multimodal GUI agents in industrial testing systems.

7 Conclusion

In this work, we introduce PAVE, a **P**ersistent **A**cross-episode **V**isual **E**xperience framework for efficient GUI agents. Departing from existing approaches that primarily reduce visual redundancy within individual screenshot or interaction trajectories, PAVE exploits reusable visual experience accumulated across previously completed tasks. By retrieving relevant historical interactions and using them as priors for a lightweight experience-conditioned controller, PAVE adaptively identifies visual tokens that require further processing while keeping the underlying action model unchanged. Extensive experiments on both offline GUI benchmarks and online in-the-wild environments demonstrate that PAVE substantially reduces visual-token computation while largely preserving task effectiveness. Our results highlight persistent cross-episode experience as a promising direction for efficient GUI agents in two potential applications.

AI USE STATEMENT

We used generative AI tools to assist with code development and debugging, language editing and refinement of the manuscript, review of the manuscript's clarity and logical structure, figure and layout refinement, and literature search. All experiments were executed by the authors, and AI-assisted code, text, figures, and references were reviewed and verified by the authors. Generative AI was not used to generate experimental results or select reported results. The authors take responsibility for the final content of this work, including all claims, results, code, and figures.

REFERENCES

Amirhossein Abaskohi, Yuhang He, Peter West, Giuseppe Carenini, Pranit Chawla, and Vibhav Vineet. ReVision: Scaling computer-use agents via temporal visual redundancy reduction. *arXiv preprint arXiv:2605.11212*, 2026. URL https://arxiv.org/abs/2605.11212.

Saeed Ranjbar Alvar, Gursimran Singh, Mohammad Akbari, and Yong Zhang. DivPrune: Diversity-based visual token pruning for large multimodal models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 9392–9401, 2025. doi: 10.1109/CVPR52734.2025.00877. URL https://openaccess.thecvf.com/content/CVPR2025/html/Alvar_DivPrune_Diversity-based_Visual_Token_Pruning_for_Large_Multimodal_Models_CVPR_2025_paper.html.

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, Jiabo Ye, Xi Zhang, Tianbao Xie, Zesen Cheng, Hang Zhang, Zhibo Yang, Haiyang Xu, and Junyang Lin. Qwen2.5-VL technical report. *arXiv preprint arXiv:2502.13923*, 2025. URL https://arxiv.org/abs/2502.13923.

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. *arXiv preprint arXiv:2403.06764*, 2024. URL https://arxiv.org/abs/2403.06764.

Kanzhi Cheng, Qiushi Sun, Yougang Chu, Fangzhi Xu, Yantao Li, Jianbing Zhang, and Zhiyong Wu. SeeClick: Harnessing gui grounding for advanced visual gui agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 9313–9332, 2024. doi: 10.18653/v1/2024.acl-long.505. URL https://aclanthology.org/2024.acl-long.505/.

Wenyi Hong, Weihan Wang, Qingsong Lv, Jiazheng Xu, Wenmeng Yu, Junhui Ji, Yan Wang, Zihan Wang, Yuxiao Dong, Ming Ding, and Jie Tang. CogAgent: A visual language model for gui agents. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 14281–14290, 2024. URL https://openaccess.thecvf.com/content/CVPR2024/html/Hong_CogAgent_A_Visual_Language_Model_for_GUI_Agents_CVPR_2024_paper.html.

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. URL https://arxiv.org/abs/2510.00536.

Daiqiang Li, Zihao Pan, Zeyu Zhang, Xuyang Liu, Shijia Xu, Ronghao Chen, Huacan Wang, Honggang Chen, Linfeng Zhang, Zhangquan Chen, and Haiyun Jiang. Where and how to prune: An empirical study of visual token pruning for gui agent navigation, 2026a. URL https://arxiv.org/abs/2603.26041v4. arXiv v4, revised 9 August 2026.

Wei Li, William E. Bishop, Alice Li, Christopher Rawles, Folawiyo Campbell-Ajala, Divya Tyamagundlu, and Oriana Riva. On the effects of data scale on UI control agents. In *Advances in Neural Information Processing Systems*, volume 37, pp. 92130–92154, 2024. doi: 10.52202/079017-2925. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/a79f3ef3b445fd4659f44648f7ea8ffd-Abstract-Datasets_and_Benchmarks_Track.html.

Yuankai Li, Tinghui Zhu, Ha Min Son, Zhe Zhao, Xin Liu, and Muhao Chen. AQuaUI: Visual token reduction for GUI agents with adaptive quadtrees. *arXiv preprint arXiv:2605.19260*, 2026b. URL https://arxiv.org/abs/2605.19260.

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 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 19498–19508, 2025. URL https://openaccess.thecvf.com/content/CVPR2025/html/Lin_ShowUI_One_Vision-Language-Action_Model_for_GUI_Visual_Agent_CVPR_2025_paper.html.

Quanfeng Lu, Wenqi Shao, Zitao Liu, Lingxiao Du, Fanqing Meng, Boxuan Li, Botong Chen, Siyuan Huang, Kaipeng Zhang, and Ping Luo. GUIOdyssey: A comprehensive dataset for cross-app gui navigation on mobile devices. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 22404–22414, 2025. URL https://openaccess.thecvf.com/content/ICCV2025/html/Lu_GUIOdyssey_A_Comprehensive_Dataset_for_Cross-App_GUI_Navigation_on_Mobile_ICCV_2025_paper.html.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 8748–8763. PMLR, 2021. URL https://proceedings.mlr.press/v139/radford21a.html.

Chris 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. AndroidWorld: A dynamic benchmarking environment for autonomous agents. In *International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/01a83bc2f2732a58e6aa731e659e7101-Abstract-Conference.html.

Yuzhang Shang, Mu Cai, Bingxin Xu, Yong Jae Lee, and Yan Yan. LLaVA-PruMerge: Adaptive token reduction for efficient large multimodal models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 22857–22867, 2025. URL https://openaccess.thecvf.com/content/ICCV2025/html/Shang_LLaVA-PruMerge_Adaptive_Token_Reduction_for_Efficient_Large_Multimodal_Models_ICCV_2025_paper.html.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

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*, 2024. URL https://arxiv.org/abs/2401.16158.

Yuhao Wang, Mu Qiao, Xindong Zhang, Yunzhi Zhuge, Lei Zhang, and Huchuan Lu. TRACE: Trajectory-robust admission with evidence ordering for efficient GUI agents, 2026. URL https://arxiv.org/abs/2609.10297. arXiv preprint arXiv:2609.10297.

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. OSWorld: Benchmarking multimodal agents for open-ended tasks in real computer environments. *arXiv preprint arXiv:2404.07972*, 2024. URL https://arxiv.org/abs/2404.07972.

Haiyang Xu, Xi Zhang, Haowei Liu, Junyang Wang, Zhaozai Zhu, Shengjie Zhou, Xuhao Hu, Feiyu Gao, Junjie Cao, Zihua Wang, Zhiyuan Chen, Jitong Liao, Qi Zheng, Jiahui Zeng, Ze Xu, Shuai Bai, Junyang Lin, Jingren Zhou, and Ming Yan. Mobile-agent-v3.5: Multi-platform fundamental gui agents. *arXiv preprint arXiv:2602.16855*, 2026a. URL https://arxiv.org/abs/2602.16855.

Zhou Xu, Bowen Zhou, Qi Wang, Shuwen Feng, and Jingyu Xiao. Spatio-temporal token pruning for efficient high-resolution gui agents. *arXiv preprint arXiv:2602.23235*, 2026b. URL https://arxiv.org/abs/2602.23235.

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. URL https://openaccess.thecvf.com/content/CVPR2025/html/Yang_VisionZip_Longer_is_Better_but_Not_Necessary_in_Vision_Language_CVPR_2025_paper.html.

Ziyun Zeng, Hang Hua, Bocheng Zou, Mu Cai, Rogerio Feris, and Jiebo Luo. MementoGUI: Learning agentic multimodal memory control for long-horizon gui agents. *arXiv preprint arXiv:2605.18652*, 2026. URL https://arxiv.org/abs/2605.18652.

Qizhe Zhang, Mengzhen Liu, Lichen Li, Ming Lu, Yuan Zhang, Junwen Pan, Qi She, and Shanghang Zhang. Beyond attention or similarity: Maximizing conditional diversity for token pruning in MLLMs. In *Advances in Neural Information Processing Systems*, volume 38, pp. 28761–28791, 2025. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/2433fec2144ccf5fea1c9c5ebdbc3924-Abstract-Conference.html.

Bowen Zhou, Zhou Xu, Wanli Li, Jingyu Xiao, Pingan Gan, and Haoqian Wang. ST-Lite: Training-free KV cache compression with spatio-trajectory guidance for long-horizon GUI agents. *arXiv preprint arXiv:2603.00188*, 2026. URL https://arxiv.org/abs/2603.00188v3.

A Implementation Details

This appendix expands the routing, retrieval, and supervision choices introduced in Section 3. It first specifies the post-encoder interface, then the experience representation, and finally the controller inputs and verified training targets.

A.1 Selecting post-encoder tokens while preserving positions

Both evaluated backbones use dynamic-resolution visual lattices and merge each 2×2 patch group before the language/action model. For a pre-merger grid (T, H, W), the number of routed visual tokens is

$$N_{\text{post}} = THW/4. \tag{10}$$

PAVE operates after this merger and counts the tokens delivered to the action model. Selected indices are sorted in original row-major order and applied consistently to image placeholders, M-RoPE positions, primary visual features, and GUI-Owl deep-stack features. Before evaluating a backbone, an injection using all tokens must reproduce the unmodified backbone's generated token IDs exactly.

For Qwen2.5-VL, routing uses the merged `pooler_output`, not the pre-merger `last_hidden_state`. One audited screenshot has `image_grid_thw=[1,222,102]`: 22,644 raw patches become 5,661 merged 3,584-dimensional tokens and exactly 5,661 image placeholders. This is an interface example, not a fixed dataset-wide token count.

A.2 Experience bank and retrieval

An experience bank page template contains retrieval embeddings, app and episode scope, task and lattice metadata, but no spatial mask. A record with a sparse mask additionally stores retained token indices. Offline masks are verified by correct grounding; in AndroidWorld, records are staged within an episode and committed only after the environment reports success for the complete task. Failure or interruption discards them. Only controller decisions retaining at most a fraction $\rho_{\text{store}} = 0.80$ of current-screen visual tokens store sparse masks; the remaining records are stored as page templates. The success criterion controls admission, while ρ_{store} requires stored masks to be sufficiently sparse. During read-only evaluation, hashes of records, labels, and embedding arrays are checked before and after testing.

Retrieval uses frozen CLIP screen, task, and OCR embeddings with score

$$s(t, j) = .5\langle z_t^s, z_j^s\rangle + .3\langle z_t^q, z_j^q\rangle + .2\langle z_t^o, z_j^o\rangle. \tag{11}$$

Here, z^s, z^q, and z^o denote the normalized screen, task, and OCR embeddings, respectively, and $\langle\cdot,\cdot\rangle$ denotes cosine similarity. Only eligible records are ranked: benchmark queries use the fixed training-only experience bank and exclude their own episode; AndroidWorld uses app-scoped retrieval, with task-conditioned launcher handling. Up to $K_{\text{ret}} = 3$ eligible records are retrieved, and the top score determines whether the similarity gate passes. Among them, records with sparse masks contribute to the spatial prior; their masks are bilinearly projected from historical lattices to the current lattice. Similarity weights are normalized over these mask-bearing records. Weighted disagreement is $\sum_j \alpha_j \tilde{m}_j(i)^2 - (\sum_j \alpha_j \tilde{m}_j(i))^2$, clamped to zero for numerical roundoff. Coverage divides the number of mask-bearing records by the number actually retrieved. Page templates can permit controller use but alone give a zero spatial prior; protected system surfaces use the full-token fallback.

A.3 Controller inputs and cached training records

The CLIP retrieval vectors and GUI-Owl visual tokens are distinct representations. Frozen, ℓ_2-normalized CLIP screen, task, and OCR embeddings rank eligible experience bank records. The controller also uses the same 512-dimensional CLIP task embedding, $q_t = z_t^q$; GUI-Owl instead receives its original task prompt g_t. After verified sparse mask construction, each training or validation query is encoded once by frozen GUI-Owl. Its cached controller record contains the merged post-encoder visual features $V_t \in \mathbb{R}^{N_t \times d_v}$, the normalized two-dimensional centers X_t, q_t, and the verified sparse mask $m_t^\star$. Masks among the K_{ret} retrieved records are projected into the position-aligned prior $P_t \in \mathbb{R}^{N_t \times 3}$. Thus the controller input is (V_t, X_t, q_t, P_t); it receives neither a historical screenshot nor a historical action. Frozen GUI-Owl features are cached rather than recomputed in each training epoch. All four streams are projected to the controller's 256-dimensional hidden space.

Table 2: Frozen retrieval, controller, and experience-storage hyperparameters.

Parameter	Value	Used in	Rationale
K_{ret}	3	retrieval	maximum retrieved records
p	.05	offline mask selection	frozen operating point
p	.20	in-the-wild mask selection	frozen operating point
τ	.90	in-the-wild router	fixed retrieval threshold
ρ_{store}	.80	bank write	store sufficiently sparse masks
T	.05	prior weights	fixed retrieval softmax temperature

A.4 Verified controller supervision

We begin with states on which the unmodified backbone produces a click inside the annotated target box. Let G_t^{+h} contain the tokens inside the target box after dilation by h post-encoder lattice cells, with $G_t = G_t^{+0}$; let O_t contain on-screen OCR cells matched to the task or previous action; let U_t denote the top and bottom layout bands, each covering approximately 8% of token rows; and let $H_{t,10}$ and $H_{t,20}$ contain the highest-scoring 10% and 20% of cells under projected masks from eligible training records. Before removal of duplicate index sets, the fixed candidate family is

$$\begin{aligned}\mathcal{C}_t = \{&G_t, G_t^{+1}, G_t^{+2}, G_t^{+4}, G_t \cup O_t, G_t^{+1} \cup O_t,\\ &G_t \cup U_t, G_t^{+1} \cup U_t, G_t \cup H_{t,10}, G_t \cup H_{t,20},\\ &G_t \cup O_t \cup H_{t,20}, \{1, \ldots, N_t\}\}.\end{aligned} \tag{12}$$

Each candidate is evaluated by rerunning frozen GUI-Owl. The smallest candidate in this fixed family that preserves a bbox hit is the verified sparse mask $m_t^\star$; if no sparse candidate succeeds, the state is not used as a sparse training target. The fine-stage target is $m_t^\star$. The coarse-stage target $c_t^\star$ marks every token in any cell of the normalized 4×4 anchor grid containing a positive token in $m_t^\star$. Here 4×4 denotes 16 spatial cells, not a fixed number of tokens per cell.

The coarse and fine stages each use two cross-attention blocks with 256 hidden dimensions, four heads, 0.1 dropout, and 4×4 anchors. Training uses $\lambda_{\text{keep}} = 0.10$ for the fine-stage keep-rate term, $\lambda_{\text{reg}} = 0.03$ for the coarse-stage keep-rate term, and $\lambda_c = 0.35$ for the coarse-stage loss. AdamW uses learning rate 3×10^{-4}, weight decay 10^{-2}, gradient-norm clipping at 1.0, batch size 2, 50 epochs, and seed 20260812. Checkpoints are selected by validation loss.

The fine-stage loss uses $\lambda_b = \lambda_{\text{keep}}$ and the coarse-stage loss uses $\lambda_b = \lambda_{\text{reg}}$. Fine-loss gradients also reach the coarse stage through its differentiable soft region map and updated token features. Hard token selection is used only at inference. Table 2 collects the frozen retrieval and storage settings; p is the sigmoid keep-probability threshold, distinct from retrieval threshold τ.

Together, these choices keep the action backbone frozen while making both the routing decision and the controller supervision explicit and reproducible.

B Evaluation Protocols

This appendix details the offline benchmark and in-the-wild evaluation protocols summarized in Section 4. It first covers the two recorded benchmarks, GUIOdyssey and AndroidControl, and then the AndroidWorld in-the-wild evaluation.

For both recorded benchmarks, a fixed *safety abstention* rule is applied before inference. A state is abstained when a case-insensitive match to any of `pin`, `passcode`, `password`, `payment`, `pay`, `card`, `billing`, `address`, `permission`, `allow`, `verification`, `verify`, `otp`, `login`, or `sign in` appears in the task text, previous action, or on-screen OCR. The rule is fixed before evaluation and uses no model prediction, retrieval score, target annotation, or task outcome. Neither PAVE nor a comparison method is executed on an abstained state, and every method is evaluated on the same remaining states.

B.1 GUIOdyssey test states

The GUIOdyssey labeling resources contain 21,000 states, 9,122 backbone bbox hits eligible as earlier experience, 6,425 eligible training records in the fixed experience bank, 4,212 training queries, and 853 validation queries. The experience bank is the stored collection of eligible records; its training support is the subset from which validation and test queries may retrieve. Validation queries retrieve only from that training support and exclude their own episode. Grounding accuracy measures whether an individual predicted click lies in the annotated target box; task success rate measures completion of entire interactive tasks.

The separate GUIOdyssey test sample contains 2,000 states from episodes not used for controller training or parameter selection. The safety-abstention rule abstains on 330 states, leaving 1,670 evaluated states. Grounding accuracy and token reduction are computed over these common evaluated states.

B.2 AndroidControl test states

AndroidControl provides 15,283 human demonstrations over 833 Apps, with episode-level goals, screenshots, accessibility trees, and JSON actions (Li et al., 2024). It is the second recorded benchmark for multi-step trajectories and history construction. The frozen protocol is:

1. Preserve the official episode split and trajectory order. The benchmark denominator contains click actions with an in-bounds demonstrated coordinate and valid target box; other action types remain in the dataset audit but not the bbox-grounding experiment.
2. Derive the target as the smallest-area accessibility node enclosing the demonstrated click, and normalize the click and box through the same screenshot dimensions.
3. Construct verified masks and train a fresh controller from training queries only. Training retrieval excludes the query episode; validation selects the checkpoint; test queries never contribute retrieval records.
4. Evaluate a separate sample of 2,000 official test states with no bank updates. All token-selection methods use the same routing decisions, backbone fallback states, and per-state token counts as PAVE.
5. Report grounding accuracy, post-encoder tokens from the current screen, frequency of controller use, safety abstentions, and paired uncertainty using the actual split-specific denominators.

AndroidControl is a recorded benchmark rather than an in-the-wild interactive setting: it supplies demonstrated trajectories but cannot programmatically verify whether GUI-Owl completes a task. AndroidWorld supplies the task success rate measurement.

The unmodified backbone pool contains 15,513 training states, 420 validation states, and 5,067 test states. Its 12,470 training bbox hits form the fixed experience bank support; the test split never enters the bank. The test sample draws 2,000 official states from 1,484 episodes, with no more than two states from any one episode. The safety-abstention rule abstains on 117 states, leaving 1,883 evaluated states for the token and grounding comparison.

Matched-token baselines. Uniform Grid selects spatially distributed current-screen tokens, and Random Selection samples token positions. Random History supplies random eligible history to the same controller. FastV and VisionZip are adaptations to GUI-Owl: FastV ranks image positions using the final prompt token's attention in the second language-model layer, then reruns action generation with the selected positions. VisionZip uses similarity to the mean visual feature in place of CLS attention, allocates 75% of the token budget to dominant tokens and the remainder to contextual representatives, and averages assigned features into those representatives, including deep-stack features. These comparisons match the final visual-token count.

B.3 AndroidWorld

Ten of 116 task specifications are reserved for development and three high-risk specifications are excluded, leaving 103 test task specifications across 21 app packages. Tasks are grouped by package.

Table 3: **Qwen2.5-VL-7B-Instruct evaluation.** Grounding accuracy and visual-token reduction on held-out GUIOdyssey and AndroidControl states. All token-selection methods use PAVE's per-state token budget; the unmodified backbone processes all current-screen visual tokens.

Method	GUIOdyssey Accuracy	GUIOdyssey Token Reduction	AndroidControl Accuracy	AndroidControl Token Reduction
PAVE	**29.52%**	**19.07%**	**44.66%**	**10.70%**
Uniform Grid	26.77%	19.07%	42.38%	10.70%
Random Selection	27.25%	19.07%	43.87%	10.70%
Random History	28.02%	19.07%	44.18%	10.70%
FastV (adapted)	28.26%	19.07%	43.34%	10.70%
VisionZip (adapted)	25.27%	19.07%	42.17%	10.70%
Unmodified backbone	29.40%	0.00%	44.34%	0.00%

Each backbone/PAVE pair receives the same task seed and deterministic reinitialization. A pair is retained when initial screenshots are pixel-identical (class A), or when all differences are confined to prespecified decorative regions and an unmodified backbone probe yields the same tool-call JSON (class C). Structural differences, different probe actions, exceptions, or goal-text mismatches are excluded before scoring.

C Statistical Analysis and Reproducibility

This appendix specifies the uncertainty estimates and audit artifacts referenced by the main evaluation. AndroidWorld n counts task pairs that pass the initial-observation audit, have measurements from both methods, and have identical goal text. Visual-token reduction is computed as $(1 - \sum_{t\in\mathcal{E}} |K_t| / \sum_{t\in\mathcal{E}} N_t) \times 100\%$, where $\mathcal{E}$ contains the evaluated decisions. Task bootstraps resample tasks; app-cluster bootstraps resample foreground packages and retain all tasks in each sampled package. GUIOdyssey bootstraps resample paired states; AndroidControl test intervals resample episodes and retain all evaluated states in each sampled episode.

All reported configurations freeze the action backbone, prompt, parser, controller checkpoint, retrieval scope, and token accounting before evaluation. The reproducibility package records split manifests, task order, bank hashes, controller configuration, action-parser version, per-state retained indices, and paired-result ledgers. These artifacts connect every aggregate in the paper to its frozen experimental inputs.

D Qwen Evaluation

This appendix reports the separate evaluation with Qwen2.5-VL-7B-Instruct (Bai et al., 2025). The backbone is fully frozen, and Qwen-specific controllers are trained independently; GUI-Owl features and controller weights are not reused. An injection using all tokens produces exactly the same generated token IDs and answer as the unmodified Qwen model. PAVE then applies sorted indices while preserving positions on the post-merger lattice. The prompt requests one JSON click in raw screenshot pixels; the parser accepts only in-bounds coordinates and normalizes them to GUIOdyssey's 0–1000 box frame. Non-click, unparsable, and out-of-bounds outputs are invalid without repair.

E In-the-Wild Evaluation

The main AndroidWorld evaluation begins each independent run with an empty experience bank; successful episodes add records that may be retrieved by later tasks in the same app. We additionally evaluate the same seed-101 task instances with a fixed experience bank to measure performance when previously accumulated experience is available from the start.

Table 4: Token-reduction uncertainty. Tasks or foreground Apps, rather than individual steps, are resampled.

Run	Valid tasks	Reduction	Task-bootstrap 95% CI	App-cluster 95% CI
Round 1	87	9.46%	$[5.44, 14.39]$	$[3.75, 16.99]$
Round 2	82	7.49%	$[5.23, 10.16]$	$[4.76, 10.55]$
Round 3	86	7.86%	$[5.52, 10.54]$	$[5.46, 10.70]$
Main pooled	255	8.29%	$[6.47, 10.47]$	$[4.85, 11.86]$
Fixed bank	71	**22.11%**	$[16.88, 27.32]$	$[14.41, 31.35]$

E.1 Uncertainty and experience available at each decision

Table 4 gives task- and app-resampled uncertainty for the three independent runs, their pooled result, and the additional fixed-bank evaluation. The main result pools 255 valid task pairs and 3,646 PAVE model decisions; 793 decisions invoke the controller (21.75%). The backbone succeeds on 125/255 tasks and PAVE on 123/255, with seven PAVE-only and nine backbone-only successes. The paired success difference is -0.8 percentage points (95% CI $[-3.8, 2.5]$; exact two-sided McNemar $p = 0.804$). This does not establish equivalence.

Table 5 is a separate descriptive telemetry analysis: it includes all logged PAVE model calls, excluding handoffs, without restricting to the valid paired-task subset. Its 4,242 main-run and 1,450 fixed-bank records therefore do not aggregate to the paired results (3,646 and 1,052 decisions, respectively).

Table 5: Experience availability in all logged PAVE model calls, excluding handoffs. This telemetry includes calls outside the valid paired-task subset and is not a decomposition of the main evaluation.

Evaluation	Memory state	Steps	Share	Controller use	Tokens/decision
Main logs	none	2098	49.5%	0.0%	2550.0
Main logs	page template only	1576	37.2%	34.1%	2197.2
Main logs	records with sparse masks	568	13.4%	68.3%	1913.4
Fixed-bank logs	none	414	28.6%	0.0%	2550.0
Fixed-bank logs	page template only	604	41.7%	63.7%	1940.9
Fixed-bank logs	records with sparse masks	432	29.8%	**85.0%**	**1682.8**

The uncertainty estimates support token reductions in both paired evaluations. The separate telemetry describes the association between retrieved evidence and token cost; its different inclusion rule precludes using it to recompute the paired aggregates.

E.2 Evaluation with a fixed experience bank

The fixed experience bank is the final bank produced by successful episodes in the seed-101 run of the main in-the-wild evaluation. It contains 477 records: 351 page templates and 126 records with sparse masks. Of the 351 page templates, 314 come from successful unmodified-backbone decisions and 37 come from successful controller decisions whose masks did not satisfy the sparse-mask storage criterion. The records are not human demonstrations.

The additional evaluation revisits the same seed-101 task instances in the same order. The bank remains read-only, no records are written during evaluation, and its hash is unchanged before and after the run. Across 71 valid task pairs, the unmodified backbone succeeds on 34/71 tasks (47.9%) and PAVE on 33/71 (46.5%). There are two PAVE-only and three backbone-only successes (exact two-sided McNemar $p = 1.000$). Across 1,052 PAVE model decisions, controller use is 58.46% and mean token cost is 1,986.1, a 22.11% reduction relative to 2,550.0.

E.3 Experience accumulation and revisits

Table 6 provides the numerical values for each app visit number. The $8+$ row pools the eighth and later visits, rather than describing another experiment. Each row groups logged tasks by app visit number within each independent run, before pooling. This analysis covers 278 logged task episodes and 3,366 dominant-app decisions, excluding launcher, keyboard, and other system-surface steps; it

Table 6: Descriptive app-exposure telemetry grouped by app visit number. 8+ pools the eighth and later visits. Only decisions in each task's dominant foreground app are counted; the population is not restricted to valid paired tasks.

App visit number	Tasks	Steps	Candidates/step	Controller use	Token reduction
1	58	732	0.03	1.23%	0.71%
2	45	621	3.23	13.20%	4.95%
3	39	471	6.77	15.71%	5.75%
4	29	305	10.80	17.70%	6.15%
5	22	162	13.11	25.31%	10.38%
6	14	139	19.37	23.02%	10.46%
7	12	137	22.76	29.20%	10.47%
8+	59	799	41.30	41.68%	**13.82%**

differs from both the paired-task subset and all-call telemetry. Later rows include *different* tasks in the same app; the fixed-bank evaluation instead revisits the seed-101 instances.

Among 68 tasks valid in both Round 1 and the fixed-bank evaluation, token reduction increases by 13.08 pp (app-cluster 95% CI $[8.42, 16.98]$) and controller use by 34.86 pp ($[22.96, 44.14]$). Controller use increases for 40 tasks, decreases for four, and is unchanged for 24. Together, these analyses show that accumulated successful experience makes token selection available more often.

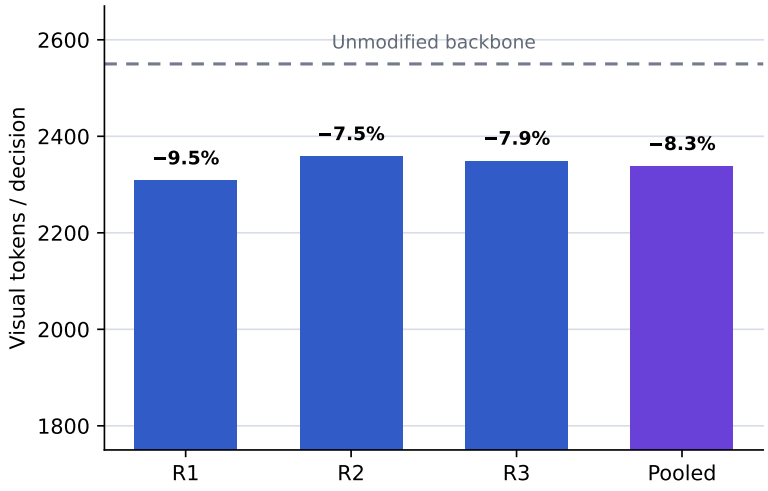


Figure 6: **In-the-wild token cost.** Bars show post-encoder current-screen visual tokens per decision for the three independent in-the-wild runs and their pooled result. Labels report reduction relative to the unmodified backbone.

F Runtime Efficiency

To determine whether the additional modules increase runtime, we compare the end-to-end runtime of PAVE with the unmodified backbone. Across all evaluated states, PAVE averages 3.743 seconds, compared with 3.762 seconds for the unmodified backbone. On states using the controller, mean latency decreases by 3.0%, from 3.665 seconds to 3.556 seconds. These results show that the additional modules do not increase measured end-to-end runtime.

G Qualitative Routing Evidence

Figure 4 presents one GUIOdyssey and one AndroidControl example in the main paper. Figure 7 compares four token-selection methods on the same GUIOdyssey screen at the same token cost. Figures 8–11 provide additional examples from both benchmarks.

As seen in several cases, the selected tokens include parts of the top and bottom of the screen. One reason is that the training-mask candidates include the target region together with these layout bands, which can preserve headers and navigation controls. Such a candidate becomes a training target only if the frozen action model still clicks inside the target box. The controller can therefore learn to retain edge regions when useful; it does not follow a fixed rule to keep them on every screen.

The additional examples make the projection-and-re-scoring process explicit. In each case, the retrieved historical interface provides a spatial prior that is projected onto the current screen and then re-scored by the controller. The final selected tokens concentrate on the action-relevant region while retaining local context when needed, across both GUIOdyssey and AndroidControl.

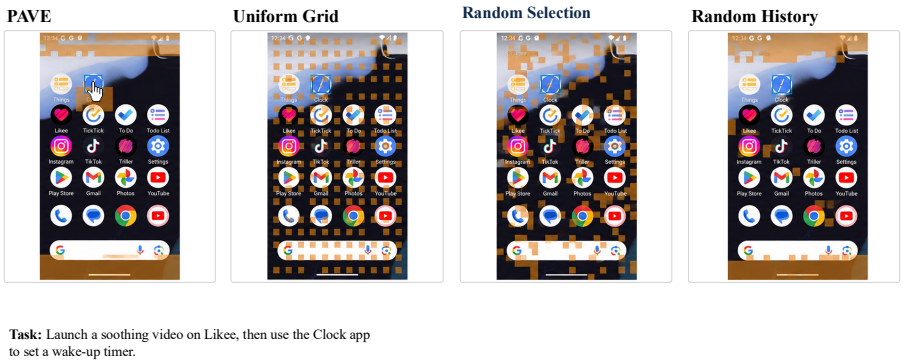


Figure 7: **Different token selection methods.** PAVE, Uniform Grid, Random Selection, and Random History each retain 184 of 920 current-screen visual tokens on the same GUIOdyssey state. Only PAVE executes the click; the other three methods do not execute a click and therefore fail on this state.



Figure 8: **Additional qualitative examples on GUIOdyssey.**

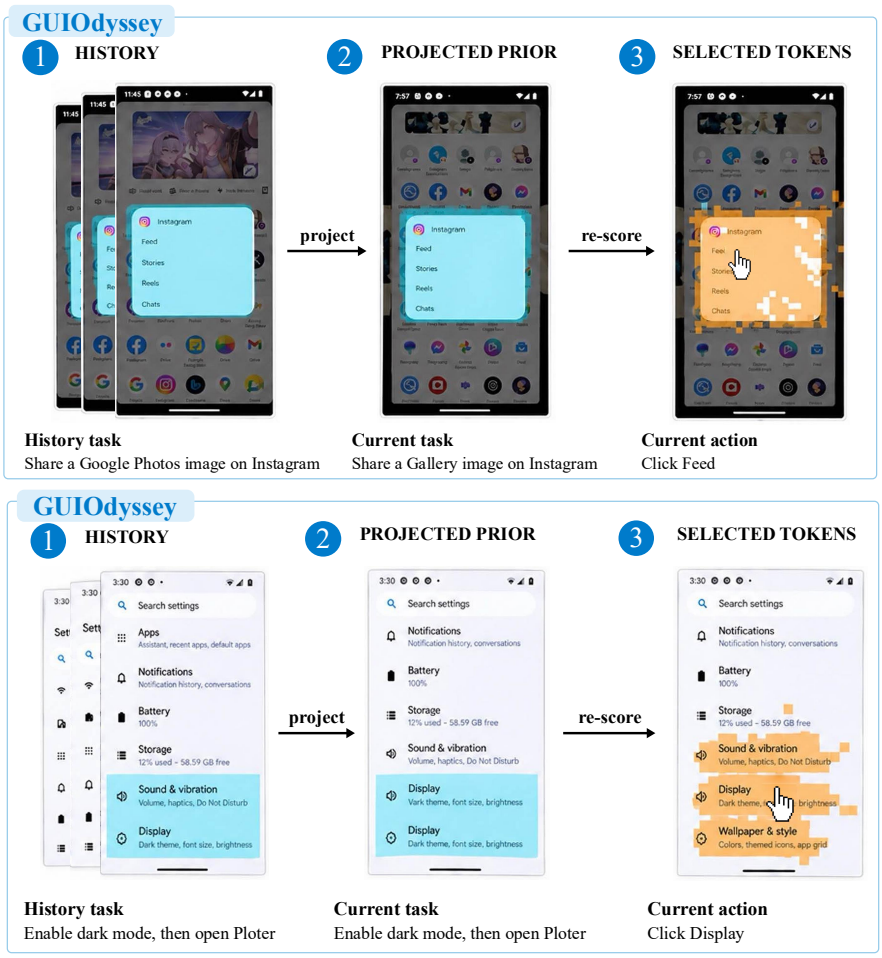


Figure 9: **Additional qualitative examples on GUIOdyssey (continued).**



Figure 10: **Additional qualitative examples on AndroidControl.**

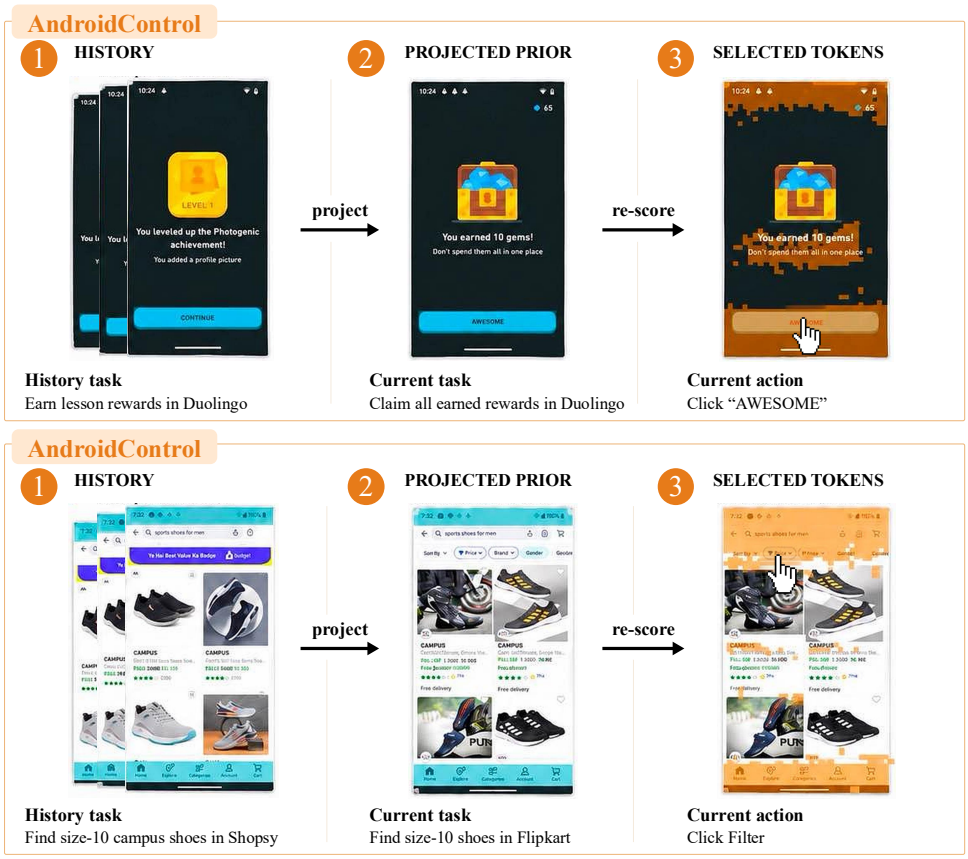


Figure 11: **Additional qualitative examples on AndroidControl (continued).**