GUIFLOW: TEMPORAL REDUNDANCY-AWARE INCREMENTAL INFERENCE FOR GUI AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

GUI agents built on vision-language models interact with continuously evolving interfaces, yet conventional inference repeatedly recomputes representations of largely overlapping observations and histories. This mismatch makes multi-step interaction costly, while changes in context, interface content, and action arguments prevent straightforward cache reuse. We introduce GUIFlow, a framework for full-pipeline incremental inference in multi-image GUI interaction. Our central insight is that advancing a decision need not require rebuilding all supporting representations: each interaction step can instead selectively update computational states maintained across the trajectory. To support new decisions as the context evolves, GUIFlow adapts how past computation is reused: historical KV states are position-aligned and injected into the current context; change signals guide selective updates of current-observation states in the vision encoder and language model; and earlier action responses serve as speculative drafts verified by the target model. Together, these mechanisms enable cross-step reuse throughout visual encoding, prefill, and decoding without additional training, a separate draft model, or changes to the model-visible inputs and action interface. Experiments demonstrate up to $3.39\times$ geometric-mean speedup across offline multi-image benchmarks and up to $3.49\times$ in online, closed-loop interaction on OSWorld, while maintaining task performance broadly comparable to the dense baseline.

1 INTRODUCTION

AI agents are moving beyond answering questions toward executing tasks in digital environments, including everyday applications on computers and mobile devices. Graphical user interfaces (GUIs) provide a broadly applicable way to interact with existing software without relying solely on application-specific APIs. Advances in vision-language models (Bai et al., 2025) have enabled GUI agents to interpret screens and act on them through repeated observation and action (Qin et al., 2025; Xu et al., 2026a; Xie et al., 2024). However, completing multi-step tasks requires sustained model inference, whose cumulative latency limits the responsiveness and practical usability of these agents.

The continuity of interaction offers an opportunity to reduce this cost. Successive decision steps share supporting information: historical screenshots recur across overlapping context windows, neighboring observations share persistent interface regions, and action responses often repeat structural patterns. For example, editing a text field changes only part of a screen, while the surrounding interface and much of the interaction history remain available from the preceding step. Figure 1 provides evidence at both the image and representation levels: across the evaluated tasks, more than 70% of observation pairs have RGB-image error no greater than 5% at each temporal gap from one to four steps. The illustrated KV-drift maps show similar spatial patterns of change in an early and the final layer, while the trajectory example shows substantial state stability between major interface changes. These observations suggest that advancing a decision can build on persistent computational states and selectively update what changes, rather than rebuild all supporting representations at every step.

Recent efficiency methods exploit temporal structure through KV-cache compression, as in GUI-KV and ST-Lite (Huang et al., 2025; Zhou et al., 2026), or learned visual-patch reduction, as in ReVision (Abaskohi et al., 2026). Reducing visual context, however, does not by itself make past computational states reusable under new conditions or ensure benefits across the full inference pipeline, raising two broader challenges. The first is **maintaining reusable computation under**

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.

evolving interactions: past computational states must support new decisions even as their context changes, requiring a distinction between what remains reusable and what needs updating. The second is **turning local reuse into pipeline-wide acceleration**: savings within individual stages may be limited by computation elsewhere and by the overhead of reuse itself. Addressing this challenge requires extending reuse across visual encoding, prefill, and decoding while translating computational savings into actual latency reductions. These challenges motivate our central question: *Can GUI agents accelerate real-world interaction through full-pipeline incremental inference, selectively updating existing computational states as the interaction evolves?*

To address these challenges, we introduce **GUIFlow**, a temporal redundancy-aware framework for full-pipeline incremental inference in GUI agents. GUIFlow treats each interaction step as a selective update to computational states maintained across the trajectory, rather than an independent inference pass. For historical inputs, it injects position-aligned KV states for recurring screenshots into the current context, enabling approximate reuse beyond unchanged prefixes. For current observations, change signals guide selective updates in the vision encoder and deeper language-model layers, while source tracking and reuse limits control state staleness. For action generation, earlier responses serve as speculative drafts rather than fixed answers, with the target model verifying draft tokens and correcting mismatches. Together, these mechanisms extend reuse across visual encoding, prefill, and decoding, reducing repeated computation beyond any single stage. GUIFlow requires neither additional training nor a separate draft model, and leaves the model-visible inputs and action interface unchanged.

We evaluate GUIFlow on AITW (Rawles et al., 2023) and Mind2Web (Deng et al., 2023; Cheng et al., 2024) for offline multi-image navigation and on OSWorld (Xie et al., 2024) for online closed-loop interaction, covering both fixed trajectories and evolving environments. GUIFlow achieves up to $3.39\times$ geometric-mean speedup across the offline benchmarks and up to $3.49\times$ speedup on OSWorld, while maintaining task performance broadly comparable to the dense baseline. Across these settings, mean per-step processing latency is reduced by factors of 1.94–2.55. These results show that selective state updates can turn temporal redundancy into both computational savings and lower execution latency. Our contributions are threefold:

(1) We introduce an incremental inference paradigm for GUI agents. This paradigm reframes successive decisions as selective updates to persistent computational states rather than independent inference passes, exploiting temporal redundancy across interaction steps.

(2) We propose GUIFlow, a training-free framework for full-pipeline incremental inference. GUIFlow integrates historical KV injection, source-aware current-state refinement, and verified action generation to extend cross-step reuse across visual encoding, prefill, and decoding.

(3) We demonstrate the effectiveness of GUIFlow in both offline navigation and online closed-loop interaction. Experiments on AITW, Mind2Web, and OSWorld show substantial speedups and lower processing latency while maintaining task performance broadly comparable to dense inference.

2 Related Work

Efficient GUI Agents. Visual-token reduction (Alvar et al., 2025; Li et al., 2025c), operator skipping (Luo et al., 2026), and KV-cache compression (Li et al., 2024) offer complementary routes to reducing inference cost. FastV prunes visual tokens after early decoder layers (Chen et al., 2024), whereas VisionZip selects dominant tokens and merges contextual information before language-model processing (Yang et al., 2025). For GUI interaction, ShowUI introduces interface-aware token selection (Lin et al., 2025). GUI-KV, STaR-KV, and ST-Lite exploit spatial importance and temporal structure to compress screenshot-history KV caches (Huang et al., 2025; Han et al., 2026; Zhou et al., 2026), while ReVision learns to remove temporally redundant visual patches and fine-tunes the agent on compressed histories (Abaskohi et al., 2026). Complementing input and cache compression, GUIFlow focuses on selectively updating persistent computational states across successive decisions.

Computation Reuse and Incremental Inference. Temporal redundancy motivates both state caching (Xu et al., 2026b; Ji et al., 2026) and temporal-aware acceleration (Li et al., 2025b; 2026; 2025a) in embodied control. For language models, prefix caching reuses KV states for identical leading contexts, but recurring content outside an unchanged prefix requires additional treatment

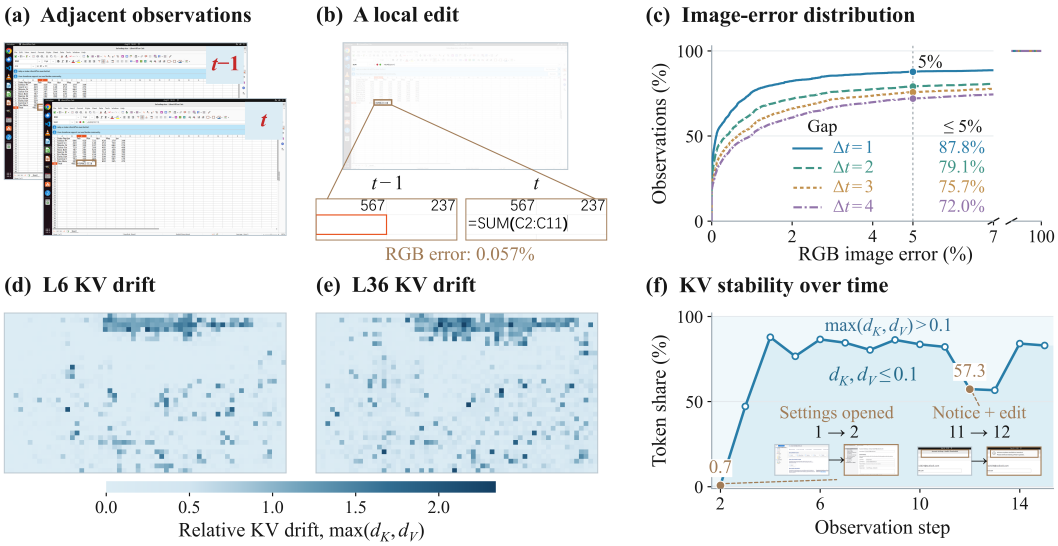


Figure 1: **Incremental changes in GUI observations and internal representations. (a–c) Image-level changes.** A local edit modifies only a small part of successive screenshots; across the evaluated tasks, more than 70% of observation pairs have RGB-image error at most 5% at each temporal gap $\Delta t \in \{1, 2, 3, 4\}$. **(d–f) Representation-level changes.** For the same observation pair, KV-drift maps at layer 6 and the final layer 36 show broadly similar spatial concentrations of change. Along the illustrated trajectory, a large share of tokens has small KV drift ($d_K, d_V \leq 0.1$) during stable-interface segments, with drops around substantial interface changes. Together, these observations motivate selectively updating changing states rather than rebuilding all representations at every step.

of positional and contextual changes. CacheBlend addresses this issue in retrieval-augmented generation by selectively recomputing states to account for cross-chunk interactions (Yao et al., 2025). TRACE combines visual-token selection with cross-step cache reuse for GUI agents (Wang et al., 2026). At the generation stage, speculative decoding evaluates candidate continuations with the target model, allowing multiple tokens to be accepted in one verification pass (Leviathan et al., 2023); REST retrieves drafts from existing text without a separate draft model (He et al., 2024). GUIFlow coordinates historical-state reuse, current-observation refinement, and draft verification under evolving contexts, organizing visual encoding, prefill, and decoding into a unified incremental inference process.

3 Method

3.1 Full-Pipeline Incremental Inference

At step t, a task instruction q, current screenshot I_t, and history H_t of preceding screenshots and actions form the agent's input X_t. Step-independent execution encodes the screenshots, prefills the language model, and autoregressively generates a response y_t for action execution. Despite KV caching within each response, this execution can repeatedly process overlapping histories and persistent interface content across steps. Figure 1 examines whether changes between decisions require rebuilding these representations. At the image level, panels (a–b) illustrate a localized edit, while panel (c) aggregates observation pairs across tasks: the share with RGB-image error at most 5% is 87.8% for adjacent steps and remains 72.0% at a four-step gap. Thus, visual continuity persists beyond immediately adjacent observations, providing opportunities to carry forward existing visual states.

At the representation level, panels (d–e) show that the same observation pair produces broadly similar spatial concentrations of KV drift at layer 6 and the final layer 36. Panel (f) further shows that a large share of tokens retains small KV drift during stable-interface segments, whereas substantial interface changes coincide with drops in this share. These patterns motivate using early-layer deviations to guide deeper-state updates and monitoring state changes as interaction progresses, rather than treating

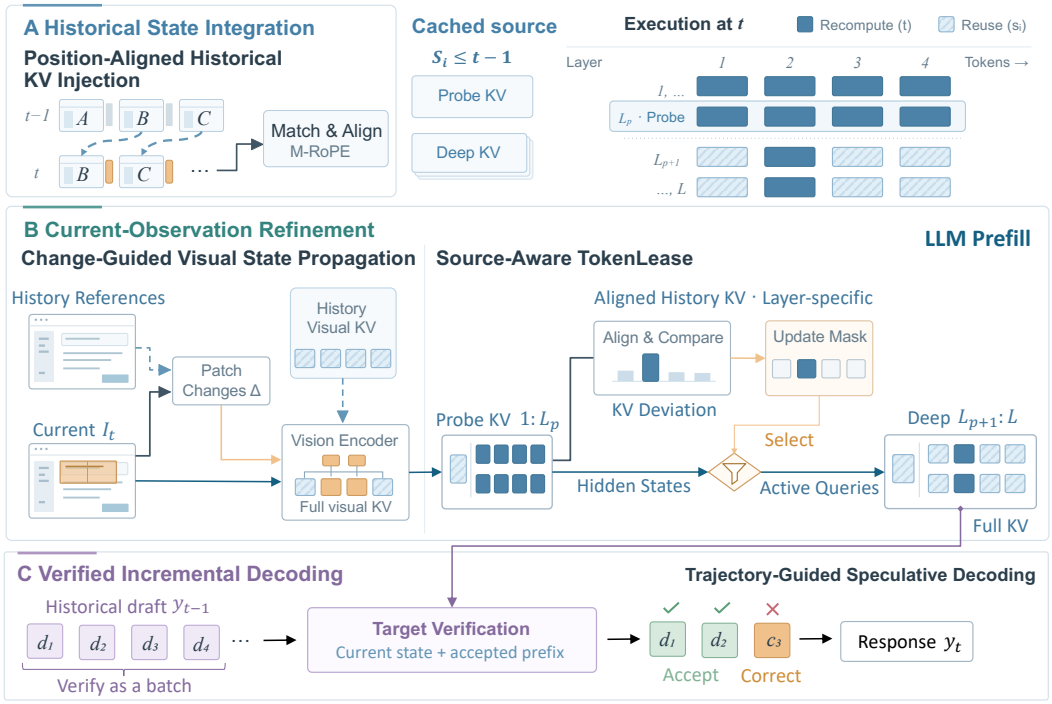


Figure 2: **Overview of GUIFlow.** GUIFlow treats successive GUI decisions as incremental updates to existing computation rather than independent inference passes. **(A) Historical State Integration.** Historical KV states are positionally aligned and injected into the current context. **(B) Current-Observation Refinement.** Pixel changes guide visual-encoder updates, while source-aware KV deviations guide deeper language-model updates. **(C) Verified Incremental Decoding.** Conditioned on the prefill state formed by (A) and (B), the target model verifies historical response drafts, accepting matching tokens and correcting mismatches. Together, these mechanisms exploit temporal redundancy across visual encoding, prefill, and decoding without additional training or a separate draft model.

every new screenshot as entirely new computation. Because position and preceding context also affect KV states, visual similarity alone is not sufficient to decide reuse.

Building on these observations, GUIFlow treats each new decision as a selective update to computation maintained within the same task trajectory. Let $\mathcal{C}_{t-1}$ denote the state retained before step t. Incremental inference produces both the current response and the state available to the next step:

$$(y_t, \mathcal{C}_t) = \mathcal{G}_\theta^{\text{inc}}(X_t, \mathcal{C}_{t-1}), \tag{1}$$

where $\mathcal{G}_\theta^{\text{inc}}$ is an execution procedure for the fixed model with parameters θ, not a separately trained model. The maintained state includes historical KV caches, visual and language-layer caches with reuse metadata, and previous responses used as drafts. Cached representations replace selected computations, whereas response drafts require target-model verification before acceptance. The updated state $\mathcal{C}_t$ retains reusable computation and incorporates newly computed states and the current response.

GUIFlow organizes incremental execution around three sources of temporal redundancy, as illustrated in Figure 2. First, historical screenshots recur across overlapping context windows, motivating reuse of their cached KV states with positional alignment as the window advances. Second, successive observations share persistent interface regions, motivating selective refresh of visual and language states based on observed changes and representation deviations. Finally, action responses within a trajectory often share structural patterns and token sequences, motivating historical drafts whose candidate tokens are verified by the target model and corrected at mismatches. Unavailable or incompatible caches trigger initialization or recomputation. Together, these mechanisms extend incremental execution across visual encoding, prefill, and decoding without changing model-visible

inputs or the action interface. Intermediate-state reuse can be approximate, and draft verification under the resulting prefill state does not imply equivalence to dense inference.

3.2 Incremental State Integration for Historical Context

We first address redundant computation in historical context, focusing on the KV states of screenshots retained across interaction steps. As the context window advances, these screenshots may shift in position, requiring a mechanism beyond unchanged-prefix matching.

Position-Aligned Historical KV Injection. The principle is to populate the current context with position-aligned historical KV states and compute the remaining regions, without adding tokens or changing the model-visible input. GUIFlow maps eligible historical image blocks in X_t to their cached counterparts in $\mathcal{C}_{t-1}$. It first identifies a contiguous matching prefix using multimodal fingerprints: text tokens are identified by token ID and type, while image tokens are identified by image content, grid layout, and within-image offset. Beyond this prefix, historical image blocks are matched by content, grid, and token count. This mapping separates blocks eligible for cache injection from regions requiring computation, including intervening text and action tokens outside the matching prefix and historical images that cannot be reused.

For each matched historical token, GUIFlow transfers the cached KV states to the corresponding current position. Let p_s and p_t denote its source and current multimodal positions, and let $R(p)$ denote the corresponding M-RoPE rotation. At layer ℓ, the position-aligned states are

$$\widetilde{K}_t^{(\ell)} = R(p_t)R(p_s)^\top K_s^{(\ell)}, \qquad \widetilde{V}_t^{(\ell)} = V_s^{(\ell)}, \tag{2}$$

where $K_s^{(\ell)}$ already carries the source positional rotation; the correction acts on rotary dimensions and leaves other dimensions unchanged. These states populate their slots in the current logical KV layout, allowing recomputed queries to attend to both reused and newly computed states under the current causal mask.

Finally, GUIFlow uses a positive integer hyperparameter M_{hist} to cap the number of sliding reuses of historical-image KV states, preventing indefinite reuse without recomputation. Each sliding copy increments the inherited per-token reuse count, whereas exact-prefix reuse leaves it unchanged. Before a historical image block is reused, its maximum source-token count is checked against M_{hist}; if the count is at least M_{hist}, the block is recomputed and its counts are reset to zero.

3.3 Incremental State Refinement for Current Observations

The current observation captures the latest environment state and directly informs the next action, while sharing persistent interface regions and potentially reusable internal states with preceding observations. GUIFlow therefore selectively refreshes current-observation computation at two levels, using pixel changes to guide visual-state updates and context-dependent representation deviations to guide deeper language-model updates.

Change-Guided Visual State Propagation. GUIFlow propagates existing visual states while concentrating new computation on changed regions. Patch-level pixel differences against a rolling reference identify an update set $\mathcal{A}_t$, expanded to include complete patch-merging groups. At vision-encoder layer ℓ, the KV states are updated as

$$Z_{t,i}^{(\ell)} = \begin{cases} \widehat{Z}_{t,i}^{(\ell)}, & i \in \mathcal{A}_t, \\ Z_{\text{ref},i}^{(\ell)}, & i \notin \mathcal{A}_t, \end{cases} \qquad Z \in \{K, V\}, \tag{3}$$

where $\widehat{Z}_{t,i}^{(\ell)}$ denotes a newly computed state and $Z_{\text{ref},i}^{(\ell)}$ its cached counterpart in the rolling reference. Selected patch queries attend to this full mixture of new and cached visual KV and undergo feed-forward computation; unselected patches retain their cached states. Updated states become the next reference; incompatible layouts, extensive changes, or reaching the sparse-update limit trigger full recomputation.

Source-Aware TokenLease. TokenLease probes context-dependent representation changes before deciding which deep states to refresh. Current-image tokens pass through the first L_p language-model

Table 1: **Main results on offline multi-image navigation.** FLOPs (T) are averaged per step; speedup is the Dense/method FLOPs ratio, with an equal-weight geometric mean across datasets. GUIFlow achieves 3.37–3.39× geometric-mean speedups while maintaining scores close to Dense.

Model	Method	AITW			Mind2Web			Geo. Mean (×) ↑
		Action Matching (%) ↑	FLOPs (T) ↓	Speedup (×) ↑	Step SR (%) ↑	FLOPs (T) ↓	Speedup (×) ↑	
Qwen3-VL 4B-Instruct	Dense	61.20	31.94	1.00	49.77	43.97	1.00	1.00
	FastV	60.26	22.14	1.44	48.31	30.10	1.46	1.45
	VisionZip	60.23	18.91	1.69	42.09	24.90	1.77	1.73
	GUI-KV	58.59	31.94	1.00	49.25	43.96	1.00	1.00
	STaR-KV	60.34	31.93	1.00	49.61	43.95	1.00	1.00
	Op-Skip	58.29	24.65	1.30	**53.41**	34.21	1.29	1.29
	HistPrune	61.20	25.47	1.25	49.75	34.67	1.27	1.26
	GUIFlow	**61.74**	**9.23**	**3.46**	49.16	**13.37**	**3.29**	**3.37**
UI-TARS 1.5-7B	Dense	69.06	62.45	1.00	**53.40**	85.78	1.00	1.00
	FastV	67.99	41.27	1.51	52.40	55.72	1.54	1.53
	VisionZip	66.11	42.01	1.49	49.31	57.54	1.49	1.49
	GUI-KV	**69.12**	62.45	1.00	52.17	85.78	1.00	1.00
	STaR-KV	69.10	62.44	1.00	53.05	85.77	1.00	1.00
	Op-Skip	65.17	50.30	1.24	49.49	69.20	1.24	1.24
	HistPrune	69.06	52.30	1.19	53.29	71.58	1.20	1.20
	GUIFlow	68.93	**18.33**	**3.41**	53.13	**25.49**	**3.37**	**3.39**

layers, and their probe KV is compared with the position-aligned probe KV of their actual deep-cache sources, not necessarily the previous step. We quantify this deviation and its residual mass as

$$d_i = \max_{Z \in \{K,V\}} \frac{\mathrm{RMS}(Z_i^{\mathrm{cur}} - \bar{Z}_i^{\mathrm{src}})}{\max\big(\mathrm{RMS}(\bar{Z}_i^{\mathrm{src}}), \delta\big)}, \qquad m_i = d_i^2, \tag{4}$$

where Z_i^{cur} and $\bar{Z}_i^{\mathrm{src}}$ denote the current and position-aligned source probe states at layer L_p, respectively, and $\delta > 0$ stabilizes the normalization. The refresh set $\mathcal{R}_t$ includes pixel-changed tokens and tokens reaching the reuse limit M_{cur}, then expands in descending residual mass until

$$\sum_{i \in \mathcal{I}_t \setminus \mathcal{R}_t} m_i \le \varepsilon \sum_{i \in \mathcal{I}_t} m_i, \tag{5}$$

where $\mathcal{I}_t$ indexes current-image tokens and $\varepsilon \in [0, 1]$ sets the unrefreshed residual-mass budget. Beyond layer L_p, selected image queries are computed and others reuse deep KV; text queries and the full logical KV layout are retained. Refreshed tokens renew their sources and reset reuse counts; reused tokens retain their sources and increment counts.

3.4 Verified Incremental Decoding for Action Generation

We next accelerate autoregressive decoding by exploiting temporal redundancy in action responses. Outputs at successive interaction steps often share structural patterns and token sequences, although action arguments may change with the current observation. Rather than regenerating these sequences token by token, GUIFlow uses historical outputs to propose continuations and verifies them under the current context.

Trajectory-Guided Speculative Decoding. Following the draft–verify principle of speculative decoding (Leviathan et al., 2023), GUIFlow treats the action-response portion of an earlier output, such as y_{t-1}, as a draft for y_t. The candidate pool retains the most recent response for each action type within the same task, allowing earlier steps to supply alternatives when the immediately preceding action differs. These drafts require neither a separate model nor additional training and are not appended to the model-visible prompt as extra history. Instead, their candidate tokens are evaluated in a batched target-model forward pass from the current prefill state and the accepted response prefix.

Table 2: **Main results on online OSWorld interaction.** FLOPs (T) are averaged per step/request for each evaluated run; speedups are Dense/method mean FLOPs ratios across rollout trajectories. GUIFlow achieves 3.32–3.49× speedups relative to Dense.

Model	Method	Mean Score (%) ↑	Strict SR (%) ↑	FLOPs (T) ↓	Speedup (×) ↑
Qwen3-VL 8B-Instruct	Dense	24.30	23.55	247.01	1.00
	FastV	23.16	22.16	165.50	1.49
	VisionZip	21.22	20.22	161.48	1.53
	GUI-KV	22.81	21.61	246.54	1.00
	HistPrune	20.82	19.94	194.52	1.27
	DivPrune	22.93	22.71	185.89	1.33
	GUIFlow	**27.90**	**26.87**	**70.85**	**3.49**
GUI-Owl 1.5-8B-Instruct	Dense	**48.79**	**47.09**	278.12	1.00
	FastV	45.83	44.32	187.36	1.48
	VisionZip	44.36	42.38	183.18	1.52
	HistPrune	45.74	43.77	219.07	1.27
	DivPrune	41.79	40.17	182.84	1.52
	GUIFlow	46.30	44.60	**83.81**	**3.32**

For greedy decoding, let u denote the accepted prefix and $d^{(j)}$ a candidate continuation of length n_j. Writing $p_{\theta,t}$ for the target-model distribution conditioned on the current prefill state, we compute

$$\begin{aligned} \widehat{d}_i^{(j)} &= \arg\max_v p_{\theta,t}(v \mid u, d_{<i}^{(j)}), \\ a_j &= \max\{k \in \{0,\dots,n_j\} : \; d_i^{(j)} = \widehat{d}_i^{(j)} \text{ for all } 1 \le i \le k\}. \end{aligned} \tag{6}$$

GUIFlow selects the longest matching prefix, favoring more recent drafts in a tie. It retains the accepted tokens and their newly computed KV, discards rejected candidate states, and generates a target-model correction at the first mismatch before continuing decoding. When applicable, later structural matches allow re-entry into a historical draft; otherwise, generation proceeds token by token. Matching drafts thus let one verification pass advance multiple output tokens, reducing serial decoding calls while leaving the current token decisions to the target model.

4 Experiments

4.1 Main Results

Offline Multi-Image Navigation. Many multimodal baselines optimize individual requests, while GUI-specific alternatives exploit temporal structure through context compression or historical caching. Repeated screenshot histories and locally changing interfaces offer a broader opportunity: advancing a decision need not require recomputing all supporting representations. Table 1 shows that GUIFlow achieves the lowest FLOPs across both models and datasets, with geometric-mean speedups of 3.37× on Qwen3-VL and 3.39× on UI-TARS while maintaining task scores close to Dense. Across the four model–dataset settings, GUIFlow delivers 3.29–3.46× speedups, compared with at most 1.77× among the evaluated baselines.

Online Closed-Loop Interaction. We complement offline action prediction on fixed trajectories with closed-loop evaluation in OSWorld, where actions alter subsequent observations and task success is evaluated from the resulting environment state. This coupling requires cached computation to support decisions as the trajectory evolves. GUIFlow achieves 3.49× and 3.32× speedups on Qwen3-VL and GUI-Owl, respectively, with higher mean scores and strict success rates than the evaluated acceleration baselines (Table 2). On Qwen3-VL, GUIFlow reaches 27.90% mean score and 26.87% strict success rate, compared with 24.30% and 23.55% for Dense. These findings extend the evidence for incremental inference from fixed histories to interactive execution; we examine wall-clock benefits in the following analysis.

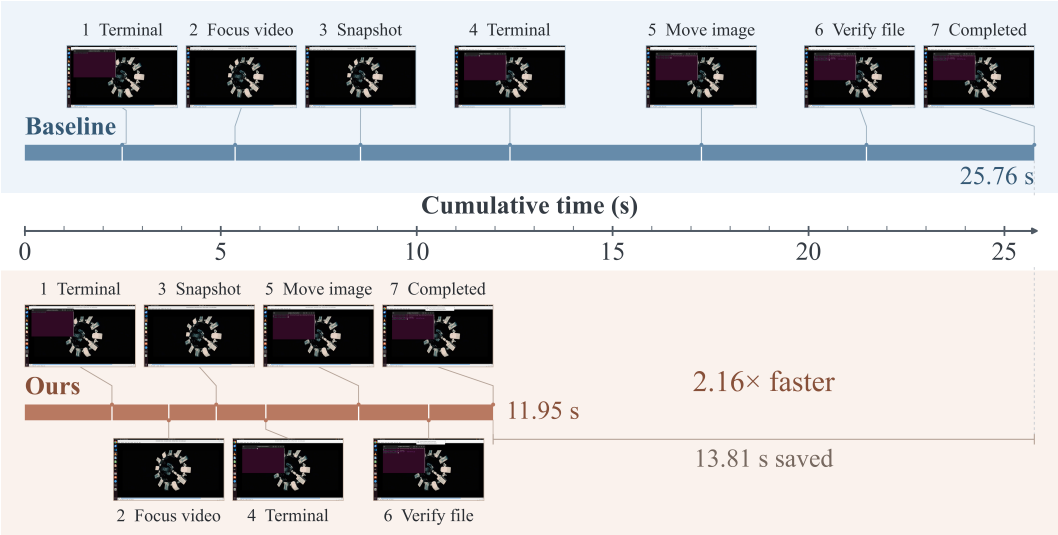


Figure 3: **Qualitative comparison of task execution time on an OSWorld task.** Dense inference and GUIFlow complete the same seven interaction stages. GUIFlow reduces task execution time from 25.76 to 11.95 seconds, corresponding to a 2.16× speedup and 13.81 seconds saved. The horizontal axis measures cumulative task execution time.

Table 3: **FLOPs and wall-clock efficiency of GUIFlow.** Total latency covers per-step processing and inference, excluding environment execution. Speedups use Dense/GUIFlow ratios of unrounded means; online records come from different trajectories.

Benchmark	Model	FLOPs (T) ↓	Total Latency (ms) ↓	Speedup	
				FLOPs (×) ↑	Latency (×) ↑
AITW	Qwen3-VL-4B	9.23	470.64	**3.46**	**1.96**
	UI-TARS-1.5-7B	18.33	547.23	**3.41**	**1.94**
Mind2Web	Qwen3-VL-4B	13.37	633.91	**3.29**	**1.97**
	UI-TARS-1.5-7B	25.49	603.08	**3.37**	**2.32**
OSWorld	Qwen3-VL-8B	70.85	1502.69	**3.49**	**2.51**
	GUI-Owl-1.5-8B	83.81	1713.34	**3.32**	**2.55**

4.2 Analysis

Speedup and Latency Reduction. Table 3 reports 3.29–3.49× speedups across the evaluated offline and online settings, alongside reductions in wall-clock latency by factors of 1.94–2.55. For example, GUI-Owl's mean processing latency on OSWorld decreases from 4362.91 to 1713.34 ms. Processing latency also reflects memory access, cache management, and input/output costs, so its reduction differs from the FLOPs reduction.

Component Contributions. Table 4 shows complementary benefits across the cumulative configurations. Historical KV injection provides the largest initial FLOPs reduction, with sparse visual updates and TokenLease bringing further savings. With the same TokenLease settings, speculative decoding slightly increases FLOPs but lowers cumulative server processing time from 27.51 to 17.85 s/task. This is consistent with trading additional verification work for fewer serial decoding calls: state updates reduce repeated computation, while historical drafts improve generation efficiency.

Probe Depth and Reuse Limit. Table 5 characterizes sensitivity to probe depth and reuse limit. In the layer sweep, $L_p = 6$ achieves the highest task scores; shallower probing saves slightly more FLOPs at lower quality, while deeper probing adds cost without improving quality. At $L_p = 6$,

Table 4: **Cumulative component ablation on Qwen3-VL-8B in OSWorld.** Checkmarks indicate enabled components; quality uses 361 tasks, with missing results scored as zero. TokenLease uses $L_p = 6$ and $M_{\text{cur}} = 3$ in both enabled configurations. FLOPs and Total average valid task measurements; Total sums server processing time and excludes environment execution.

Enabled Components				Task Performance		Inference Cost	
History KV	Sparse Vision	Token Lease	Spec. Decode	Mean Score (%) ↑	Strict SR (%) ↑	FLOPs (T/task) ↓	Total (s/task) ↓
–	–	–	–	24.30	23.55	2981.19	45.43
✓	–	–	–	27.33	26.32	1010.29	29.54
✓	✓	–	–	26.21	24.93	962.02	29.35
✓	✓	✓	–	25.43	24.65	**819.07**	27.51
✓	✓	✓	✓	**27.90**	**26.87**	841.33	**17.85**

Table 5: **Sensitivity to TokenLease probe depth and reuse limit on Qwen3-VL-8B in OSWorld.** The layer sweep fixes $M_{\text{cur}} = 2$, and the reuse-limit sweep fixes $L_p = 6$. † marks the shared default value of the swept parameter; quality uses all 361 tasks, with missing results scored as zero. Bold marks the best value within each sweep.

Sweep	Setting	Mean Score (%) ↑	Strict SR (%) ↑	FLOPs (T/step) ↓	Total (s/step) ↓
Probe Depth ($M_{\text{cur}} = 2$)	$L_p = 3$	24.28	23.27	**224.70**	3.59
	$L_p = 6^{\dagger}$	**27.35**	**26.59**	228.22	**3.56**
	$L_p = 9$	26.48	25.48	230.43	3.58
	$L_p = 12$	25.25	24.65	231.01	3.62
Reuse Limit ($L_p = 6$)	$M_{\text{cur}} = 1$	25.93	24.65	226.28	3.57
	$M_{\text{cur}} = 2$	27.35	**26.59**	228.22	**3.56**
	$M_{\text{cur}} = 3^{\dagger}$	**27.60**	**26.59**	224.76	3.57
	$M_{\text{cur}} = 4$	26.76	25.48	**224.33**	3.60

$M_{\text{cur}} = 3$ gives the highest mean score and ties the best strict success rate; extending reuse to four saves only 0.2% additional FLOPs with lower scores. The shared defaults, $L_p = 6$ and $M_{\text{cur}} = 3$, were fixed before the main evaluations; this study analyzes their sensitivity on Qwen3-VL/OSWorld.

Qualitative Visualization. Figure 3 compares cumulative task execution time for Dense inference and GUIFlow on an OSWorld task that both complete successfully. Both executions follow the same seven interaction stages, including terminal operations, focusing the video, taking a snapshot, moving the image, and verifying the resulting file. Dense inference takes 25.76 seconds in total, whereas GUIFlow requires 11.95 seconds, saving 13.81 seconds and yielding a $2.16\times$ speedup in this example. The time gap grows over the rollout, illustrating cumulative time savings across the same seven interaction stages. Appendix A provides split-level Mind2Web results, category-level OSWorld analysis, and TokenLease mechanism statistics, while Appendix B details the benchmarks, models, and evaluation setup. Appendix C further presents eight successful trajectories across diverse applications and task lengths, showing similar reductions in task execution time.

5 Conclusion

We presented GUIFlow, a training-free framework that incrementally updates persistent states across GUI interaction. By reusing position-aligned historical KV states, selectively refreshing current-observation states, and verifying trajectory-derived drafts, GUIFlow reduces repeated computation across visual encoding, prefill, and decoding. It achieves up to $3.39\times$ and $3.49\times$ FLOPs speedups on offline benchmarks and OSWorld, respectively, while lowering processing latency and maintaining task performance broadly comparable to dense inference.

AI USE STATEMENT

In this work, we used generative AI tools solely to assist with drafting, translation, and language refinement of the manuscript. We did not use generative AI tools to design or implement the proposed method, design or conduct experiments, process data, or analyze or interpret results. All AI-assisted text was reviewed and revised by the authors, who verified the technical claims and take full responsibility for the final content of this work.

ETHICS STATEMENT

This work uses existing public benchmarks and does not involve new data collection or human-subject studies. GUI agents may encounter sensitive on-screen information or execute unintended actions, and improving their inference efficiency could increase the scale of both beneficial and harmful use. GUIFlow changes how computation is reused but does not expand an agent's permissions or action space. Practical deployment should therefore retain appropriate access controls, sandboxing, privacy protections, and human oversight for the underlying agent.

REPRODUCIBILITY STATEMENT

The method and state-update rules are described in Section 3, and the main results and ablations are reported in Section 4. The appendix provides benchmark and model details, evaluation metrics, hardware information, and supplementary results. We will release the code on GitHub upon acceptance.

LIMITATIONS

This work focuses on training-free incremental inference for autoregressive VLM-based GUI agents. The efficiency of GUI agents is shaped by several complementary factors, including model architecture, serving systems, and interaction policies, which are not jointly optimized in our study. Exploring how incremental inference can be combined with advances along these dimensions is an important direction for future work.

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.

Saeed Ranjbar Alvar, Gursimran Singh, Mohammad Akbari, and Yong Zhang. Divprune: Diversity-based visual token pruning for large multimodal models. In *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 9392–9401. IEEE, 2025.

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

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

Kanzhi Cheng, Qiushi Sun, Yougang Chu, Fangzhi Xu, Li YanTao, Jianbing Zhang, and Zhiyong Wu. Seeclick: Harnessing gui grounding for advanced visual gui agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 9313–9332, 2024.

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

Yuhang Han, Wenzheng Yang, Yujie Chen, Xiangqi Jin, Yaojie Zhang, Siteng Huang, and Linfeng Zhang. Star-kv: Spatio-temporal adaptive re-weighting for kv cache compression in gui vision-language models. *arXiv preprint arXiv:2606.01790*, 2026.

Zhenyu He, Zexuan Zhong, Tianle Cai, Jason Lee, and Di He. Rest: Retrieval-based speculative decoding. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 1582–1595, 2024.

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.

Kangye Ji, Yuan Meng, Hanyun Cui, Ye Li, Jianbo Zhou, Shengjia Hua, Lei Chen, and Zhi Wang. Block-wise adaptive caching for accelerating diffusion policy. In *International Conference on Learning Representations*, 2026.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In *International conference on machine learning*, pp. 19274–19286. PMLR, 2023.

Ye Li, Jiahe Feng, Yuan Meng, Kangye Ji, Chen Tang, Xinwan Wen, Shutao Xia, Zhi Wang, and Wenwu Zhu. Ts-dp: Reinforcement speculative decoding for temporal adaptive diffusion policy acceleration. *arXiv preprint arXiv:2512.15773*, 2025a.

Ye Li, Yuan Meng, Zewen Sun, Kangye Ji, Chen Tang, Jiajun Fan, Xinzhu Ma, Shutao Xia, Zhi Wang, and Wenwu Zhu. Sp-vla: A joint model scheduling and token pruning approach for vla model acceleration. *arXiv preprint arXiv:2506.12723*, 2025b.

Ye Li, Chen Tang, Yuan Meng, Jiajun Fan, Zenghao Chai, Xinzhu Ma, Zhi Wang, and Wenwu Zhu. Prance: Joint token-optimization and structural channel-pruning for adaptive vit inference. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2025c.

Ye Li, Huanan Liu, Kangye Ji, Yuan Meng, Jiajun Fan, Yuansong Wang, Shiyu Qin, Chenglei Wu, Shu-Tao Xia, and Zhi Wang. Elegantvla: Learning when to think for efficient vision-language-action models. *arXiv preprint arXiv:2605.29438*, 2026.

Yingxin Li, Ye Li, Yuan Meng, Xinzhu Ma, Zihan Geng, Shutao Xia, and Zhi Wang. Ems: Adaptive evict-then-merge strategy for head-wise kv cache compression based on global-local importance. *arXiv preprint arXiv:2412.08521*, 2024.

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 *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 19498–19508. IEEE, 2025.

Zhaoyang Luo, Runmin Dong, Miao Yang, Fan Wei, Yushan Lai, Bin Luo, and Haohuan Fu. Attend, transform, or silence: Operator-level visual skipping for efficient multimodal llm inference. *arXiv preprint arXiv:2606.31903*, 2026.

Yujia Qin, Yining Ye, Junjie Fang, Haoming Wang, Shihao Liang, Shizuo Tian, Junda Zhang, Jiahao Li, Yunxin Li, Shijue Huang, et al. Ui-tars: Pioneering automated gui interaction with native agents. *arXiv preprint arXiv:2501.12326*, 2025.

Christopher Rawles, Alice Li, Daniel Rodriguez, Oriana Riva, and Timothy Lillicrap. Androidinthewild: A large-scale dataset for android device control. *Advances in Neural Information Processing Systems*, 36:59708–59728, 2023.

Yuhao Wang, Mu Qiao, Xindong Zhang, Yunzhi Zhuge, Lei Zhang, and Huchuan Lu. Trace: Trajectory-robust admission with evidence ordering for efficient gui agents. *arXiv preprint arXiv:2609.10297*, 2026.

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

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

Siyu Xu, Yunke Wang, Chenghao Xia, Dihao Zhu, Tao Huang, and Chang Xu. Vla-cache: Efficient vision-language-action manipulation via adaptive token caching. *Advances in Neural Information Processing Systems*, 38:164448–164473, 2026b.

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

Jiayi Yao, Hanchen Li, Yuhan Liu, Siddhant Ray, Yihua Cheng, Qizheng Zhang, Kuntai Du, Shan Lu, and Junchen Jiang. Cacheblend: Fast large language model serving for rag with cached knowledge fusion. In *Proceedings of the twentieth European conference on computer systems*, pp. 94–109, 2025.

Bowen Zhou, Zhou Xu, Wanli Li, Jingyu Xiao, and Haoqian Wang. Efficient long-horizon gui agents via training-free kv cache compression. *arXiv preprint arXiv:2603.00188*, 2026.

Table A1: **Split-level results on Mind2Web.** Step SR (%) is macro-averaged within each split, and Avg. is the unweighted mean across the three splits. Speedup is the Dense/method mean FLOPs ratio; bold denotes the best result for each model.

Model	Method	Cross-Task Step SR ↑	Cross-Website Step SR ↑	Cross-Domain Step SR ↑	Avg. Step SR ↑	Speedup (×) ↑
Qwen3-VL 4B-Instruct	Dense	51.11	48.37	49.84	49.77	1.00
	FastV	49.55	46.73	48.64	48.31	1.46
	VisionZip	42.36	41.23	42.69	42.09	1.77
	GUI-KV	51.59	47.06	49.09	49.25	1.00
	STaR-KV	51.40	47.81	49.63	49.61	1.00
	Op-Skip	**54.18**	**52.48**	**53.56**	**53.41**	1.29
	HistPrune	51.09	48.22	49.93	49.75	1.27
	GUIFlow	50.59	47.45	49.45	49.16	**3.29**
UI-TARS 1.5-7B	Dense	55.04	52.19	**52.98**	**53.40**	1.00
	FastV	54.46	51.59	51.14	52.40	1.54
	VisionZip	51.98	48.08	47.87	49.31	1.49
	GUI-KV	53.30	51.99	51.21	52.17	1.00
	STaR-KV	54.51	51.88	52.76	53.05	1.00
	Op-Skip	51.16	47.69	49.62	49.49	1.24
	HistPrune	54.74	**52.22**	52.91	53.29	1.20
	GUIFlow	**55.13**	51.65	52.62	53.13	**3.37**

A Additional Experimental Results

A.1 Mind2Web Split-Level Results

Table A1 expands the aggregate Mind2Web results in Table 1 across the Cross-Task, Cross-Website, and Cross-Domain splits while retaining the corresponding speedups. On Qwen3-VL, GUIFlow achieves a $3.29\times$ speedup with an average Step SR of 49.16%, only 0.61 percentage points below Dense. This provides a substantially stronger efficiency–accuracy trade-off than VisionZip, which reaches a $1.77\times$ speedup but reduces the average Step SR to 42.09%, while the higher-scoring Op-Skip provides only a $1.29\times$ speedup. On UI-TARS, GUIFlow attains the best Cross-Task result and preserves an average Step SR within 0.27 points of Dense while delivering a $3.37\times$ speedup, compared with at most $1.54\times$ among the baselines. Across both backbones, no individual split drops by more than 0.92 percentage points from Dense.

A.2 OSWorld Category-Level Results

Table A2 reports category-level efficiency and quality results for GUI-Owl on OSWorld. Overall, GUIFlow reduces FLOPs per request by 69.9% ($3.32\times$) and total FLOPs per task by 67.7% ($3.10\times$), while keeping the mean score within 2.49 points of Dense. Task-level speedups range from $2.33\times$ to $4.41\times$ across application categories, demonstrating consistent computational savings under varying request counts.

A.3 TokenLease Mechanistic Statistics

Table A3 complements the outcome-level sensitivity results in Table 5 by exposing the internal reuse behavior of TokenLease. Across probe depths, the acceptance rate remains stable at approximately 91.6%, while moving the probe from layer 3 to layer 12 reduces the fraction of reused image tokens from 30.66% to 23.64% and the FLOPs reduction from 9.07% to 6.23%. This indicates that deeper probing primarily reduces the amount of downstream computation available for reuse rather than materially changing the lease decisions. With $L_p = 6$, increasing M_{cur} from 1 to 3 raises image-token reuse from 25.60% to 31.31% and the FLOPs reduction from 8.08% to 8.77%. Extending the limit to 4 yields no further computational benefit and slightly lowers the paired latency speedup. These

Table A2: **Category-level efficiency and quality on OSWorld with GUI-Owl-1.5-8B.** Request and score changes are measured relative to Dense; FLOPs speedups are computed as Dense/GUIFlow. Overall covers all 361 tasks; efficiency uses valid measurements.

Application	**Tasks**	**Request change (%)**	**FLOPs/request speedup (×)**	**Task FLOPs speedup (×)**	**Score change (pp)**
Chrome	46	+40.4	3.27	2.33	−0.01
GIMP	26	+14.2	3.44	3.01	−3.85
LibreOffice Calc	47	−5.6	3.23	3.42	−6.38
LibreOffice Impress	47	−16.8	3.67	4.41	+4.26
LibreOffice Writer	23	+45.5	3.45	2.37	−8.70
Multi-Apps	93	+5.7	3.19	3.02	−1.08
OS	24	+16.8	3.33	2.86	0.00
Thunderbird	15	−15.0	3.64	4.28	−6.67
VLC	17	+8.8	3.54	3.25	−11.76
VS Code	23	−6.9	3.53	3.80	−4.35
Overall	**361**	+7.1	**3.32**	**3.10**	−2.49

Table A3: **Mechanistic statistics of TokenLease on Qwen3-VL-8B in OSWorld.** The probe-depth sweep fixes $M_{\text{cur}} = 2$, and the reuse-limit sweep fixes $L_p = 6$. FLOPs reduction and latency speedup are paired against Dense on common task-step keys; † marks the shared default value of the swept parameter.

Sweep	**Setting**	Effective Tasks	Lease Acceptance (%)	Reused Image Tokens (%)	FLOPs Reduction (%)	Latency Speedup (×)
Probe Depth ($M_{\text{cur}} = 2$)	$L_p = 3$	359	91.66	30.66	9.07	1.05
	$L_p = 6^{\dagger}$	359	91.59	29.95	7.53	1.06
	$L_p = 9$	359	91.66	25.46	6.73	1.05
	$L_p = 12$	359	91.56	23.64	6.23	1.04
Reuse Limit ($L_p = 6$)	$M_{\text{cur}} = 1$	359	91.54	25.60	8.08	1.05
	$M_{\text{cur}} = 2$	359	91.59	29.95	7.53	1.06
	$M_{\text{cur}} = 3^{\dagger}$	358	91.58	31.31	8.77	1.05
	$M_{\text{cur}} = 4$	359	91.43	30.86	8.73	1.04

statistics complement Table 5 by characterizing reuse behavior around the shared defaults, $L_p = 6$ and $M_{\text{cur}} = 3$.

B EXPERIMENTAL SETUP

Benchmarks and Metrics. We evaluate offline navigation on AITW (Rawles et al., 2023) and the screenshot-based Mind2Web benchmark (Deng et al., 2023; Cheng et al., 2024). AITW contains five subsets—General, Install, GoogleApps, Single, and WebShopping—comprising 4,694 evaluated steps in our setup. We follow the official action-matching metric and report the macro-average over the five subsets; action types must match, taps are evaluated against the annotated target region, and swipes must agree along the primary direction. For Mind2Web, we evaluate 9,268 runnable steps from 1,341 tasks under teacher-forced offline navigation, providing up to four preceding screenshot–action pairs together with the current screenshot. A step is successful when the predicted point lies within the ground-truth element and the predicted operation matches the reference; we macro-average within Cross-Task, Cross-Website, and Cross-Domain and report their unweighted mean. For online evaluation, we use all 361 tasks in OSWorld's `test_nogdrive` split (Xie et al., 2024). The agent interacts with the environment in closed loop, and task quality is measured by the environment-specific final-state evaluator. Mean Score and strict success rate use all 361 tasks as the denominator; only an evaluator score of one counts as strict success. Missing or incomplete results, including tasks interrupted by simulator crashes, are assigned zero scores and counted as failures.

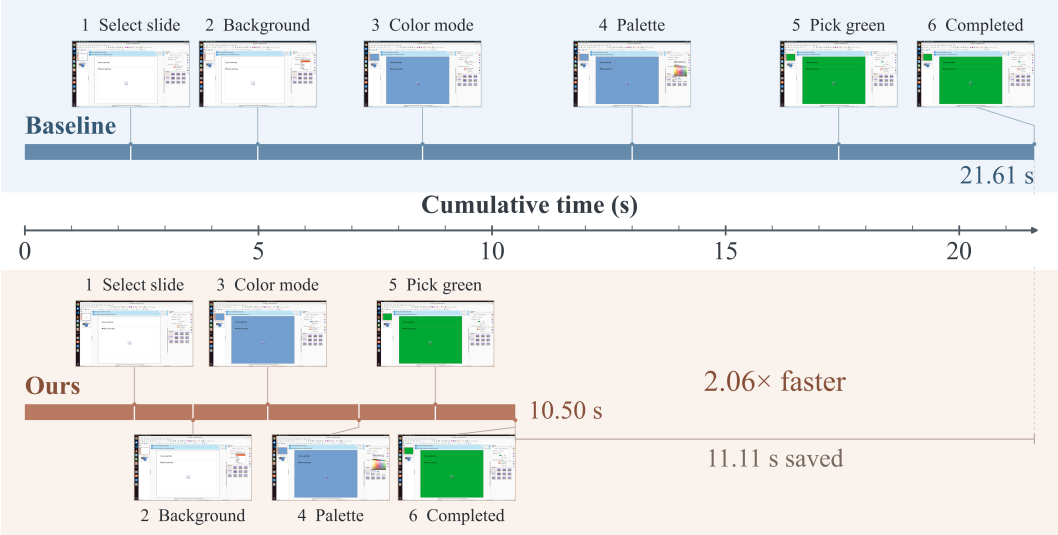
Models. Our evaluation covers both general-purpose vision-language models and GUI-specialized agents. For offline navigation, we use Qwen3-VL-4B-Instruct (Bai et al., 2025) and UI-TARS-1.5-7B, from the UI-TARS family (Qin et al., 2025). For online OSWorld interaction, we use Qwen3-VL-8B-Instruct and GUI-Owl-1.5-8B-Instruct (Xu et al., 2026a). This selection evaluates GUIFlow across two model scales and across both general multimodal and GUI-native model families. GUIFlow is applied at inference time without updating model parameters, and each model retains its native processor, prompt structure, and action representation; UI-TARS uses its `start_box` action protocol in the offline evaluation.

Hyperparameter Configuration. We determined a shared hyperparameter configuration through preliminary observations on sampled offline tasks before the main evaluations. The configuration, including $L_p = 6$ and $M_{\text{cur}} = 3$, was then fixed across all evaluated models and both offline and online benchmarks, except for the explicitly varied settings in the ablation and sensitivity studies. The OSWorld sensitivity study analyzes parameter effects and was not used to select the defaults.

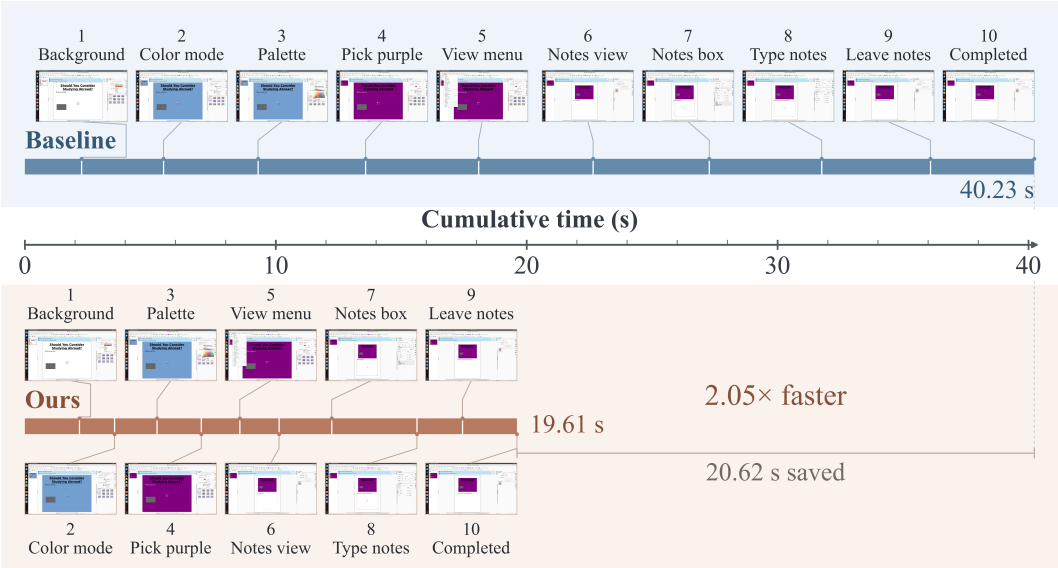
Baselines and Efficiency Measurement. We compare against Dense inference, visual-token pruning, KV compression, operator skipping, and historical caching baselines. All experiments are conducted on NVIDIA GeForce RTX 4090 GPUs. We measure efficiency using average VLM FLOPs and processing latency, and report improvements relative to Dense inference. Efficiency statistics retain valid measurements from successful and failed tasks; missing measurements from simulator crashes are excluded rather than imputed as zeros.

C Additional Qualitative Visualizations

Figures 4–7 provide eight additional successful OSWorld trajectories spanning presentation editing, file operations, document editing, image manipulation, email organization, system configuration, and code editing. In every example, Dense inference and GUIFlow follow the same interaction stages and reach the completed state, while GUIFlow consistently accumulates less task execution time. Across these trajectories, the observed speedups range from 2.01× to 2.24×, with 8.97–22.61 seconds saved per task. These trajectories complement Figure 3 by illustrating lower cumulative task execution time across applications and trajectory lengths. All timelines report task execution time.

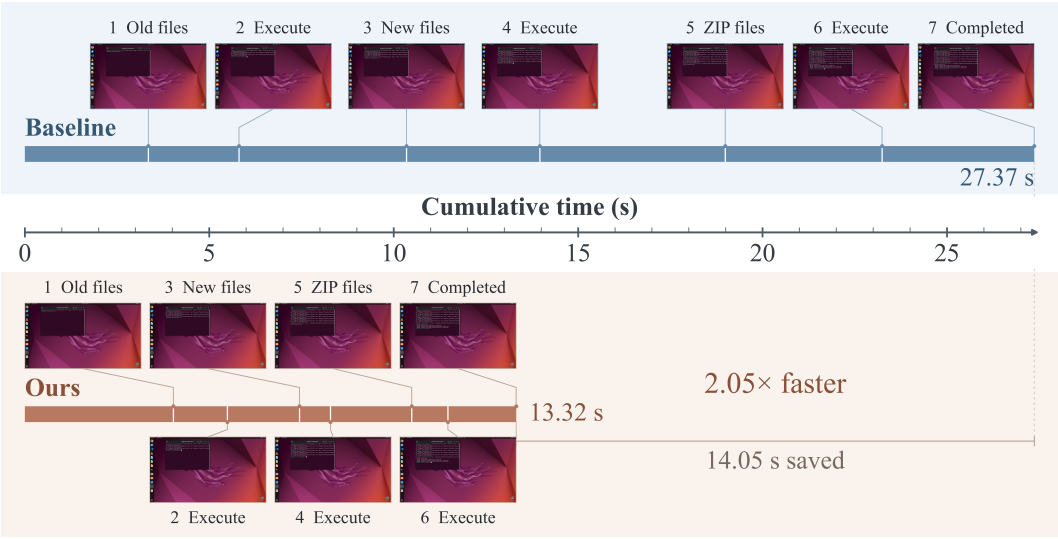


(a) Changing a slide background to green: 21.61 to 10.50 s (2.06×).

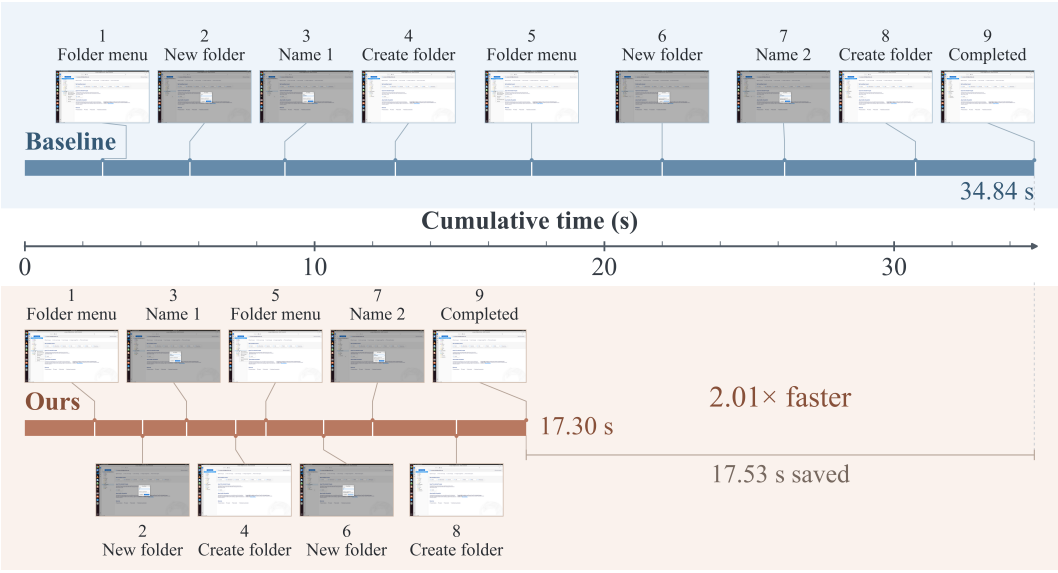


(b) Editing slide color and speaker notes: 40.23 to 19.61 s (2.05×).

Figure 4: **Additional presentation-editing trajectories.** GUIFlow preserves all interaction stages while reducing cumulative task execution time by 11.11 and 20.62 seconds, respectively.

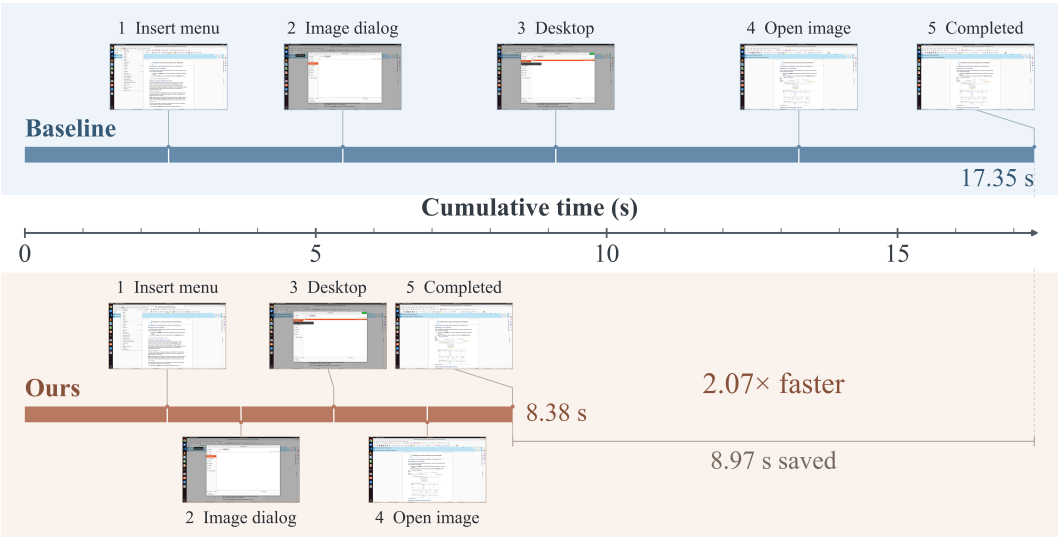


(a) Sorting and archiving files: 27.37 to 13.32 s (2.05×).

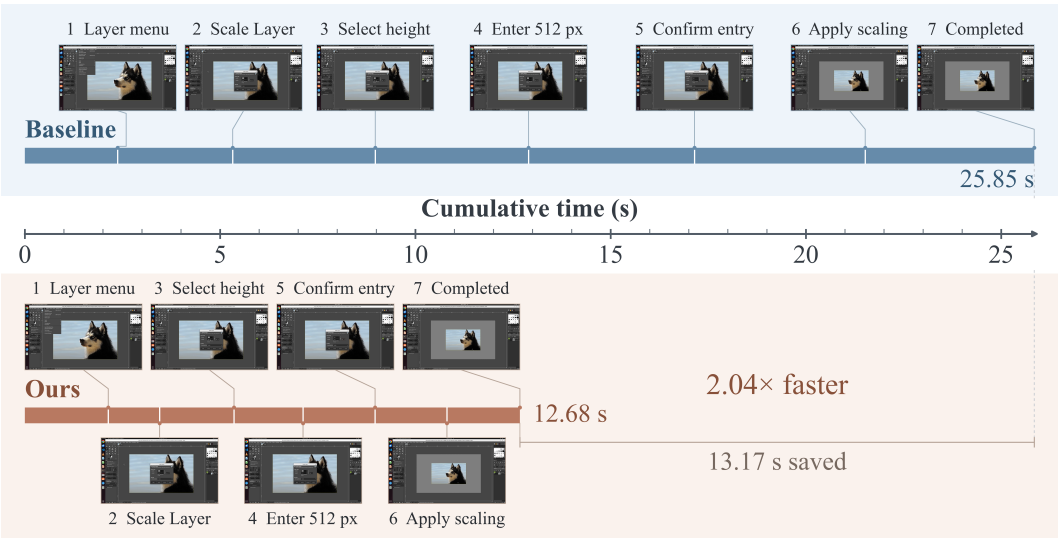


(b) Creating two mail folders: 34.84 to 17.30 s (2.01×).

Figure 5: **Additional file- and email-organization trajectories.** GUIFlow saves 14.05 and 17.53 seconds while retaining the completed task sequence.

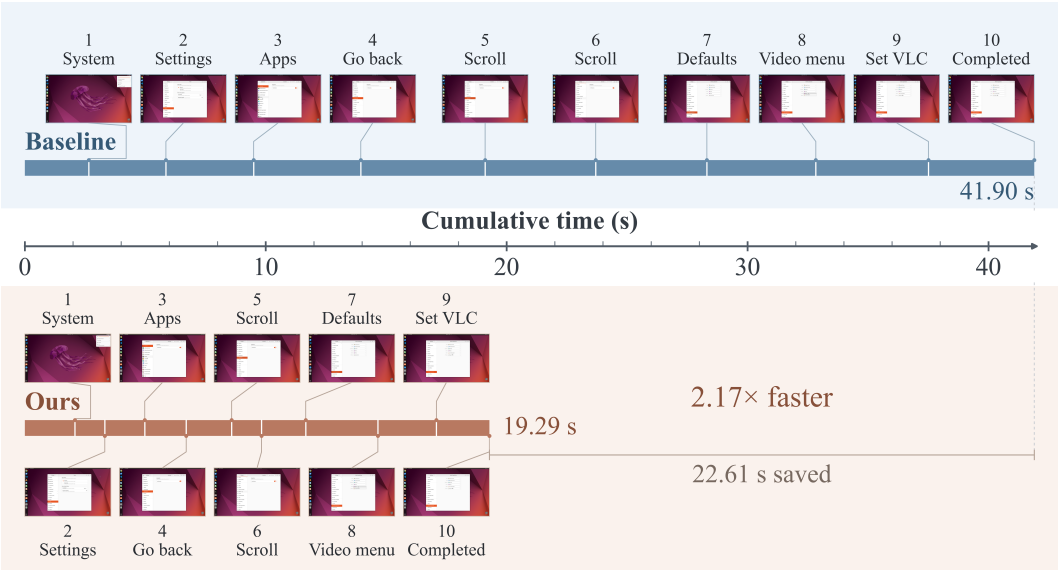


(a) Inserting an image in Writer: 17.35 to 8.38 s (2.07×).

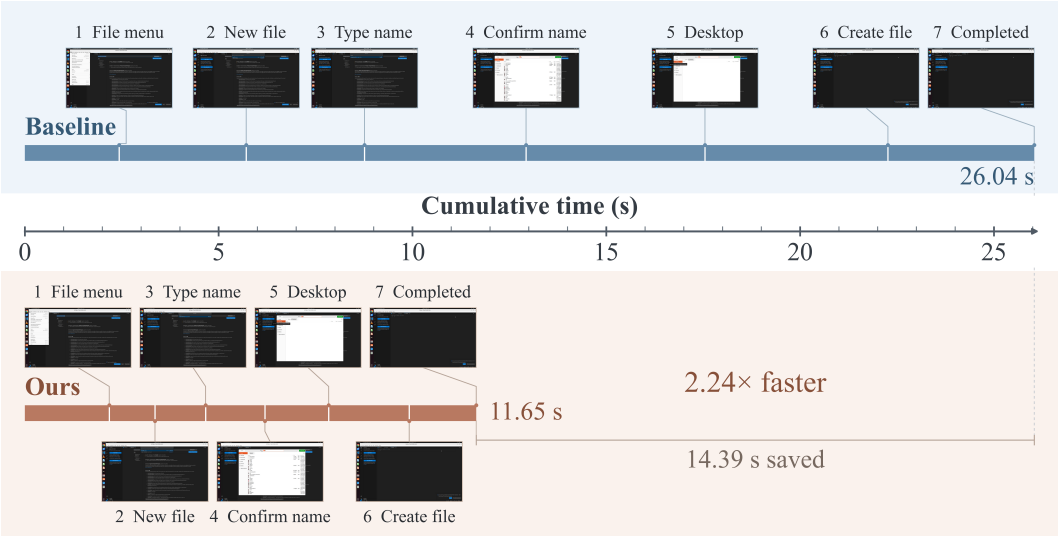


(b) Resizing an image in GIMP: 25.85 to 12.68 s (2.04×).

Figure 6: **Additional document- and image-editing trajectories.** GUIFlow saves 8.97 and 13.17 seconds while completing the same editing stages as Dense inference.



(a) Setting VLC as the default application: 41.90 to 19.29 s (2.17×).



(b) Creating a Python file in VS Code: 26.04 to 11.65 s (2.24×).

Figure 7: **Additional system-configuration and code-editing trajectories.** GUIFlow saves 22.61 and 14.39 seconds, respectively, while preserving all interaction stages.