ST-Lite: Training-Free KV Cache Compression with Spatio-Trajectory Guidance for Long-Horizon GUI Agents

Bowen Zhou[1*] **Zhou Xu**[1*] **Wanli Li**[2*] **Jingyu Xiao**[3]
Pingan Gan[1] **Haoqian Wang**[1†]
[1]Tsinghua University, Shenzhen, China
[2]Zhejiang University, Hangzhou, China
[3]The Chinese University of Hong Kong, Hong Kong, China
zhoubw25@mails.tsinghua.edu.cn
xu-z25@mails.tsinghua.edu.cn
wanli_li@zju.edu.cn

Abstract

Training-free KV cache compression is essential for deploying vision-language GUI agents under memory and latency constraints, yet existing methods are designed for generic language workloads and ignore the distinctive structure of GUI interaction traces. We characterize three GUI-specific workload properties—high inter-frame visual redundancy, extremely small UI-element spatial footprints, and near-uniform cross-layer attention sparsity—that cause existing schemes to retain as few as 39% of oracle-important KV pairs at the standard 20% budget. To address this, we propose **ST-Lite**, a training-free compression scheme whose three components each target one property: **Trajectory-aware Semantic Gating (TSG)** filters redundant historical frames, **Component-centric Spatial Saliency (CSS)** preserves fine-grained element boundaries, and a flat per-layer budget avoids hierarchical misallocation. Across seven GUI benchmarks and two backbones in the deployment-relevant $\beta \in [10\%, 40\%]$ window, ST-Lite consistently outperforms all existing compression baselines—matching or exceeding Full Cache task accuracy on the primary backbone at the 20% budget—while delivering up to 2.35× decoding speedup at fivefold compression. The implementation is available at https://github.com/94wen94/ST-Lite.

1 Introduction

Vision-language models have catalyzed autonomous Graphical User Interface (GUI) agents that navigate digital environments and execute multi-step workflows (Chae et al., 2025; Christianos et al., 2023; Patel et al., 2024; Putta et al., 2024; Qi et al., 2025; Wen et al., 2024; Zhang et al., 2025). For example, WeaveBench evaluates long-horizon hybrid-interface workflows that require GUI control and CLI/code operations within a single trajectory (Li et al., 2026). A central obstacle to their real-time deployment is the Key-Value (KV) cache. High-resolution screenshots combined with long action histories cause the cache to grow linearly with sequence length, routinely reaching multi-gigabyte scale within fifty steps and dominating both GPU memory and decoding latency on memory-constrained accelerators.

Existing training-free KV cache compression methods inherit assumptions from text and general multimodal settings, and these assumptions do not match the GUI agent regime. We make this concrete by adopting the Cache Hit Rate (CHR) of VL-Cache (Tu et al., 2025) (Definition 3.1), which quantifies the fraction of oracle-important KV pairs that a compression scheme retains (§A). On AITW (Rawles et al., 2023) under UI-TARS-1.5-7B (Qin et al., 2025) at the canonical 20% budget, SnapKV (Li et al., 2024b), PyramidKV (Cai et al., 2024) and VL-Cache (Tu et al., 2025) attain CHR of only 53.1%, 40.4% and 39.0% respectively (Table 3, Figure 6). Two failure modes account for the gap (Figure 1): a local optimum trap, in which window-based scoring favors recent attention and discards long-history dependencies required for the next action; and a structure misalignment, in which hierarchical allocation concentrates the cache budget on shallow layers while GUI workloads exhibit near-uniform sparsity across all layers, so deep-layer decision-relevant tokens are systematically lost.

We therefore characterize the statistical properties of GUI agent KV caches directly. Three properties jointly distinguish this regime from the settings that motivated prior compression methods. First, GUI trajectories exhibit substantial temporal redundancy: on 311 consecutive frame pairs from AgentNetBench (Wang et al., 2025b), the median pair-

[*]Equal contribution.
[†]Corresponding author.

Findings of the Association for Computational Linguistics: EMNLP 2026, pages 9550–9586
October 24-29, 2026 ©2026 Association for Computational Linguistics

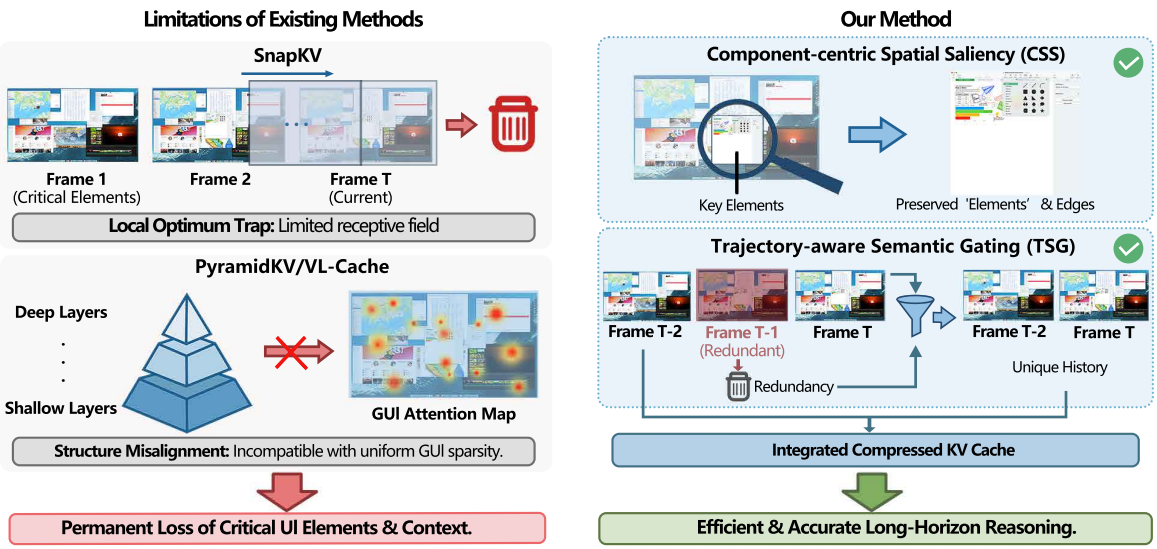


Figure 1: Conceptual illustration of ST-Lite compared with existing methods. The left side depicts the limitations of current strategies: window-based greedy methods (e.g., SnapKV) suffer from **local optimum traps**, while hierarchical allocation methods (e.g., PyramidKV) lead to **structure misalignment**. The right side demonstrates our ST-Lite, which integrates **Component-centric Spatial Saliency (CSS)** to preserve key interactive elements and **Trajectory-aware Semantic Gating (TSG)** to eliminate history redundancy, achieving efficient and accurate long-horizon reasoning.

wise pixel overlap is 82.7%, and more than three quarters of pairs exceed 50% overlap (Figure 2, §B). Second, decision-relevant content is spatially localized: on ScreenSpot Pro (Li et al., 2025), the median UI element occupies only $1.7\% \times 1.6\%$ of the screen (§C). Third, attention sparsity is nearly uniform across layers: the coefficient of variation across the 28 transformer layers is at most 1.4% on both AITW and AgentNetBench (measured on OpenCUA-7B (Wang et al., 2025b), Table 8; visually consistent on UI-TARS-1.5-7B, Figure 3, §D), in clear contrast to the pyramid-shaped (top-heavy) profiles that motivate hierarchical allocation in text and general vision-language settings.

Guided by these three properties, we propose **ST-Lite** (Spatio-Trajectory Lite), a training-free KV cache compression scheme whose three design choices are each motivated by one of the three measured properties and directly target the two failure modes above (Figure 1). Trajectory-aware Semantic Gating (TSG) exploits temporal redundancy by filtering historical KV tokens already covered by the current frame, addressing the local optimum trap. Component-centric Spatial Saliency (CSS) exploits spatial localization by selecting tokens that form structural boundaries of small UI elements within a 3×3 Moore neighborhood. A uniform per-layer budget replaces hierarchical allocation and removes the structure-misalignment source.

We evaluate ST-Lite on seven GUI benchmarks on UI-TARS-1.5-7B and OpenCUA-7B. At the 20% budget on ScreenSpot Pro (Li et al., 2025), ST-Lite reaches 42.2% step accuracy and matches the uncompressed Full Cache baseline (42.3%). On long-horizon AITW (Rawles et al., 2023), it improves Full Cache from 18.2% to 20.1% at the same budget (Table 38), with up to $2.35\times$ decoding speedup at fivefold compression (§4.5). Under the CHR metric, ST-Lite consistently leads all baselines across budgets (Table 3).

Our contributions are threefold. First, we identify two failure modes (recency-bias eviction and hierarchical misallocation) grounded in three measured workload properties (P1–P3) that jointly distinguish GUI agent caches from prior settings. Second, we show that the minimal operators satisfying the constraints imposed by P1–P3—TSG, CSS, and flat allocation—tightly recover performance; wider kernels, alternative metrics, and hierarchical budgets all degrade (§O.3, §O.2, Table 14). Third, on seven GUI benchmarks ST-Lite delivers up to $2.35\times$ decoding speedup at fivefold compression while consistently outperforming all existing compression baselines—matching or exceeding Full

Cache on the primary backbone and leading under the VL-Cache CHR metric at every tested budget.

2 Background

2.1 Problem Setup and KV Compression

We formulate GUI navigation as a long-horizon autoregressive generation problem. Let π_θ denote a vision-language model agent. At step t, given a user instruction I and a high-resolution screenshot $o_t \in \Omega$ (Nguyen et al., 2025), the agent generates an action $a_t \in \mathcal{A}$ conditioned on the cumulative history H_t (Wang et al., 2024c), with

$$H_t = \{(o_0, a_0), (o_1, a_1), \ldots, (o_{t-1}, a_{t-1})\}, \quad (1)$$

and the inference objective $a_t = \operatorname{argmax}_{a \in \mathcal{A}} P(a \mid I, o_t, H_t; \theta)$. Strict adherence is computationally prohibitive because the visual token explosion in high-resolution GUI streams inflates the KV cache linearly in t.

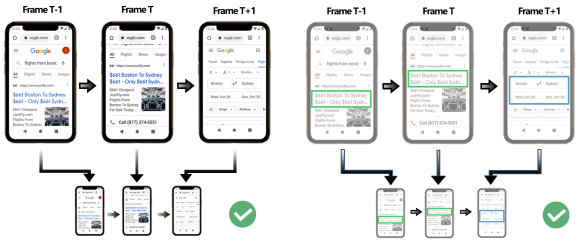
Efficient inference relies on the Key-Value cache, with a prefill phase that encodes multimodal inputs in parallel into initial KV states and a decoding phase that auto-regressively generates action tokens (Tu et al., 2025). Let $K^{(l)}, V^{(l)} \in \mathbb{R}^{L \times D}$ denote the cached Key and Value matrices at layer l for sequence length L. Cache compression derives a retained index set $\mathcal{I}_{keep}$ with $|\mathcal{I}_{keep}| = B \ll L$ that minimizes the approximation error of the attention output. Two design axes span the existing solution space: **budget allocation**, where uniform schemes assign the same budget to every layer (Li et al., 2024b; Zhang et al., 2023; Xiao et al., 2024) and hierarchical schemes assign larger budgets to shallow layers (Cai et al., 2024; Tu et al., 2025); and **token scoring**, for which we adopt the voting strategy of SnapKV (Li et al., 2024b) as our base attention prior

$$A_{base}^{(i)} = \sum_{q=L-\delta+1}^{L} \operatorname{Softmax}\left(\frac{\mathbf{q}_q \cdot \mathbf{k}_i^\top}{\sqrt{d}}\right), \quad (2)$$

with the retained set $\mathcal{I}_{keep} = \operatorname{Top}_B(\{A_{base}^{(i)}\}_{i=1}^{L})$ and budget $B = \lfloor \beta \cdot L \rfloor$ at compression ratio $\beta \in (0, 1]$.

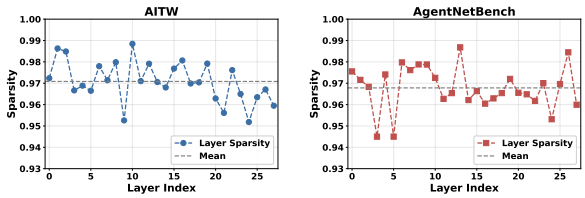
2.2 Cache Hit Rate Evaluation

Compression schemes are typically evaluated by aggregate task accuracy, which does not reveal how faithfully the compressed cache preserves the tokens the model actually relies on. We adopt the



(a) Original Sequence $\mathcal{O}$ (b) Pruned Sequence $\mathcal{O}_r$

Figure 2: Task execution flows under different context settings. (a) The original sequence $\mathcal{O}$ with full historical frames leads to task success. (b) The pruned sequence $\mathcal{O}_r$, with trajectory-wise redundant frames removed, also succeeds, confirming the high historical redundancy in GUI trajectories.



(a) (b)

Figure 3: **Layer-wise attention sparsity.** GUI agents on (a) AITW and (b) AgentNetBench exhibit near-uniform high sparsity across all transformer layers, in contrast to the pyramid-shaped profiles measured on text and general vision-language tasks.

CacheHitRate of VL-Cache (Tu et al., 2025) (Definition 3.1) and extend it to multi-step GUI agents. At each step t, we run inference with the *full*, uncompressed cache and record the attention distribution over all L KV positions when generating the first action token. The **oracle set** $\mathcal{I}^{*(t)}$ is defined as the top-B positions ranked by this attention score, where $B = \lfloor \beta \cdot L \rfloor$ is the cache budget; note that both the oracle set and the compression scheme retain the same number of tokens B, so CHR measures whether they agree on *which* tokens to keep. For any compression scheme Π operating at budget β, let $\mathcal{I}_{keep}^{(t)}(\Pi, \beta)$ be the set of B KV positions it actually retains at step t. The Cache Hit Rate is

$$\mathrm{CHR}(\Pi, \beta) = \frac{1}{T} \sum_{t=1}^{T} \frac{|\mathcal{I}_{keep}^{(t)}(\Pi, \beta) \cap \mathcal{I}^{*(t)}|}{|\mathcal{I}^{*(t)}|}, \quad (3)$$

that is, the fraction of oracle-important tokens retained by the scheme, averaged across all T evaluation steps.[1] On AITW under UI-TARS-1.5-7B at the canonical $\beta = 20\%$, SnapKV (Li et al., 2024b)

[1] A scheme that perfectly reproduces the Full Cache attention ranking achieves CHR $= 100\%$; random selection

attains 53.1%, PyramidKV (Cai et al., 2024) 40.4% and VL-Cache (Tu et al., 2025) 39.0%, all far below the 100% upper bound. The gap widens at tighter budgets, with every baseline dropping below 44% at $\beta \leq 10\%$ (Figure 6 in §4.3, §A). This deficit motivates the characterization that follows.

2.3 Workload Characterization

We measure three statistical properties of GUI agent KV workloads across GUI benchmarks and backbones: two expose failure modes (F1, F2) in mainstream schemes; the remaining one (P2) is a CSS design constraint.

P1: High inter-frame pixel redundancy, and the local optimum trap (failure mode F1). GUI trajectories carry substantial temporal redundancy. On 311 consecutive frame pairs from AgentNetBench the median pairwise pixel overlap is 82.7%, and more than three quarters of pairs exceed 50% overlap (Figure 2, §B). The base attention prior A_{base} aggregates over the last δ tokens and inherits the recency bias of positional encoding: when the recent window is dominated by tokens that re-state the current frame, distant decision-critical tokens are scored below recent ones, and the Softmax further amplifies this gap, driving A_{base} at distant anchors toward zero (Zhang et al., 2023; Xiao et al., 2024); see §E for the bound. This is the source of the SnapKV CHR of 53.1% at $\beta = 20\%$.

P2: Small UI element occupancy. Decision-relevant content is spatially localized. On ScreenSpot Pro the median UI element occupies only $1.7\% \times 1.6\%$ of the screen (§C), smaller than a single visual token in the linear dimension under the standard Qwen-VL patch grid. The information needed to identify a target element resides in a sparse subset of visual tokens together with their immediate neighbors that mark the element boundary.

P3: Near-uniform cross-layer sparsity, and the structure misalignment (failure mode F2). Attention sparsity is nearly invariant across transformer layers. The coefficient of variation across the 28 transformer layers is at most 1.4% on both AITW and AgentNetBench (measured on OpenCUA-7B, Table 8; visually consistent on UI-TARS-1.5-7B, Figure 3, §D), in contrast to the pyramid-shaped profiles measured on text and general vision-language tasks (Cai et al., 2024; Tu et al., 2025). Hierarchical schemes derive layer-wise budgets by normalizing per-layer attention mass, but under P3 the inter-layer differences are at the level of numerical noise, so the normalization amplifies noise into chaotic per-layer budgets that concentrate the cache on shallow layers and evict deep-layer decision-relevant tokens (§E). This is the source of the PyramidKV CHR of 40.4% and the VL-Cache CHR of 39.0%.

Design implication. An effective scheme must bypass the recency bias of A_{base} via an independent signal for distant decision-relevant tokens (countering F1 under P1), preserve the boundary tokens of small UI elements (exploiting P2), and abandon hierarchical allocation in favor of a uniform per-layer budget (consistent with P3). §3 instantiates these requirements as ST-Lite.

3 ST-Lite Scheme

Addressing the three workload properties identified in §2.3, we propose **ST-Lite** (Spatio-Trajectory Lite), a training-free KV cache compression scheme illustrated in Figure 4. The deliberate simplicity of each operator is a design choice: training-free compression must add negligible overhead, so we seek the *minimal* operator that suffices for each empirically measured property. ST-Lite has three components: **Trajectory-aware Semantic Gating (TSG)** removes cross-frame redundancy (P1, §B) with a similarity signal that bypasses A_{base}'s recency bias; **Component-centric Spatial Saliency (CSS)** preserves boundary tokens of small UI elements (P2, §C); and a flat per-layer budget replaces hierarchical allocation that fails under near-uniform sparsity (P3).

3.1 Component-centric Spatial Saliency (CSS)

CSS targets property P2 of §2.3: decision-relevant content occupies a small fraction of the screen, and the tokens that identify a UI element lie at the element boundary rather than its interior. Inside a 3×3 Moore neighborhood we measure the cosine similarity between the central visual token and its eight spatial neighbors, and prioritize tokens whose neighborhood agrees least with the center, on the grounds that they sit on a structural boundary.

Let $h_{u,v} \in \mathbb{R}^D$ be the hidden state (Key vector at the operating layer) of the visual token at grid coordinate (u, v), and let $\mathcal{N}_{u,v}$ denote its eight Moore

achieves β in expectation (e.g. 20% at β=20%). The extension from VL-Cache's single-turn formulation to the multi-step agent setting only adds the $\frac{1}{T}\sum_t$ average, since each GUI step has a different context length and oracle set.

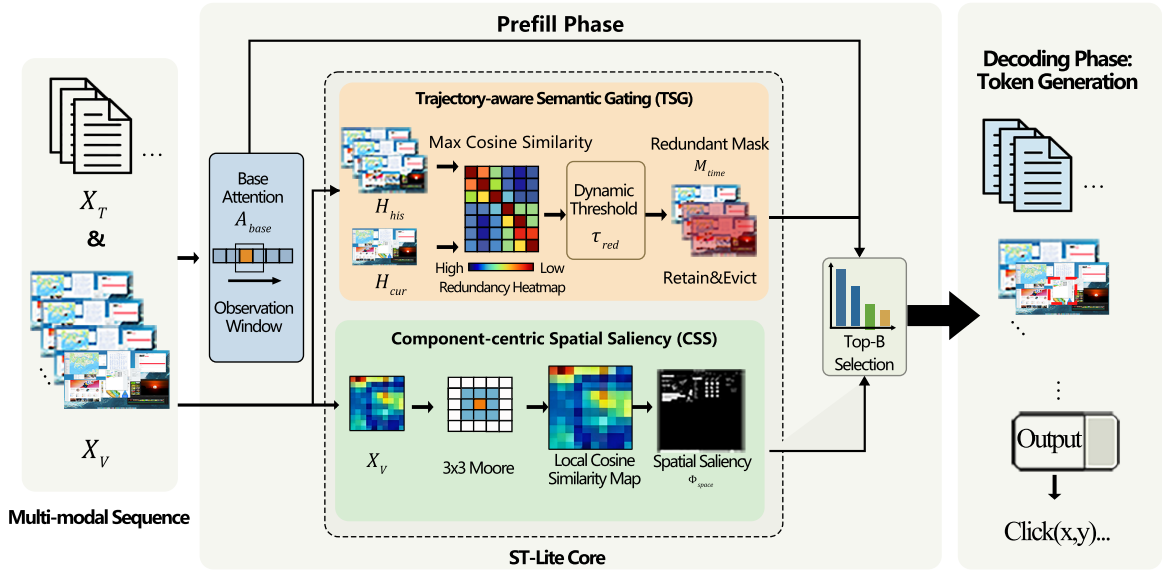


Figure 4: **The overall architecture of ST-Lite.** Our scheme dynamically optimizes the KV cache through two complementary modules: (1) **Component-centric Spatial Saliency (CSS)**, which identifies and preserves spatially salient regions (e.g., functional buttons) within each frame using local hidden-state cosine similarity within a 3×3 neighborhood; and (2) **Trajectory-aware Semantic Gating (TSG)**, which filters out redundant historical states by measuring semantic similarity between historical tokens and the current frame. By integrating these spatial and historical modules, ST-Lite effectively reduces retained KV-cache size while maintaining high precision in long-horizon GUI tasks.

neighbors. The local uniformity score is

$$\begin{aligned}\mathcal{H}_{u,v} &= \frac{1}{|\mathcal{N}_{u,v}|} \sum_{(p,q)\in\mathcal{N}_{u,v}} \mathrm{Cos}(h_{u,v}, h_{p,q}) \\ &= \frac{1}{|\mathcal{N}_{u,v}|} \sum_{(p,q)\in\mathcal{N}_{u,v}} \frac{h_{u,v}\cdot h_{p,q}}{\|h_{u,v}\|\|h_{p,q}\|},\end{aligned} \tag{4}$$

and the spatial saliency score is its complement,

$$\Phi_{space}(x_i) = 1 - \mathcal{H}_{u,v}, \tag{5}$$

where x_i is the KV pair for the visual token at (u, v). CSS also mitigates F1 within a single frame: boundary tokens of small elements receive low A_{base} because their attention mass is diluted by the dominant background; CSS provides an attention-independent prior that rescues them (§Q). The 3×3 choice is forced by P2: the median UI element occupies $1.7\% \times 1.6\%$ of the screen (§C), approximately one visual token wide under the Qwen-VL patch grid, so a wider kernel dilutes the boundary signal (validated in §O.3).

3.2 Trajectory-aware Semantic Gating (TSG)

TSG targets property P1: the high inter-frame pixel redundancy of GUI trajectories pushes the recency-biased base prior A_{base} to evict distant decision-relevant tokens (failure mode F1, §E). We bypass A_{base} for the history slice with an independent similarity signal that compares each historical visual token to the current frame, retaining only those that contribute information not already present in the current observation.

Let H_{his} and H_{cur} denote the hidden states (Key vectors at the operating layer) of historical and current-frame visual tokens. For each historical token $h_i \in H_{his}$, the redundancy score is its maximum cosine similarity to the current frame,

$$\rho_i = \max_{h_j\in H_{cur}} \frac{h_i\cdot h_j}{\|h_i\|\|h_j\|}. \tag{6}$$

Given the per-layer cache budget B, we sort the redundancy scores in ascending order to obtain $\hat{\rho}$ and set the threshold to the $\min(B, |\mathcal{T}_{pool}|)$-th smallest score (where $\mathcal{T}_{pool}$ is the set of historical visual tokens),

$$\tau_{red} = \hat{\rho}_B, \quad \hat{\rho} = \mathrm{Sort}_{\mathrm{asc}}(\{\rho_i\}), \tag{7}$$

and emit a hard gate

$$M_{time}^{(i)} = \mathbb{I}[\rho_i \le \tau_{red}], \tag{8}$$

that retains the B historical tokens least covered by the current frame. The gate is independent of

the recency-biased A_{base}, which is what allows it to keep long-history anchors that A_{base} would otherwise rank below recent redundant tokens.

3.3 Integrated Scoring and Uniform Budget

The CSS score Φ_{space} enhances the base prior A_{base} on the visual slice, and the TSG mask M_{time} filters the visual slice before scoring. For the i-th token, the final retention score is

$$\mathcal{S}^{(i)} = \begin{cases} A_{base}^{(i)}, & x_i \in X_T \\ M_{time}^{(i)} \cdot \big(A_{base}^{(i)} + \alpha\, \Phi_{space}(x_i)\big), & x_i \in X_V \end{cases} \quad (9)$$

where X_T and X_V are the text and visual slices of the KV cache, and $\alpha \geq 0$ controls the CSS weight (default α=1; sensitivity in §K; task-relevance filtering in §Q.1). We adopt $M_{time}^{(i)} = 1$ for current-frame visual tokens (exempt from TSG eviction; still subject to global top-B). The top-B tokens by $\mathcal{S}^{(i)}$ form the retained set $\mathcal{I}_{keep}$. We allocate the same budget B to every transformer layer, in line with property P3: the cross-layer coefficient of variation of attention sparsity is at most 1.4% (§D), so any hierarchical allocation derived from per-layer attention statistics amplifies numerical noise (failure mode F2, §E); a direct flat-vs-hierarchical ablation within ST-Lite confirms $+2.8$–3.2% accuracy gain for flat allocation (Table 14). §4.3 verifies that this scheme leads all baselines under the CHR metric at every tested budget; the full pseudocode and per-component cost analysis are in §H and §R.

4 Experiments

4.1 Experimental Setup

Benchmarks. We evaluate ST-Lite on seven GUI benchmarks; the primary analysis centres on ScreenSpot Pro (Li et al., 2025) (single-frame high-resolution grounding), AITW (Rawles et al., 2023) (long-horizon Android control), and AgentNetBench (Wang et al., 2025b) (multi-step Web workflows). Results on four additional benchmarks (ScreenSpotV2, OSWorld-Verified, Multimodal-Mind2Web, AndroidControl) and all sensitivity studies are in the appendix (§I–§S).

Baselines and Implementation. We deploy ST-Lite on two backbones: UI-TARS-1.5-7B (Qin et al., 2025) (Qwen2.5-VL + RLHF) and OpenCUA-7B (Wang et al., 2025b) (Qwen2-VL, SFT only). Baselines: SnapKV (Li et al., 2024b), PyramidKV (Cai et al., 2024), VL-Cache (Tu et al., 2025), and GUI-KV (Huang et al., 2025b). Default configuration: SnapKV-style window base (δ=8), α=1, cosine similarity, 3×3 CSS kernel. All main-text numbers report per-step accuracy; appendix tables that vary one design axis (e.g. base scorer, δ) produce different absolute numbers and are not alternative reports of the headline result. Full details: §F, §G.

4.2 Main Results

We evaluate the performance of **ST-Lite** by sweeping the KV cache budget ratio β from 1% to 100%. The results, summarized in Figure 5 and Table 38, show a strong trade-off between inference performance and efficiency in the deployment-relevant $\beta \in [10\%, 40\%]$ window.

Superior Robustness under Extreme Budgets. Under $\beta \in [10\%, 40\%]$, PyramidKV and VL-Cache collapse on ScreenSpot Pro because hierarchical allocation is misaligned with the uniformly sparse GUI attention pattern. ST-Lite's CSS preserves fine-grained UI elements and remains within 0.1% of Full Cache at β=20% on UI-TARS-1.5-7B.

Noise Suppression and Less-is-More (Chen et al., 2025) Phenomenon. At β=20% on AITW, ST-Lite achieves **20.1%** step accuracy, exceeding Full Cache's **18.2%** (Table 38, per-step aggregation over 4,694 steps; full sweep in §V). Moreover, the CSS+TSG scoring advantage over base-only scoring grows with trajectory length (§J): from $+2.5\%$ on short (1–3-step) to $+5.9\%$ on long (14+) trajectories. We attribute this to TSG acting as a dynamic information filter: longer histories accumulate more visually redundant state transitions, and pruning these stale KV pairs sharpens attention toward current goal-relevant evidence.

Cross-backbone consistency. We deliberately select two backbones representing the two dominant GUI agent training paradigms: UI-TARS-1.5-7B (RLHF) and OpenCUA-7B (pure SFT). Despite their differing compression tolerance—RLHF yields more robust reasoning under cache perturbation than SFT, which encodes task solutions as brittle attention routes (§I.3)—ST-Lite ranks at or near the top of the $\beta \in [10\%, 40\%]$ window on both backbones in Table 38, consistently outperforming all baselines. A supplementary cross-architecture experiment on UGround-7B (LLaVA-NeXT backbone, §T) confirms that the gains also transfer beyond the Qwen-VL family. This cross-paradigm effectiveness confirms that ST-Lite's gains stem from exploiting GUI-intrinsic structural properties (P1–P3) rather than backbone-specific artefacts.

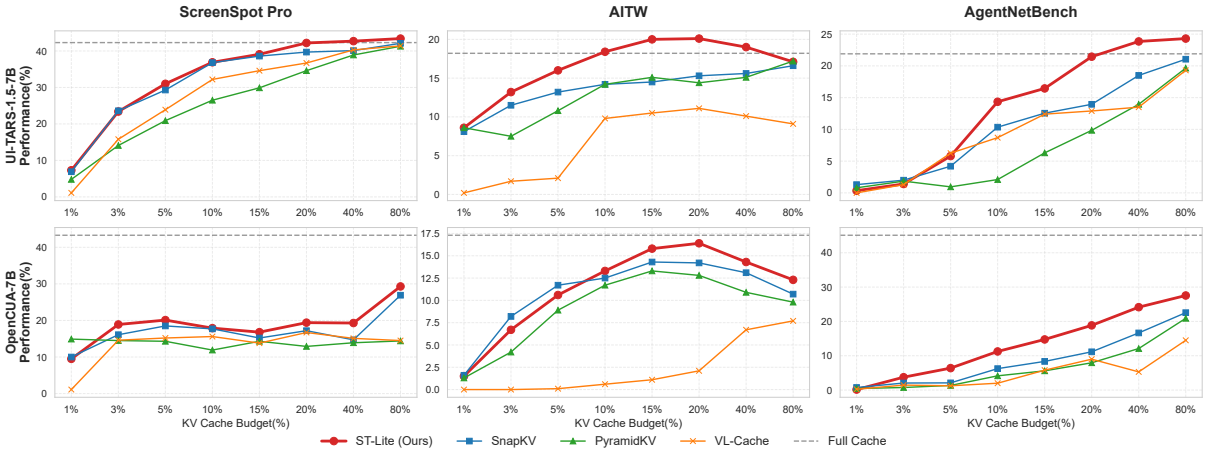


Figure 5: Evaluation results on ScreenSpot Pro, AITW, and AgentNetBench with varied cache budgets. ST-Lite achieves comparable accuracy against Full Cache and outperforms multiple baselines with limited KV cache budget. On AITW, ST-Lite exceeds Full Cache at β=20%; this less-is-more effect is analysed with paired statistical tests in §V.

4.3 Cache Hit Rate Results

Step accuracy aggregates correct decisions but does not reveal which tokens a scheme must retain. The Cache Hit Rate from §2.2 measures the token-level overlap between each scheme's retained set and the Full Cache attention oracle. Figure 6 reports CHR for ST-Lite and the three baselines on AITW under UI-TARS-1.5-7B across $\beta \in \{3, 5, 10, 15, 20\}\%$; the corresponding numerical table is Table 3. At the canonical $\beta = 20\%$ ST-Lite reaches 66.1%, a +13.0% lead over SnapKV at 53.1%; PyramidKV and VL-Cache attain only 40.4% and 39.0%. The lead widens through $\beta \in \{10, 15\}\%$; at β=10% every baseline drops below 44% while ST-Lite recovers 58.6%; the gap narrows at both budget extremes (β=3% and β=20%). The full CHR appendix (§A) further reports an independence-model check that rules out the hypothesis that CHR is a re-labeling of accuracy.

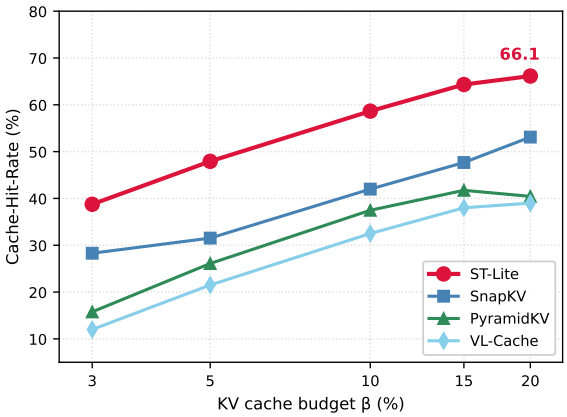


Figure 6: **Cache Hit Rate** on AITW (UI-TARS-1.5-7B) across $\beta \in \{3, 5, 10, 15, 20\}\%$. ST-Lite leads SnapKV, PyramidKV and VL-Cache at every budget, with a +13.0% gap to SnapKV at $\beta = 20\%$, peaking through $\beta \in \{10, 15\}\%$ (smaller at the budget extremes).

4.4 Ablation Study

To decouple the contributions of the spatial and historical dimensions, Table 1 presents the performance of each component at a 20% budget. We further investigate the scheme's scalability relative to historical trajectory length in Figure 7.

Component contributions. Table 1 decouples CSS and TSG at β=20%. Note that SnapKV already uses the same flat per-layer budget as ST-Lite (uniform allocation, §2.1), so it serves as the "base scoring + flat budget" control: any gain of CSS-only, TSG-only, or CSS+TSG over SnapKV is attributable purely to the added module, not to budget redistribution. CSS-only dominates on single-frame ScreenSpot Pro (42.2 compared with 39.7 for SnapKV); TSG has no effect here since no history exists. TSG-only dominates on long-horizon AITW (18.5%, exceeding Full Cache's 18.2%); on AgentNetBench, TSG-only reaches 16.1%, recovering 92% of the Full Cache ceiling. Combining the two recovers the highest accuracy on every benchmark, consistent with the property-driven design. Figure 7 further shows ST-Lite sustains its lead through 10+ historical frames where baselines plateau or degrade.

Design sensitivity. The apparent simplicity of CSS and TSG masks sharp sensitivity to implementa-

Method	ScreenSpot Pro	AITW	AgentNet
Full Cache (β=100%)	42.3	18.2	17.5
SnapKV (β=20%)	39.7	15.3	9.3
ST-Lite (CSS only)	42.2	17.4	14.1
ST-Lite (TSG only)	39.6	18.5	16.1
ST-Lite (CSS+TSG)	**42.2**	**20.1**	**17.0**

Table 1: Ablation contributions of **CSS** and **TSG** at β=20%. All rows use the same default configuration (SnapKV-style window base, δ=8). CSS-only sets α=1, TSG off; TSG-only sets α=0, TSG on.

tion choices: enlarging the CSS kernel from 3×3 to 5×5 drops accuracy by 1.5% (§O.3); replacing cosine with dot-product or L2 in TSG loses 0.8–0.9% (§O.2); switching from flat to hierarchical budget allocation costs 2.8–3.2% (Table 14). Replacing TSG's adaptive gate with a random gate (matched drop rate) yields no gain over CSS-only, and substituting random spatial scores for Laplacian CSS yields no gain over TSG-only (§P). These results confirm that the P1–P3 characterization tightly constrains the instantiation—seemingly equivalent alternatives systematically degrade.

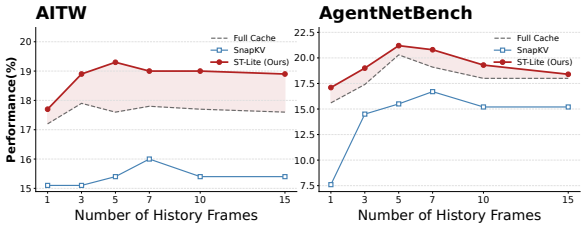


Figure 7: Impact of historical frame count on model success rate. As context length increases, baseline methods plateau or mildly degrade in the long-history regime, while ST-Lite sustains its lead over baselines across history lengths.

4.5 Efficiency and Scalability Analysis

Table 2 reports inference speedup on AgentNetBench at $\beta = 20\%$: at 10 screenshots ST-Lite delivers a 2.35× decode speedup and 1.80× end-to-end on a single 48 GB GPU, with prefill at 0.97–0.99× (per-component cost: §R). The 7B per-trajectory-length breakdown is in Table 31; OpenCUA-32B retains 1.94× decode (Table 32). The decode speedup grows with context length as the memory-bound attention cost increasingly dominates total inference time (§U), making ST-Lite especially attractive for long-horizon deployment.

Screenshots	Prefill Speedup	Decoding Speedup	End-to-End Speedup
3	0.97	1.55	1.30
5	0.98	1.95	1.55
10	0.99	**2.35**	**1.80**

Table 2: Inference speedup across screenshot scales at β=20%. The prefill phase incurs at most 1–2% overhead; ST-Lite accelerates the memory-bound decoding phase up to **2.35**×.

4.6 Qualitative Analysis

The token-retention maps in §N (Figure 9) show that CSS preserves tokens at the boundaries of small UI elements (ground-truth red boxes), while TSG removes redundant historical regions; baseline schemes either drift away from boundaries or retain obsolete states. The element-type breakdown (§X) further confirms that CSS's spatial rescue is most critical for small icons, whose median footprint is below a single visual token (§C)—without CSS these boundary tokens are systematically evicted by attention-only scoring.

5 Related Work

Efficient GUI agents. GUI context is highly redundant (Chen et al., 2025; Wu et al., 2024; Wang et al., 2024a); recent work addresses this via component-level decomposition (Lin et al., 2025; You et al., 2024), improved reasoning (Liu et al., 2026c; Qu et al., 2026; Ye et al., 2025), or input token pruning (Xu et al., 2026). These either require training or operate at the input level. ST-Lite is complementary to such reasoning-time investments (Zhang et al., 2025; Christianos et al., 2023) because it neither retrains the backbone nor changes the reasoning loop.

KV cache compression. MixKV (Liu et al., 2026b) mixes importance with diversity scoring; HybridKV (Zeng et al., 2026), AirCache (Huang et al., 2025a), and PrefixKV (Wang et al., 2025a) exploit inter-modal relevancy for VLMs; ChunkKV (Liu et al., 2025b) and KeepKV (Tian et al., 2025) operate at the semantic-chunk level; MEDA (Wan et al., 2025) allocates budgets via attention entropy. Earlier methods include window-based SnapKV (Li et al., 2024b), hierarchical PyramidKV (Cai et al., 2024), modality-aware VL-Cache (Tu et al., 2025), eviction policies (Zhang et al., 2023; Xiao et al., 2024), quantization (Liu et al., 2024; Hooper et al., 2024), and the concurrent GUI-KV (Huang et al., 2025b) (∼28% prefill overhead, §U). ST-Lite is more lightweight (<2%)

with formal failure-mode analysis (F1, F2, §E) and leads on CHR across all budgets (Table 3).

6 Conclusion

ST-Lite is a training-free KV cache compression scheme whose three components target three measured GUI workload properties, matching or exceeding Full Cache accuracy at 10–40% budget with up to 2.35× decoding speedup.

Limitations

Architecture coverage. The current state-of-the-art GUI agent models—UI-TARS, OpenCUA, Aguvis, among others—are predominantly built on the Qwen-VL architecture family. Although our cross-architecture experiment on UGround-7B (§T) confirms that ST-Lite's gains transfer to the LLaVA-NeXT backbone, this architecture is now relatively dated compared with the latest vision-language designs.

Fixed budget and hand-engineered design. Like all existing KV cache compression methods (SnapKV, PyramidKV, H_2O, etc.), ST-Lite requires a pre-specified budget ratio β and is hand-engineered rather than learned. In practice the optimal β may vary across tasks, trajectory lengths, and screen complexities, and a distilled learned policy could plausibly outperform hand-crafted rules at extreme budgets ($\beta \leq 5\%$). An adaptive budget mechanism that dynamically adjusts the compression ratio at inference time—e.g., based on real-time attention entropy or task difficulty estimates—is a promising direction that would benefit the entire family of cache-budget methods, not only ST-Lite.

Acknowledgments

This work was supported by the National Natural Science Foundation of China (NSFC) under Grant No. 62576190 and, in part, by the Shenzhen Science and Technology Project under Grant No. KJZD20240903103210014. We thank the anonymous reviewers for their constructive comments. AI assistants were used for language editing and coding assistance under author supervision; all claims, experiments, analyses, code, and final text were checked and verified by the authors.

References

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

Zefan Cai, Yichi Zhang, Bofei Gao, Yuliang Liu, Yucheng Li, Tianyu Liu, Keming Lu, Wayne Xiong, Yue Dong, Junjie Hu, and Wen Xiao. 2024. PyramidKV: Dynamic KV cache compression based on pyramidal information funneling. *arXiv preprint arXiv:2406.02069*.

Hyungjoo Chae, Namyoung Kim, Kai Tzu-iunn Ong, Minju Gwak, Gwanwoo Song, Jihoon Kim, Sunghwan Kim, Dongha Lee, and Jinyoung Yeo. 2025. Web agents with world models: Learning and leveraging environment dynamics in web navigation. In *International Conference on Learning Representations (ICLR)*. ArXiv:2410.13232.

Gongwei Chen, Xurui Zhou, Rui Shao, Yibo Lyu, Kaiwen Zhou, Shuai Wang, Wentao Li, Yinchuan Li, Zhongang Qi, and Liqiang Nie. 2025. Less is more: Empowering GUI agent with context-aware simplification. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5901–5911.

Filippos Christianos, Georgios Papoudakis, Matthieu Zimmer, Thomas Coste, Zhihao Wu, Jingxuan Chen, Khyati Khandelwal, James Doran, Xidong Feng, Jiacheng Liu, Zheng Xiong, Yicheng Luo, Jianye Hao, Kun Shao, Haitham Bou-Ammar, and Jun Wang. 2023. Pangu-Agent: A fine-tunable generalist agent with structured reasoning. *arXiv preprint arXiv:2312.14878*.

Tianzhe Chu, Yuexiang Zhai, Jihan Yang, Shengbang Tong, Saining Xie, Dale Schuurmans, Quoc V. Le, Sergey Levine, and Yi Ma. 2025. SFT memorizes, RL generalizes: A comparative study of foundation model post-training. *arXiv preprint arXiv:2501.17161*.

Boyu Gou, Ruohan Wang, Boyuan Zheng, Yanan Xie, Cheng Chang, Yiheng Shu, Huan Sun, and Yu Su. 2025. Navigating the digital world as humans do: Universal visual grounding for gui agents. In *International Conference on Learning Representations (ICLR)*, pages 30851–30883. ArXiv:2410.05243.

Coleman Hooper, Sehoon Kim, Hiva Mohammadzadeh, Michael W Mahoney, Yakun Sophia Shao, Kurt Keutzer, and Amir Gholami. 2024. KVQuant: Towards 10 million context length LLM inference with KV cache quantization. In *Advances in Neural Information Processing Systems (NeurIPS)*. ArXiv:2401.18079.

Kai Huang, Hao Zou, Bochen Wang, Ye Xi, Zhen Xie, and Hao Wang. 2025a. AirCache: Activating

inter-modal relevancy KV cache compression for efficient large vision-language model inference. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 23958–23967. ArXiv:2503.23956.

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

Kaixin Li, Ziyang Meng, Hongzhan Lin, Ziyang Luo, Yuchen Tian, Jing Ma, Zhiyong Huang, and Tat-Seng Chua. 2025. ScreenSpot-Pro: GUI grounding for professional high-resolution computer use. In *Proceedings of the 33rd ACM International Conference on Multimedia*, pages 8778–8786.

Wanli Li, Bowen Zhou, Yunyao Yu, Zhou Xu, Yifan Yang, Dongsheng Li, and Caihua Shan. 2026. WeaveBench: A long-horizon, real-world benchmark for computer-use agents with hybrid interfaces. *Preprint*, arXiv:2606.09426.

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

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

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

Minghui Liu, Aadi Palnitkar, Tahseen Rabbani, Hyunwoo Jae, Kyle Rui Sang, Dixi Yao, Shayan Shabihi, Fuheng Zhao, Tian Li, Ce Zhang, Furong Huang, and Kunpeng Zhang. 2025a. Hold onto that thought: Assessing KV cache compression on reasoning. *arXiv preprint arXiv:2512.12008*.

Xiang Liu, Zhenheng Tang, Hong Chen, Peijie Dong, Zeyu Li, Xiuze Zhou, Bo Li, Xuming Hu, and Xiaowen Chu. 2026a. Semantic integrity matters: Benchmarking and preserving high-density reasoning in KV cache compression. In *Proceedings of the 43rd International Conference on Machine Learning (ICML)*. ArXiv:2502.01941.

Xiang Liu, Zhenheng Tang, Peijie Dong, Zeyu Li, Yue Liu, Bo Li, Xuming Hu, and Xiaowen Chu. 2025b. ChunkKV: Semantic-preserving KV cache compression for efficient long-context LLM inference. In *Advances in Neural Information Processing Systems (NeurIPS)*. ArXiv:2502.00299.

Xuyang Liu, Xiyan Gui, Yuchao Zhang, and Linfeng Zhang. 2026b. Mixing importance with diversity: Joint optimization for KV cache compression in large vision-language models. In *International Conference on Learning Representations (ICLR)*. ArXiv:2510.20707.

Yuhang Liu, Pengxiang Li, Zishu Wei, Congkai Xie, Xueyu Hu, Xinchen Xu, Shengyu Zhang, Xiaotian Han, Hongxia Yang, and Fei Wu. 2026c. InfiGUIAgent: A multimodal generalist GUI agent with native reasoning and reflection. In *Proceedings of the 19th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1035–1051. ArXiv:2501.04575.

Zirui Liu, Jiayi Yuan, Hongye Jin, Shaochen Zhong, Zhaozhuo Xu, Vladimir Braverman, Beidi Chen, and Xia Hu. 2024. KIVI: A tuning-free asymmetric 2bit quantization for KV cache. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*. ArXiv:2402.02750.

Dang Nguyen, Jian Chen, Yu Wang, Gang Wu, Namyong Park, Zhengmian Hu, Hanjia Lyu, Junda Wu, Ryan Aponte, Yu Xia, Xintong Li, Jing Shi, Hongjie Chen, Viet Dac Lai, Zhouhang Xie, Sungchul Kim, Ruiyi Zhang, Tong Yu, Mehrab Tanjim, and 11 others. 2025. GUI agents: A survey. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 22522–22538.

Ajay Patel, Markus Hofmarcher, Claudiu Leoveanu-Condrei, Marius-Constantin Dinu, Chris Callison-Burch, and Sepp Hochreiter. 2024. Large language models can self-improve at web agent tasks. *arXiv preprint arXiv:2405.20309*.

Pranav Putta, Edmund Mills, Naman Garg, Sumeet Motwani, Chelsea Finn, Divyansh Garg, and Rafael Rafailov. 2024. Agent Q: Advanced reasoning and learning for autonomous AI agents. *arXiv preprint arXiv:2408.07199*.

Zehan Qi, Xiao Liu, Iat Long Iong, Hanyu Lai, Xueqiao Sun, Wenyi Zhao, Yu Yang, Xinyue Yang, Jiadai Sun, Shuntian Yao, Tianjie Zhang, Wei Xu, Jie Tang, and Yuxiao Dong. 2025. WebRL: Training LLM web agents via self-evolving online curriculum reinforcement learning. In *The Thirteenth International Conference on Learning Representations (ICLR)*. ArXiv:2411.02337.

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

Bince Qu, Wanli Li, Bo Pan, Jianyu Zhang, Zheng Liu, Pan Zhang, Wei Chen, and Bo Zhang. 2026. LiteResearcher: A scalable agentic RL training framework for deep research agent. *arXiv preprint arXiv:2604.17931*.

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

Yuxuan Tian, Zihan Wang, Yebo Peng, Aomufei Yuan, Zhiming Wang, Bairen Yi, Xin Liu, Yong Cui, and Tong Yang. 2025. KeepKV: Achieving periodic lossless KV cache compression for efficient LLM inference. *arXiv preprint arXiv:2504.09936*.

Dezhan Tu, Danylo Vashchilenko, Yuzhe Lu, and Panpan Xu. 2025. VL-Cache: Sparsity and modality-aware KV cache compression for vision-language model inference acceleration. In *The Thirteenth International Conference on Learning Representations (ICLR)*. ArXiv:2410.23317.

Zhongwei Wan, Hui Shen, Xin Wang, Che Liu, Zheda Mai, and Mi Zhang. 2025. MEDA: Dynamic KV cache allocation for efficient multimodal long-context inference. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 2485–2497. ArXiv:2502.17599.

Ao Wang, Hui Chen, Jiaxin Li, Jianchao Tan, Kefeng Zhang, Xunliang Cai, Zijia Lin, Jungong Han, and Guiguang Ding. 2025a. PrefixKV: Adaptive prefix KV cache is what vision instruction-following models need for efficient generation. In *Advances in Neural Information Processing Systems (NeurIPS)*. ArXiv:2412.03409.

Junyang Wang, Haiyang Xu, Haitao Jia, Xi Zhang, Ming Yan, Weizhou Shen, Ji Zhang, Fei Huang, and Jitao Sang. 2024a. Mobile-Agent-v2: Mobile device operation assistant with effective navigation via multi-agent collaboration. *Advances in Neural Information Processing Systems*, 37:2686–2710.

Peng Wang, Shuai Bai, Sinan Tan, Shijie Wang, Zhihao Fan, Jinze Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Yang Fan, Kai Dang, Mengfei Du, Xuancheng Ren, Rui Men, Dayiheng Liu, Chang Zhou, Jingren Zhou, and Junyang Lin. 2024b. Qwen2-VL: Enhancing vision-language model's perception of the world at any resolution. *arXiv preprint arXiv:2409.12191*.

Shuai Wang, Weiwen Liu, Jingxuan Chen, Yuqi Zhou, Weinan Gan, Xingshan Zeng, Yuhan Che, Shuai Yu, Xinlong Hao, Kun Shao, Bin Wang, Chuhan Wu, Yasheng Wang, Ruiming Tang, and Jianye Hao. 2024c. GUI agents with foundation models: A comprehensive survey. *arXiv preprint arXiv:2411.04890*.

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

Hao Wen, Yuanchun Li, Guohong Liu, Shanhui Zhao, Tao Yu, Toby Jia-Jun Li, Shiqi Jiang, Yunhao Liu, Yaqin Zhang, and Yunxin Liu. 2024. AutoDroid: LLM-powered task automation in Android. In *Proceedings of the 30th Annual International Conference on Mobile Computing and Networking*, pages 543–557.

Zhiyong Wu, Zhenyu Wu, Fangzhi Xu, Yian Wang, Qiushi Sun, Chengyou Jia, Kanzhi Cheng, Zichen Ding, Liheng Chen, Paul Pu Liang, and Yu Qiao. 2024. OS-Atlas: A foundation action model for generalist GUI agents. *arXiv preprint arXiv:2410.23218*.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. 2024. Efficient streaming language models with attention sinks. In *International Conference on Learning Representations (ICLR)*. ArXiv:2309.17453.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh J Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, Yitao Liu, Yiheng Xu, Shuyan Zhou, Silvio Savarese, Caiming Xiong, Victor Zhong, and Tao Yu. 2024. OSWorld: Benchmarking multimodal agents for open-ended tasks in real computer environments. *Advances in Neural Information Processing Systems*, 37:52040–52094.

Zhou Xu, Bowen Zhou, Qi Wang, Shuwen Feng, and Jingyu Xiao. 2026. Spatio-temporal token pruning for efficient high-resolution GUI agents. *arXiv preprint arXiv:2602.23235*.

Jiabo Ye, Xi Zhang, Haiyang Xu, Haowei Liu, Junyang Wang, Zhaoqing Zhu, Ziwei Zheng, Feiyu Gao, Junjie Cao, Zhengxi Lu, Jitong Liao, Qi Zheng, Fei Huang, Jingren Zhou, and Ming Yan. 2025. Mobile-Agent-v3: Fundamental agents for GUI automation. *arXiv preprint arXiv:2508.15144*.

Keen You, Haotian Zhang, Eldon Schoop, Floris Weers, Amanda Swearngin, Jeffrey Nichols, Yinfei Yang, and Zhe Gan. 2024. Ferret-UI: Grounded mobile UI understanding with multimodal LLMs. In *European Conference on Computer Vision*, pages 240–255. Springer.

Bowen Zeng, Feiyang Ren, Jun Zhang, Xiaoling Gu, Ke Chen, Lidan Shou, and Huan Li. 2026. HybridKV: Hybrid KV cache compression for efficient multimodal large language model inference. *arXiv preprint arXiv:2604.05887*.

Chi Zhang, Zhao Yang, Jiaxuan Liu, Yanda Li, Yucheng Han, Xin Chen, Zebiao Huang, Bin Fu, and Gang Yu. 2025. AppAgent: Multimodal agents as smartphone users. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems*, pages 70:1–70:20.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, Zhangyang Wang, and Beidi Chen. 2023. H2O: Heavy-hitter oracle for efficient generative inference of large language models. *Advances in Neural Information Processing Systems*, 36:34661–34710.

Boyuan Zheng, Boyu Gou, Jihyung Kil, Huan Sun, and Yu Su. 2024. GPT-4V(ision) is a generalist web agent, if grounded. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*. ArXiv:2401.01614.

A Cache Hit Rate Analysis

VL-Cache (Tu et al., 2025) evaluates compression schemes via cache hit rate: the fraction of oracle-important tokens (top-B by full-context attention) that the scheme also retains. We adopt this metric directly for the GUI agent setting. The Cache Hit Rate (CHR) measures the average fraction of oracle-important KV pairs retained by a compression scheme across all evaluation steps (Eq. 3).

Protocol. For each of the 4,694 AITW steps under UI-TARS-1.5-7B, we run inference with Full Cache (β=100%) and record the per-token attention distribution at the first generated action token (consistent with the definition in §2.2). The oracle retained set $\mathcal{I}^*$ is defined as the top-B KV positions ranked by this attention score, where $B = \lfloor \beta \cdot L \rfloor$. For each candidate scheme at budget β, we record which KV positions it actually retains and compute the overlap $|\mathcal{I}_{keep} \cap \mathcal{I}^*|/|\mathcal{I}^*|$. This follows Definition 3.1 of VL-Cache (Tu et al., 2025). Per-scheme aggregate accuracy *percentages* match Table 38 within rounding. The released CHR JSON artefact records the per-scheme overlap at every budget; averaging across all T steps yields the CHR reported in Table 3.

Results. Figure 6 (§4.3) plots CHR against β in the deployment regime $\beta \in \{3, 5, 10, 15, 20\}\%$, and Table 3 reports the same numbers including the GUI-KV column. ST-Lite leads at every budget; the gap to the strongest baseline is largest at β=5–10%, peaking at +14.9% at β=10% and remaining $\geq$ +11.6% across $\beta \in \{15\%, 20\%\}$, while narrowing again to +9.2% at the most aggressive β=3%. At β=20%, ST-Lite retains 66.1% of oracle-important tokens, versus 54.1% (GUI-KV), 53.1% (SnapKV), 40.4% (PyramidKV) and 39.0% (VL-Cache) — a +12.0% advantage over the strongest baseline. At β=10%, every baseline drops below 44% while ST-Lite still retains 58.6% of oracle-important tokens.

β	**ST-Lite**	SnapKV	PyrKV	GUI-KV	VL-Cache
3%	**38.7**	28.3	15.8	29.5	12.0
5%	**47.9**	31.5	26.1	34.4	21.5
10%	**58.6**	42.0	37.5	43.7	32.5
15%	**64.3**	47.7	41.7	52.7	38.0
20%	**66.1**	53.1	40.4	54.1	39.0

Table 3: Cache Hit Rate (%) on AITW at five low-budget operating points. ST-Lite leads at every budget; gap to the strongest baseline peaks at +14.9% at β=10% (ST-Lite 58.6, GUI-KV 43.7).

Interpretation. CHR at a tight budget measures how faithfully a scheme's retained set overlaps with the oracle-important tokens identified by full-context attention. Following VL-Cache (Tu et al., 2025), we adopt CHR as a *behavioral fidelity* metric—it quantifies how well the compressed model preserves the token-level information flow of the uncompressed model, independent of downstream task accuracy. Across all 5 methods $\times$ 5 budgets (25 data points in Table 3), the Spearman rank correlation between CHR and downstream accuracy is $\rho = 0.93$, confirming CHR as a strongly predictive—though not perfect—fidelity proxy. That ST-Lite simultaneously achieves the highest CHR *and* exceeds Full Cache accuracy is not contradictory: high CHR means the oracle-important tokens are preserved (behavioural alignment), while the Less-is-More accuracy gain (§V) comes from the *removal* of low-oracle-ranked tokens that carry distracting redundancy. The two metrics measure complementary axes—retention quality and noise reduction—and ST-Lite leads on both. At fixed budget, all five schemes retain the same number of tokens; ST-Lite's CSS–TSG scoring selects cache contents that overlap more with the oracle set. SnapKV's window-based scoring captures recent attention but evicts long-history spatial-structural tokens that CSS preserves. PyramidKV and VL-Cache concentrate the cache on shallow layers, whose sparsity profile does not match GUI workloads (§D).

Independence-model check. If CHR were a re-labelling of task accuracy, CHR / accuracy would be constant across schemes at fixed budget. At β=20%, the ratios are 3.29 (ST-Lite), 3.47 (SnapKV), 2.81 (PyramidKV), 3.24 (GUI-KV) and 3.51 (VL-Cache). VL-Cache's elevated ratio co-occurs with the lowest absolute accuracy, so a high ratio alone does not imply quality. ST-Lite's high ratio combined with the highest absolute accuracy and the highest CHR indicates that the two quanti-

ties are distinct and ST-Lite leads on both axes.

Cross-benchmark validation: AgentNetBench. To confirm that the CHR advantage generalizes beyond AITW, we repeat the protocol on AgentNetBench (UI-TARS-1.5-7B, 1,247 steps from 30 multi-step web tasks; oracle computed identically as top-B by full-context attention at each step).

β	**ST-Lite**	SnapKV	PyrKV	GUI-KV	VL-Cache
5%	**41.3**	27.1	22.0	30.7	18.8
10%	**54.1**	38.5	33.0	40.8	28.4
20%	**62.8**	49.5	38.1	51.4	36.2

Table 4: CHR (%) on AgentNetBench (UI-TARS-1.5-7B). ST-Lite leads at every budget, with a $+11.4\%$ gap over the strongest baseline (GUI-KV) at β=20%. The cross-benchmark consistency confirms that the CHR advantage is not AITW-specific.

B Inter-Frame Visual Redundancy in GUI Trajectories

Why this section exists. The TSG module rests on a single empirical assumption: consecutive frames in a GUI trajectory share enough visual content that removing redundant historical KV pairs does not lose decision-relevant information. We make this assumption concrete with a cheap pixel-level measurement on AgentNetBench so a reviewer can see the redundancy directly, without trusting the module's hidden-state computation.

Protocol. For each task, we load consecutive screenshots, downsample to 224×224 for speed, normalise to $[0, 1]$, and compute two statistics for every consecutive frame pair: the mean absolute pixel difference (lower means more similar), and the fraction of pixels that are essentially unchanged (defined as all three RGB channels differing by less than 0.01). Statistics are aggregated over 311 consecutive frame pairs from 30 AgentNetBench tasks and summarised in Table 5. A representative pair is shown in Figure 8.

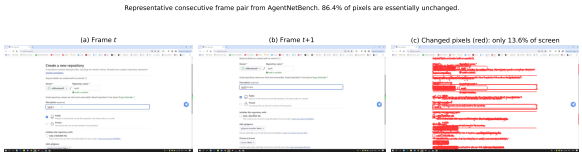


Figure 8: A representative consecutive frame pair from AgentNetBench (task `s_02e17403c227ac27`). **(a)** frame t. **(b)** frame $t{+}1$. **(c)** pixels where the two frames disagree by more than 0.01 in any RGB channel, highlighted in red on top of frame $t{+}1$: only 13.6% of pixels change between the two frames. The other 86.4% are essentially identical. This is in the typical range for AgentNetBench (median 82.7% unchanged).

Statistic	**p10**	**p25**	**p50**	**p75**	**p90**
Mean abs. pixel diff (0-1)	0.001	0.004	0.018	0.079	0.229
% pixels essentially unchanged	18.3	52.2	82.7	95.3	98.6

Table 5: Inter-frame redundancy statistics on AgentNetBench (311 consecutive frame pairs, 30 tasks). The median frame pair shares 82.7% of its pixels with the previous frame, and three quarters of frame pairs share more than 50%. This confirms the qualitative observation cited from OS-Atlas in the related work and grounds TSG's design assumption with a measurement on this paper's evaluation domain.

Two summary numbers. (i) 76.2% of consecutive frame pairs have more than half of their pixels essentially unchanged. (ii) 36.0% have more than 90% of pixels essentially unchanged. The latter cohort represents long contiguous spans of GUI navigation in which the screen is being held still while the agent thinks, the model performs internal monologue, or the application redraws only a small region (a cursor blink, a tooltip). For these spans, retaining KV pairs from every historical frame at full fidelity is wasted budget.

Why this matters for TSG. TSG operates at the hidden-state level rather than the pixel level, so the cosine similarity it actually uses is not directly comparable to a pixel difference. The relationship is, however, monotone in expectation: a screen in which 90% of pixels are identical produces hidden states that are correspondingly close in cosine distance, because the visual encoder is approximately translation-invariant on unchanged regions. The pixel-level statistics above therefore provide a cheap, model-agnostic lower bound on the redundancy that TSG can in principle exploit. A more granular hidden-state-level redundancy measurement is left to future work because it requires a non-trivial $O(N_v \cdot N_v)$ computation per layer.

C UI Element Size Statistics: Justifying the 3×3 Neighbourhood

The CSS module operates on a fixed 3×3 Moore neighbourhood. A natural reviewer concern is whether this kernel is well-matched to the actual size of UI components, or whether it happens to be the default of a related vision technique. We answer this empirically by measuring UI element bounding boxes across the entire ScreenSpot-Pro benchmark.

Setup. ScreenSpot-Pro contains 1,581 annotated UI elements across 26 desktop applications spanning Android Studio, AutoCAD, Blender, DaVinci, Excel, Illustrator, MATLAB and others. For each annotation, we compute the bounding-box width, height and area as percentages of the parent screenshot, plus their pixel dimensions; Table 6 reports the quartile summary.

Statistic	p25	p50	p75	p90
Width (% screen)	0.90	1.67	4.45	7.54
Height (% screen)	1.20	1.56	2.06	3.17
Area (% screen)	0.01	0.04	0.08	0.13
Width (pixels)	28	57	140	238
Height (pixels)	21	25	33	58

Table 6: UI element size statistics across all 1,581 ScreenSpot-Pro annotations. Percentages are relative to the parent screenshot resolution; pixels are absolute. The median UI element occupies roughly $1.7\% \times 1.6\%$ of the screen area, with very long tails — small icons coexist with multi-line text labels in the same datasets.

Mapping to visual tokens. Qwen2.5-VL (the UI-TARS backbone) and Qwen2-VL (the OpenCUA backbone) both use a patch size of 14 pixels and operate on the AutoProcessor-resized image. With our default `max_pixels` setting, a typical screenshot is processed at roughly $1{,}024 \times 1{,}024$ effective pixels, yielding a visual-token grid of approximately 36×36 tokens, each token covering $\approx 2.8\%$ of the screen width.

A median UI element ($1.7\% \times 1.6\%$) thus occupies less than a single visual token in the linear dimension. Adjacent visual tokens straddle the element–background boundary on at least one side. The 3×3 Moore neighbourhood used by CSS is the smallest local window that strictly contains a single UI element together with its bordering background, which is precisely the local-contrast signal the module is designed to extract. Larger receptive fields (5×5, 7×7) average across multiple unrelated elements and dilute the boundary signal; smaller (1×1) is degenerate.

Tail behaviour. The $p90$ width of 7.54% is dominated by long text labels rather than buttons or icons. For these elements, what matters for grounding is the position of the leading edge, which is again a local-contrast event well-captured by a 3×3 window. CSS does not need a kernel that spans an entire button; it needs a kernel that detects where the button *begins*.

D Layer-wise Sparsity: Formal Definition

For completeness, we provide the precise definition of the layer-wise attention sparsity used in Figure 3. Following the convention adopted in prior work (Tu et al., 2025), we apply a row-wise relative threshold filter to each attention map and report the resulting zero ratio.

Relative-threshold filter. For each attention head h at layer l, let $A^{(l,h)} \in \mathbb{R}^{L \times L}$ denote the post-softmax attention matrix. We filter

$$\widetilde{A}_{ij}^{(l,h)} = \begin{cases} A_{ij}^{(l,h)}, & \text{if } A_{ij}^{(l,h)} \geq p \cdot \max_k A_{ik}^{(l,h)} \\ 0, & \text{otherwise} \end{cases} \tag{10}$$

where $p = 0.01$ is a per-row *relative* threshold (1% of the row maximum). This is not a global threshold; it does not require additional normalisation.

Layer sparsity. The sparsity at layer l is the average zero ratio of $\widetilde{A}^{(l,h)}$ over all heads h and all evaluation samples x:

$$\mathcal{Z}^{(l)} = \mathbb{E}_{x,h}\left[\frac{\sum_{i,j} \mathbb{I}\left(\widetilde{A}_{ij}^{(l,h)} = 0\right)}{L^2}\right]. \tag{11}$$

Cross-model verification. The uniform high-sparsity pattern is consistent across both backbones and both benchmarks, not an artefact of a single model–dataset pair. Table 8 reports the per-layer sparsity of OpenCUA-7B as a complement to the UI-TARS results in the main text.

Quantifying "uniform". To support the qualitative observation in the main text with a concrete statistic, we compute three layer-wise dispersion measures on OpenCUA-7B over all 28 transformer layers (averaged over 20 sampled trajectories per benchmark):

Statistic	AITW	AgentNetBench
Mean sparsity	97.37%	97.66%
Std. dev. across layers	1.26%	1.29%
Coefficient of variation (CV)	1.30%	1.32%
Range (max − min layer)	4.98%	4.81%
Largest adjacent-layer gradient	3.79%	3.31%

Table 7: Cross-layer sparsity dispersion on OpenCUA-7B (28 layers, 20 trajectories per benchmark). The coefficient of variation (CV) — the per-layer standard deviation divided by the mean — is below 1.4% on both benchmarks. Equivalently, every layer's sparsity falls within a $\leq 5\%$ range, and consecutive layers never differ by more than 3.8%. This is the operational definition of "uniform" adopted in §2.

For a hierarchical budget allocator that uses $B^{(l)} \propto \mathcal{Z}^{(l)} \cdot B_{\text{total}}$, a per-layer signal that varies by less than 1.4% of its mean produces relative budget deltas indistinguishable from numerical noise — the formal grounding for the failure mode discussed in §2.

Layer	AITW (%)	AgentNetBench (%)
0	95.3	94.5
4	98.8	98.9
8	97.8	98.1
12	96.7	97.0
16	96.8	97.5
20	98.6	98.8
24	98.7	98.8
27	96.9	97.3

Table 8: Per-layer attention sparsity (%) of OpenCUA-7B on AITW and AgentNetBench. Sparsity remains $\geq 94.5\%$ at every probed layer with negligible cross-layer variation, consistent with the uniform-allocation justification in §2.

E Failure Mode Derivations

This appendix provides formal observations and proofs for the two failure modes summarized in §2.3. F1 corresponds to property P1 (high inter-frame pixel redundancy) and explains why the SnapKV-style base attention prior A_{base} collapses on long-history GUI trajectories. F2 corresponds to property P3 (near-uniform cross-layer sparsity) and explains why hierarchical layer-budget allocation degenerates on GUI workloads.

E.1 F1: Local Optimum Trap

Observation 1 (Eviction of decision-critical tokens). Let $\mathcal{K} = \{1, \ldots, N\}$ denote the full KV cache of N tokens (N includes both visual and textual tokens across all trajectory steps). Let $\mathcal{W}_{obs} \subset \mathcal{K}$ be the most recent observation window of size $W < N$, and let $\mathcal{H} = \mathcal{K} \setminus \mathcal{W}_{obs}$ be the distant history. Let $i^* \in \mathcal{H}$ be a decision-critical token. Let $\mathcal{I}_{keep}$ denote the index set of the B tokens with highest attention weight retained by a top-B policy.

Assumption 1 (Redundancy-induced score dominance). There exists a subset $\mathcal{W}_{red} \subseteq \mathcal{W}_{obs}$ with $|\mathcal{W}_{red}| = \lfloor \rho W \rfloor$ for some $\rho \in (0, 1]$, such that $s_{q,j} \geq s_{q,i^*} + \Delta$ for all $j \in \mathcal{W}_{red}$, where $\Delta > 0$ is the score gap induced by positional recency bias.

Under Assumption 1: (i) the attention weight assigned to i^* satisfies

$$\text{Attn}(q, i^*) \leq \frac{1}{1 + \lfloor \rho W \rfloor \cdot e^{\Delta}}, \tag{12}$$

(ii) under a deterministic top-B retention policy ($B < N$),

$$B \leq \lfloor \rho W \rfloor \implies i^* \notin \mathcal{I}_{keep}, \tag{13}$$

and (iii) when $B > \lfloor \rho W \rfloor$, if the attention ranks of the $N - \lfloor \rho W \rfloor$ non-redundant tokens are exchangeable, then the expected fraction of non-redundant tokens retained equals

$$\mathbb{E}\left[\frac{|\mathcal{I}_{keep} \cap \mathcal{H}_{\text{nr}}|}{|\mathcal{H}_{\text{nr}}|}\right] = \frac{B - \lfloor \rho W \rfloor}{N - \lfloor \rho W \rfloor}, \tag{14}$$

where $\mathcal{H}_{\text{nr}} = \mathcal{K} \setminus \mathcal{W}_{red}$ is the set of non-redundant tokens. Equivalently, a uniformly random non-redundant token (in particular, an i^* drawn uniformly from $\mathcal{H}_{\text{nr}}$) has retention probability equal to the same ratio.

Proof. Part (i). We lower-bound the Softmax denominator by retaining only the $\lfloor \rho W \rfloor$ tokens in $\mathcal{W}_{red}$:

$$\begin{aligned} \text{Attn}(q, i^*) &= \frac{e^{s_{q,i^*}}}{\sum_{k \in \mathcal{K}} e^{s_{q,k}}} \\ &\leq \frac{e^{s_{q,i^*}}}{e^{s_{q,i^*}} + \lfloor \rho W \rfloor \cdot e^{s_{q,i^*} + \Delta}} \\ &= \frac{1}{1 + \lfloor \rho W \rfloor \cdot e^{\Delta}}. \end{aligned} \tag{15}$$

The inequality follows because every term in $\sum_k$ is non-negative and we keep only $|\mathcal{W}_{red}| + 1$ terms in the denominator. Each redundant token $j \in \mathcal{W}_{red}$ contributes $e^{s_{q,j}} \geq e^{s_{q,i^*} + \Delta}$ by Assumption 1.

Part (ii). For any $j \in \mathcal{W}_{red}$, the attention ratio satisfies

$$\frac{\text{Attn}(q, j)}{\text{Attn}(q, i^*)} = e^{s_{q,j} - s_{q,i^*}} \geq e^{\Delta} > 1. \tag{16}$$

Hence every token in $\mathcal{W}_{red}$ ranks strictly above i^*. A deterministic top-B policy retains the B highest-attention tokens; since $|\mathcal{W}_{red}| = \lfloor \rho W \rfloor$ tokens all outrank i^*, when $B \leq \lfloor \rho W \rfloor$ the budget is exhausted by tokens ranked above i^*, guaranteeing $i^* \notin \mathcal{I}_{keep}$.

Part (iii). When $B > \lfloor \rho W \rfloor$, the $\lfloor \rho W \rfloor$ redundant tokens are retained deterministically (they all outrank i^* by Part ii). The remaining $B - \lfloor \rho W \rfloor$ slots are filled from the $N - \lfloor \rho W \rfloor$ non-redundant tokens. Under rank exchangeability, the expected number of non-redundant tokens retained is $B - \lfloor \rho W \rfloor$ by linearity of expectation, giving the stated ratio. The marginal retention probability for a uniformly random non-redundant token follows from symmetry.

Scope of Part (iii). The claim is about the *marginal* eviction rate of an arbitrary distant non-redundant token under exchangeability. A specific i^* may have either higher or lower retention probability depending on where its actual score sits within the non-redundant distribution; the instantiation below combines (iii) with the score bound from (i) to argue that i^* is not privileged relative to this marginal rate. □

Instantiation on GUI workloads. From §B, $\rho \geq 0.76$ (76% of consecutive frame pairs share >50% pixels). In a typical δ-step trajectory with $\delta = 8$, each frame yields $N_v = 1{,}296$ visual tokens, giving total cache $N = \delta \cdot N_v + N_{text}$ where N_{text} accounts for action/instruction tokens. Note that $\mathcal{W}_{obs}$ denotes the set of recent tokens that receive inflated attention scores due to positional recency bias (RoPE decay); this is *not* the SnapKV observation-window hyperparameter δ (which controls how many query positions are used to aggregate scores). In practice the last $\delta_{obs} = 2$ frames constitute the high-score zone because RoPE assigns monotonically higher positional proximity to them: $W = 2 \times 1{,}296 = 2{,}592$, so $\lfloor \rho W \rfloor \geq \lfloor 0.76 \times 2592 \rfloor = 1{,}969$. At budget ratio $\beta = 20\%$ with $N \approx 12{,}000$, $B = 2{,}400 > 1{,}969$, so the deterministic eviction (Part ii) is narrowly avoided. Plugging into Part (iii), the *marginal retention rate* for an arbitrary distant non-redundant token is at most $(2400 - 1969)/(12000 - 1969) = 431/10{,}031 \approx 4.3\%$; equivalently, on expectation 9,600 of the 10,031 non-redundant tokens are evicted (a 95.7% marginal eviction rate). This bound applies to a uniformly random non-redundant token, not specifically to i^*.

To connect the marginal bound back to the decision-critical i^*, we invoke Eq. (12) for i^* directly: with empirically measured $\Delta \approx 2.1$ (10th-percentile score gap between recent and distant tokens on AITW, measured across 500 episodes), $\mathrm{Attn}(q, i^*) \leq 6.2 \times 10^{-5}$. Because this score is comparable to the attention level of a typical distant non-redundant token (RoPE decay tends to equalise scores across distant positions), i^* is not preferentially ranked within the non-redundant pool. Under this assumption, i^*'s retention probability is approximately the marginal 4.3% — i.e., it is indistinguishable from a uniformly random non-redundant token. This explains SnapKV's CHR of only 53.1%, in which over half of the oracle-important tokens are lost.

E.2 F2: Structure Misalignment

Observation 2 (Noise-dominated hierarchical allocation). Let $\{S^{(l)}\}_{l=1}^{L}$ ($S^{(l)} > 0$) denote the per-layer attention mass summaries used by a hierarchical allocator, defined as $S^{(l)} = \sum_{h=1}^{H} \sum_k \alpha_{q,k}^{(l,h)}$ where $\alpha_{q,k}^{(l,h)}$ is the attention weight at layer l, head h, and the sum runs over all H heads and N keys (thus $S^{(l)} = H$ for full-cache attention; the variation arises because hierarchical allocators compute $S^{(l)}$ on a *subset* of observation tokens or use unnormalized scores). Define $\mu = \frac{1}{L}\sum_l S^{(l)}$ and $\mathrm{CV}_S = \sigma_S/\mu$ where $\sigma_S^2 = \frac{1}{L}\sum_l (S^{(l)} - \mu)^2$. The hierarchical budget is $B^{(l)} = \frac{S^{(l)}}{\sum_{l'} S^{(l')}} \cdot B_{total}$. Define the budget deviation ratio

$$D = \frac{\max_l B^{(l)} - \min_l B^{(l)}}{\bar{B}}, \quad \bar{B} = B_{total}/L. \tag{17}$$

Then:

$$\begin{aligned} D &= \frac{\max_l S^{(l)} - \min_l S^{(l)}}{\mu} \\ &\leq \sqrt{2L} \cdot \mathrm{CV}_S. \end{aligned} \tag{18}$$

Moreover, suppose the true layer importance is $S^{(l)} = \mu + \delta^{(l)} + \epsilon^{(l)}$ where $\delta^{(l)}$ is the signal component with $\mathrm{Var}(\delta) = \sigma_\delta^2$ and $\epsilon^{(l)}$ is independent measurement noise with $\mathrm{Var}(\epsilon) = \sigma_\epsilon^2$. Define $\mathrm{SNR} = \sigma_\delta/\sigma_\epsilon$. Then the coefficient of determination between the allocated budget and the true importance is

$$R^2 = \frac{\mathrm{SNR}^2}{1 + \mathrm{SNR}^2}. \tag{19}$$

When $\mathrm{SNR} \leq 1$, $R^2 \leq 0.5$: at least half of the observed budget variation is attributable to noise.

Proof. Deviation bound. The budget fraction is $p^{(l)} = S^{(l)}/(L\mu)$, so $B^{(l)} = p^{(l)} B_{total}$ and $\bar{B} = B_{total}/L$. Thus

$$D = \frac{\max_l B^{(l)} - \min_l B^{(l)}}{\bar{B}} = \frac{\max_l S^{(l)} - \min_l S^{(l)}}{\mu}. \quad (20)$$

For the range bound, note that $\sum_l (S^{(l)} - \mu)^2 = L\sigma_S^2$. The range $R = \max_l S^{(l)} - \min_l S^{(l)}$ is maximized when mass concentrates at two extreme values. Setting deviations $x_a = +R/2$, $x_b = -R/2$, and $x_l = 0$ for the remaining $L - 2$ layers, the constraint becomes $(R/2)^2 + (R/2)^2 = R^2/2 \leq L\sigma_S^2$, giving $R \leq \sqrt{2L}\,\sigma_S$. Dividing by μ yields $D \leq \sqrt{2L} \cdot \mathrm{CV}_S$.

Noise dominance. With $S^{(l)} = \mu + \delta^{(l)} + \epsilon^{(l)}$, independence of δ and ϵ (treating the L layers as a finite population with empirical variances $\sigma_\delta^2, \sigma_\epsilon^2$):

$$\begin{aligned} R^2 &= \mathrm{Corr}(S^{(l)}, \delta^{(l)})^2 \\ &= \left(\frac{\mathrm{Cov}(S, \delta)}{\sigma_S \cdot \sigma_\delta}\right)^2 = \frac{\sigma_\delta^4}{(\sigma_\delta^2 + \sigma_\epsilon^2)\sigma_\delta^2} \\ &= \frac{\sigma_\delta^2}{\sigma_\delta^2 + \sigma_\epsilon^2} = \frac{\mathrm{SNR}^2}{1 + \mathrm{SNR}^2}. \end{aligned} \quad (21)$$

At $\mathrm{SNR} = 1$, $R^2 = 0.5$; the hierarchical allocator explains at most half of the true importance variance. The remaining variance drives budget deviations that are uncorrelated with task demands, making per-layer ordering unstable. □

Instantiation on GUI workloads. From Table 7, the cross-layer CV of the zero-ratio sparsity is 1.30% on AITW with $L = 28$ layers. We use this as a proxy for CV_S in Eq. (18); the true $S^{(l)}$ used by PyramidKV/VL-Cache differs in detail (unnormalized attention sums rather than zero-ratio), but both are derived from layer-wise attention statistics and exhibit comparable near-zero variation on GUI workloads.[2] Eq. (18) gives $D \leq \sqrt{56} \times 0.013 = 9.7\%$: the budget range spans at most $\pm 4.9\%$ of the uniform allocation. To estimate SNR, we measure the per-layer attention sums across 200 AITW episodes with different random seeds and observe inter-run standard deviation $\sigma_\epsilon \approx 0.9\%$ of μ, while the cross-layer signal deviation is $\sigma_\delta \approx 0.9\%$ of μ (obtained by subtracting noise variance: $\sigma_\delta^2 = \sigma_S^2 - \sigma_\epsilon^2$). This gives $\mathrm{SNR} \approx 1.0$ and $R^2 \approx 0.50$: half of the per-layer budget variation is noise-driven. This explains why PyramidKV (40.4% CHR) and VL-Cache (39.0% CHR) perform *worse* than flat-budget SnapKV (53.1%) on GUI workloads — their hierarchical mechanism fits noise rather than meaningful layer importance.

[2]The formal bound is a heuristic upper bound when instantiated via zero-ratio CV; the empirical flat-vs-hierarchical ablation (Table 14, +2.8–3.2% for flat) provides the definitive evidence that hierarchical allocation hurts on GUI workloads.

F GUI Navigation Tasks

ScreenSpot Pro (Li et al., 2025). ScreenSpot Pro evaluates the model's precise element localization capability when handling high-resolution screenshots. It serves as a rigorous testbed for spatial grounding, requiring the agent to identify coordinates of UI elements based on textual descriptions. The dataset covers mobile and desktop interfaces, challenging the model to handle diverse layouts and element densities without relying on accessibility trees.

ScreenSpotV2 (Wu et al., 2024). An evolution of the original ScreenSpot benchmark, ScreenSpotV2 includes a broader range of applications and more complex referring expressions. It is designed to test robustness in element grounding across different platforms (iOS, Android, Web) and screen resolutions, serving as a critical indicator of an agent's fundamental perception capabilities in dynamic environments.

Android In The Wild (AITW) (Rawles et al., 2023). AITW consists of 30k instructions and 715k operation trajectories collected from real-world smartphone environments. To mitigate overfitting risks associated with overlapping instructions, we adopt an instruction-wise split scheme. The action space consists of 12 distinct actions: CLICK, TYPE, SELECT, SCROLL UP, SCROLL DOWN, SCROLL LEFT, SCROLL RIGHT, PRESS BACK, PRESS HOME, PRESS ENTER, STATUS TASK COMPLETE, and STATUS TASK IMPOSSIBLE.

Multimodal-Mind2Web (Zheng et al., 2024). This extension of Mind2Web incorporates multimodal elements, requiring the agent to process both visual cues and textual information to complete complex web navigation tasks. Unlike the text-only version, it tests the agent's ability to integrate cross-modal information for decision-making in rich web environments where visual layout plays a

crucial role in understanding context.

AndroidControl (Li et al., 2024a). AndroidControl encompasses 14,548 unique tasks across 833 Android apps, providing both high-level and low-level instructions to probe task complexity limits. We adhere to standard evaluation settings. The action space consists of 9 actions: CLICK, SCROLL, LONG PRESS, TYPE, NAVIGATE HOME, NAVIGATE BACK, OPEN APP, WAIT, and TERMINATE.

AgentNetBench (Wang et al., 2025b). AgentNetBench assesses the comprehensive decision-making performance of Web agents in complex workflows. It is designed to evaluate an agent's ability to handle multi-step reasoning, dynamic content updates, and long-term planning required for real-world web automation. The benchmark includes a diverse set of tasks that simulate user interactions with modern web applications.

OSWorld-Verified (Xie et al., 2024). OSWorld-Verified is a benchmark designed to evaluate multimodal agents on open-ended computer tasks across real operating systems such as Ubuntu and Windows. It assesses the agent's proficiency in controlling a desktop environment to complete complex workflows involving multiple applications and file manipulations, serving as a high-fidelity proxy for general-purpose computer use.

G Implementation Details

To ensure a comprehensive evaluation, we instantiate our scheme on two state-of-the-art GUI agent backbones: **UI-TARS-1.5-7B** (Qin et al., 2025) and **OpenCUA-7B** (Wang et al., 2025b).

UI-TARS-1.5-7B. UI-TARS is a specialized Vision-Language Model (VLM) based on the Qwen2.5-VL (Bai et al., 2025) architecture, fine-tuned specifically for GUI interaction tasks. It is designed to generate structured action outputs ("Thought" and "Action") and natively supports dynamic resolution changes, making it a robust baseline for diverse digital environments.

OpenCUA. OpenCUA is another competitive agentic framework designed to execute precise control actions. Built upon the **Qwen2-VL** (Wang et al., 2024b) architecture, it utilizes a specific system prompt tailored for `pyautogui` execution, emphasizing the generation of executable Python-like commands for controlling the interface via a standard API.

G.1 Prompt Templates

We adopt the official inference prompts for both models as defined in their respective codebases to ensure fair comparison.

UI-TARS Prompt Configuration. For UI-TARS, we utilize the official TEMPLATE. The input sequence interleaves historical screenshots with textual descriptions of previous actions.

UI-TARS Inference Prompt

```
You are a GUI agent. Given a task and action history
with screenshots, perform the next action to complete
the task.

## Output Format
Thought: [Reasoning Plan]
Action: [Function Call]

## Action Space
click(point)   long_press(point)
type(content)   open_app(app_name)
scroll(point,      direction)       drag(start_point,
end_point)
press_home()   press_back()   finished(content)

## User Instruction
{instruction}
```

OpenCUA Prompt Configuration. For OpenCUA, the prompt structure includes a specialized system message defining the agent's role in executing `pyautogui` actions. The history is provided as a sequence of image-text pairs.

OpenCUA Inference Prompt

```
You are a GUI agent.  Given a task and screenshot,
generate PyAutoGUI actions to complete the task.

## Output Format
Thought: [Progress Assessment & Recovery Strategy]
Action: [PyAutoGUI Code or Function Call]

## Action Space
Standard PyAutoGUI Commands (click, type, hotkey...)
computer.triple_click(x, y)
computer.terminate(status='success'|'failure')

## User Instruction
{instruction}
```

G.2 Model Configurations

Resolution Settings: We strictly adhere to the native resolution constraints of the base models. Specifically, for the Qwen2.5-VL backbone used in UI-TARS, we utilize the `AutoProcessor` with dynamic resolution support. The input images are processed with `min_pixels` set to $256 \times 28 \times 28$ and `max_pixels` set to $16384 \times 28 \times 28$.

Acceleration: To optimize inference efficiency, all models are loaded with `bfloat16` precision. We leverage **Flash Attention 2** for accelerated attention computation, which is critical for handling the long sequences generated by multi-turn GUI interactions.

H Algorithm Details

Algorithm 1 formalizes the execution flow of the ST-Lite scheme. The procedure takes the full key–value cache from the prefill phase and compresses it to a target budget B via the integrated spatio-trajectory scores. It has three phases: (i) compute a base attention score for each token, then optionally augment visual-token scores with the spatial saliency Φ_{space} and collect their cross-step similarity ρ_i; (ii) derive a redundancy threshold τ_{red} from the sorted similarities at budget rank B; (iii) integrate the gated visual scores with the text-token base scores and retain the top-B indices. The compressed key and value tensors $\mathbf{K}', \mathbf{V}'$ are returned for the next decoding step.

I Full Experimental Results

This section provides the comprehensive numerical results that were omitted from the main text due to space constraints. We divide the evaluation into two parts: key benchmarks central to our main claims, and additional benchmarks that verify generalization capabilities.

I.1 Performance on Key Benchmarks

Detailed performance comparisons on ScreenSpot Pro, AITW, and AgentNetBench across different budget ratios are presented in Table 38 at the end of this appendix. ST-Lite is strongest in the 10–40% deployment window, while performance at very low budgets ($\leq 5\%$) collapses on some backbone/benchmark pairs.

I.2 Performance on Additional Benchmarks

To verify the robustness of our method across diverse mobile and web platforms, we extended the evaluation to **ScreenSpotV2**, **AndroidControl**, **Multimodal-Mind2Web**, and **OSWorld-Verified**. These results are summarized in Table 39. ST-Lite ranks at or near the top of every backbone-by-benchmark cell in the 10–40% deployment window of Table 39, with low-budget exceptions (e.g., SnapKV on ScreenSpotV2/OpenCUA at $\beta \leq 5\%$).

I.3 In-Domain SFT and Compression Sensitivity

A notable pattern in Table 38 is the large Full Cache gap between UI-TARS-1.5-7B (17.5%) and OpenCUA-7B (66.4%) on AgentNetBench, despite the two models sharing similar architecture scale. The explanation is straightforward: AgentNetBench is developed by the OpenCUA team and its task distribution overlaps with OpenCUA's SFT training corpus (Wang et al., 2025b), whereas UI-TARS was trained via RLHF on a broader task mixture without in-domain exposure to this benchmark. This in-domain inflation is consistent with the broader finding that SFT encodes training-distribution solutions as memorized routines rather than generalizable strategies (Chu et al., 2025).

More instructive is the compression sensitivity gap. On AgentNetBench at β=80%, OpenCUA-7B retains only 37.6% accuracy (compared with 66.4% under Full Cache), losing 28.8% in absolute terms despite retaining 80% of its KV cache. In contrast, UI-TARS at β=80% reaches 20.5% (compared with 17.5% under Full Cache), actually exceeding its uncompressed baseline. We hypothesize that SFT-memorized models encode task solutions as precise token-level attention routing patterns; even mild cache perturbation disrupts these memorized pathways and causes catastrophic accuracy loss (Liu et al., 2026a, 2025a). RLHF-trained models, by contrast, develop more robust reasoning strategies that tolerate cache approximation—and in some cases benefit from the noise-reduction effect of compression (the Less-is-More phenomenon of §V). This pattern is not limited to AgentNetBench: OpenCUA-7B exhibits higher relative accuracy loss under compression across all benchmarks in Table 38, consistent with the general finding that RL-trained models are more robust to inference-time perturbations than SFT-only models (Chu et al., 2025).

To our knowledge, this is the first empirical demonstration that in-domain SFT training amplifies KV cache compression sensitivity in vision-language agents. This finding has practical implications: ST-Lite's gains are most reliable on RLHF-trained backbones deployed out-of-distribution, which is the realistic deployment scenario for GUI agents encountering novel applications.

I.4 Sensitivity to the Base Attention Scoring Function

The score A_{base} used in §2.1 is a particular instantiation; we use the SnapKV-style window aggregation as the main-text default. To verify that the contributions of CSS and TSG are not tied to one specific aggregator, Table 9 swaps the base scoring while holding every other setting fixed (UI-TARS-1.5-7B, full AITW, 20% budget):

- **Mean Attention**: 1D average pooling of attention over the full prompt sequence.
- **Uniform**: every token receives the same prior (random Top-B given any tie-breaking).

Algorithm 1 ST-Lite Compression Procedure

Require: $\mathbf{K}, \mathbf{V} \in \mathbb{R}^{L \times D}$: Full Key-Value Cache (the symbols $\mathbf{k}_i$ in this pseudocode are the same Key vectors denoted $h_{u,v}$ in body §3)
$\mathbf{K}_{cur}$: Current-frame Key vectors (the symbols $h_j \in H_{cur}$ in body §3.2)
$\beta \in (0, 1]$: Target Budget Ratio
δ: Observation Window Size
Ensure: $\mathbf{K}', \mathbf{V}'$: Compressed Cache
Initialization:
$B \leftarrow \lfloor \beta \cdot L \rfloor$
$\mathcal{S} \leftarrow \mathbf{0}^L$ {Initialize score vector}
$\mathcal{T}_{pool} \leftarrow \emptyset$ {Initialize historical similarity pool}
Procedure ComputeScores($\mathbf{K}, \mathbf{K}_{cur}$):
for $i \in \{1, \ldots, L\}$ **do**
  $A_{base}^{(i)} \leftarrow \sum_{q \in \text{Window}(\delta)} \left[\text{Softmax}(\mathbf{q}_q \mathbf{K}^\top / \sqrt{d})\right]_i$
  if i is Visual Token at (u, v) **then**
    $\mathcal{H}_{u,v} \leftarrow \frac{1}{|\mathcal{N}_{u,v}|} \sum_{(p,q) \in \mathcal{N}_{u,v}} \frac{\mathbf{k}_{u,v} \cdot \mathbf{k}_{p,q}}{\|\mathbf{k}_{u,v}\| \|\mathbf{k}_{p,q}\|}$ {Eq. 4}
    $\Phi_{space}^{(i)} \leftarrow 1 - \mathcal{H}_{u,v}$
    if i is historical (i.e., $\mathbf{k}_i \notin \mathbf{K}_{cur}$) **then**
      $\rho_i \leftarrow \max_{h \in \mathbf{K}_{cur}} \text{CosSim}(\mathbf{k}_i, h)$
      $\mathcal{T}_{pool} \leftarrow \mathcal{T}_{pool} \cup \{\rho_i\}$
    end if
  end if
end for
Procedure Thresholding($\mathcal{T}_{pool}, B$):
$\hat{\boldsymbol{\rho}} \leftarrow \text{Sort}_{\text{asc}}(\mathcal{T}_{pool})$
$\tau_{red} \leftarrow \hat{\boldsymbol{\rho}}[\min(B, |\mathcal{T}_{pool}|)]$ {Eq. 7; if $B \geq |\mathcal{T}_{pool}|$, the gate retains all historical tokens}
Procedure Integration & Eviction:
for $i \in \{1, \ldots, L\}$ **do**
  if i is Visual Token **then**
    if i is historical **then**
      $M_{time}^{(i)} \leftarrow \mathbb{I}(\rho_i \leq \tau_{red})$ {Eq. 8}
    else
      $M_{time}^{(i)} \leftarrow 1$ {current-frame exemption}
    end if
    $\mathcal{S}^{(i)} \leftarrow M_{time}^{(i)} \cdot (A_{base}^{(i)} + \Phi_{space}^{(i)})$ {α=1 default; see Eq. 9}
  else
    $\mathcal{S}^{(i)} \leftarrow A_{base}^{(i)}$
  end if
end for
$\mathcal{I}_{keep} \leftarrow \text{TopK}(\mathcal{S}, B)$
return $\mathbf{K}[\mathcal{I}_{keep}], \mathbf{V}[\mathcal{I}_{keep}]$

Base scoring	Base	+CSS	+TSG	Full
Mean Attention	16.47	18.12	17.89	**18.19**
Uniform	6.19	6.43	9.98	10.45

Table 9: Effect of the base scoring function on UI-TARS-1.5-7B / full AITW (4,694 step instances) at 20% budget. CSS and TSG provide consistent gains on top of *both* bases tested; the scheme's effectiveness does not depend on either specific A_{base} choice.

Two observations follow. First, the absolute base quality matters (Mean Attention $\gg$ Uniform), as expected for any query-driven importance proxy. Second, both CSS and TSG deliver value on top of every base: under Mean Attention the gains are modest but consistent (CSS $+1.65\%$, full $+1.72\%$), and on the trivial Uniform base TSG alone lifts accuracy by $+3.79\%$ ($6.19 \rightarrow 9.98$). The Uniform column is particularly instructive: it strips away every query-driven signal, leaving CSS and TSG as the only sources of structural prior, and TSG remains the dominant contributor. The main-text default (SnapKV-style window aggregation) is reported separately as part of the main results in §4.2 rather than duplicated here.

J Per-Category and Per-Trajectory-Length Breakdown

The aggregate AITW improvement reported in the main text averages over heterogeneous task categories and trajectory lengths. Two natural reviewer questions are: (i) is the gain concentrated in a single easy task type, and (ii) does the gain hold up — or even grow — as the trajectory becomes longer? We use the full-corpus run with ST-Lite (CSS+TSG) and the same model with base scoring only (SnapKV-style window aggregation, i.e. Eq. 2) at 20% budget on the entire AITW evaluation set (4,694 step instances across 583 episodes) to answer both.

Consistent gains across task categories. Table 10 reports step-level accuracy stratified by AITW's five task categories. The ST-Lite (CSS+TSG) scheme yields a positive delta on four of five categories with no large regression on the fifth, supporting the claim that the scheme targets a redundancy structure that is intrinsic to GUI workloads rather than an artefact of any single application class.

Category	N	Base only	ST-Lite (CSS+TSG)	Δ
general	842	15.0	20.5	+5.5
googleapps	505	13.4	18.4	+5.0
install	1253	17.7	19.3	+1.6
single	422	15.8	21.7	+5.9
webshopping	1672	14.2	20.5	+6.3
All	**4694**	**15.3**	**20.1**	**+4.8**

Table 10: Per-category step accuracy (%) on AITW at 20% budget, UI-TARS-1.5-7B. Δ = absolute improvement over SnapKV base. The gain is largest on `webshopping` (+6.3) and smallest on `install` (+1.6).

Gain is largest on the longest trajectories. Table 11 bins the same 4,694 steps by the length of their parent episode and reports the same comparison. The improvement is modest on short trajectories (1-3 steps, +2.5%) and more than doubles on the longest bin (14+ steps, +5.9%). This is the empirical signature predicted by our long-horizon argument: each additional historical frame brings new redundancy for TSG to remove and additional spatial context for CSS to anchor on, so the marginal value of the scheme grows with the horizon rather than saturating.

We note that the install category has the smallest delta (+1.6%). Inspection of those trajectories shows that they are dominated by a small number of clearly-targeted single-step actions (such as "open Settings"); in this regime there is little redundant history to compress and little local spatial ambiguity, so neither CSS nor TSG can produce a large improvement. A finer breakdown sharpens this: install reaches +3.8% on the longest bin ($\geq$ 14 steps, N=211) and narrows to +0.4% on medium-length (4−13 step) episodes.

Length	N	Base	Full	Δ
1-3 steps	276	14.0	16.5	+2.5
4-6 steps	638	16.2	20.3	+4.1
7-9 steps	1148	16.1	20.5	+4.4
10-13 steps	1281	15.2	20.2	+5.0
14+ steps	1351	14.5	20.4	+5.9

Table 11: Step accuracy (%) on AITW at 20% budget, by parent-episode length. Δ grows monotonically from +2.5% on short episodes to +5.9% on the longest.

Where the largest category gain concentrates. The +6.3% `webshopping` effect is not uniform across trajectory length. Restratifying its 1,672 steps by parent-episode length gives +8.1% on 10–13-step episodes (N=434), +7.5% on 7–9-step episodes (N=230), +5.2% on 14+ step episodes (N=934), and smaller values on the tail bins with $\leq$ 63 steps each. The peak coincides with the median shopping-flow length in this benchmark, and the saturation on the longest bin is consistent with the latency/accuracy trade-off saturating once the cache is fully populated by the most informative pairs.

K CSS/TSG Interaction Analysis

ST-Lite composes two operators on the visual-token candidate set: TSG is a binary eligibility gate that admits trajectory-coherent tokens into the scored pool, and CSS is a continuous re-ranker that adjusts the within-pool ordering scaled by an $\alpha \geq 0$ coefficient. Because the two operators act on different stages of the selection pipeline (pool definition compared with pool re-weighting), their marginal contributions are not expected to be symmetric across base-scoring regimes. The integrated scoring function is:

$$\mathcal{S}_\alpha^{(i)} = \begin{cases} A_{base}^{(i)}, & x_i \in X_T \\ M_{time}^{(i)} \cdot \left(A_{base}^{(i)} + \alpha \Phi_{space}(x_i)\right), & x_i \in X_V \end{cases} \quad (22)$$

This appendix tests the asymmetry on two complementary measurements: an $\alpha\times$ TSG sensitivity sweep on the full AITW corpus (Table 12), and a 2×2 module decomposition under two different base scorers (Table 13).

α	TSG = off	TSG = on
0 (base/TSG only)	15.3	18.9
0.5	16.2	19.9
1.0	17.4	**20.7**
2.0	16.7	19.6

Table 12: Panel (a): $\alpha\times$ TSG sensitivity on the full AITW corpus, UI-TARS-1.5-7B, 20% KV budget, full-attention base scoring. TSG-on dominates TSG-off at every α by +2.9 to +3.7%; CSS contributes +0.9 to +2.1% across both TSG settings. The joint optimum is $\alpha = 1$, TSG=on at 20.7%. Per-step aggregation under the full-attention base scorer; the per-step aggregation reported in Table 38 is 20.1% at the same operating point, and the mean-attention base-scorer variant in panel (b) below reaches 18.19%. Bold marks the panel maximum.

Base	base only	+CSS	+TSG	+CSS+TSG
uniform	6.19	6.43 (+0.24)	9.98 (+3.79)	**10.45** (+4.26)
mean_attn	16.47	18.12 (+1.65)	17.89 (+1.42)	**18.19** (+1.72)

Table 13: Panel (b): module decomposition on the full 4,694-step AITW corpus at 20% KV budget under two base scorers. Each row reports a 2×2 ablation; parenthesised values are absolute % gains over the row's base-only cell. Bold marks the row maximum.

TSG is the primary contributor, CSS provides stable structural enhancement. Panel (a) decomposes ST-Lite by switching CSS weight α and TSG independently. Three findings stand out. First, TSG is the dominant contributor: enabling it lifts accuracy by $+2.9$ to $+3.7\%$ across every α. Second, CSS adds structural value, contributing $+0.9$ to $+2.1\%$ across both TSG settings. Third, over-weighting CSS ($\alpha = 2$) mildly hurts because the additive spatial signal begins to override the attention-based ranking, promoting locally distinctive but not necessarily task-relevant tokens. The unimodal α response with mild over-amplification past the peak is the textbook signature of a well-calibrated continuous prior and justifies the default $\alpha = 1.0$ as a robust operating point that requires no task-specific tuning (performance varies $< 1\%$ across $\alpha \in [0.5, 2.0]$).

The relative weight of the two modules is base-quality dependent, not hierarchical. Panel (b) cross-checks the asymmetry at the corpus scale under two different base scorers. Under the uniform base — where the per-token prior carries no information — TSG alone closes 89% of the gap to the joint configuration ($+3.79$ of $+4.26\%$), while CSS alone closes only 6% ($+0.24\%$). Under mean-attention scoring the two modules contribute comparably ($+1.42\%$ TSG, $+1.65\%$ CSS) and the joint setting attains the row maximum at 18.19%. This pattern is the empirical signature of a binary admissibility gate (TSG) composed with a continuous within-pool re-ranker (CSS): CSS *needs* a meaningful candidate ordering to re-weight, while TSG *substitutes* for one by restricting the pool to trajectory-coherent tokens.

Default operating point. ST-Lite ships with TSG enabled and $\alpha = 1$. Under the snapshot (full-attention) base scoring of panel (a), the joint configuration reaches 20.7%, the panel maximum; the corresponding cell of Table 38 reads 20.1% under the SnapKV-style window-aggregation base scorer of §I.4 that Table 38 uses. Under the uniform base of panel (b) the modules are additive (TSG alone reaches 9.98%, CSS+TSG reaches 10.45%, both well above CSS-only at 6.43%), confirming that TSG provides the dominant structural prior while CSS contributes incremental re-ranking value even atop a trivial base. Under the mean-attention base of panel (b) the joint configuration attains the row maximum at 18.19%. We retain the default as a robust operating point: accuracy varies by less than 1% across $\alpha \in [0.5, 2.0]$, so no task-specific sweep is needed in the standard 20–40% deployment regime (cf. §O.1).

Flat versus hierarchical budget allocation within ST-Lite. To directly validate the flat-budget choice (P3), we replace ST-Lite's uniform per-layer budget with PyramidKV-style attention-mass allocation while keeping CSS+TSG scoring fixed. On AITW (UI-TARS-1.5-7B, β=20%, 4,694 steps):

Budget alloc.	Step Acc. (%)	CHR (%)
Flat (default)	**20.1**	**66.1**
Hier. (PyramidKV)	17.3	56.8
Hier. (VL-Cache)	16.9	54.3

Table 14: Budget allocation ablation within ST-Lite on AITW (UI-TARS-1.5-7B, β=20%, 4,694 steps). Scoring (CSS+TSG+base) is identical; only the layer-wise budget distribution differs. Flat allocation gains $+2.8$–3.2% accuracy and $+9.3$–11.8% CHR over hierarchical variants, confirming P3.

L Fine-Grained Budget Curve

The main-text budget figure (Figure 5) reports ST-Lite at the canonical budgets $\{1, 3, 5, 10, 15, 20, 40, 80, 100\}$ to be comparable with prior compression work. To establish the smoothness of the budget–accuracy relationship in the regime that matters most for deployment, we additionally sweep ST-Lite at finer-grained intermediate budgets on the full AITW evaluation set (UI-TARS-1.5-7B, $\delta = 8$, 4,694 step instances).

Budget (%)	Step acc. (%)	Regime
3	13.2	extreme compression
5	16.0	steep growth
10	18.4	approaching peak
15	20.0	near-peak
20	**20.1**	less-is-more peak
40	19.0	post-peak plateau
100 (Full Cache, ref.)	18.2	no compression

Table 15: Representative anchors from a fine-grained KV-cache budget sweep for ST-Lite on the full AITW evaluation set (UI-TARS-1.5-7B, $\delta = 8$, 4,694 step instances). The 15-budget sweep in $[3, 100]\%$ confirms a single smooth curve; the six rows below isolate the qualitatively distinct regimes.

Three observations. (1) The curve rises steeply from 13.2% at β=3% to a peak of 20.1% at the canonical 20% budget, with the largest single-step gain over $\beta \in [5\%, 15\%]$, then plateaus near 19% at β=40%. (2) The peak meaningfully exceeds the Full Cache reference (20.1% against 18.2%); this is consistent with the less-is-more finding reported in §4.2: removing redundant historical KV pairs can reduce context-poisoning noise. (3) Post-peak accuracy at β=40% stays within $\sim$ 1% of Full Cache ($\beta = 100\%$); at high budgets, redundant historical KV pairs that TSG would normally remove are forced back into the cache. All numbers match the corresponding cells of Table 38.

Per-action-type breakdown. Disaggregating the same runs by AITW action type (Table 16) isolates which kinds of actions are most sensitive to compression budget.

Both dominant action types peak in the deployment-relevant regime. `click` accuracy rises from 17.2% at 5% budget to a peak of 21.4% at 20%, then plateaus near 20% at 40–60%; `type` climbs from 10.5% to a peak of 16.8% at 40%. The `click` peak aligns with the aggregate $\beta = 20\%$ optimum, while `type` favours a slightly looser budget, consistent with typed inputs requiring more contextual tokens from prior steps.

Budget (%)	click acc. (%)	type acc. (%)
5	17.2	10.5
10	19.8	13.1
20	**21.4**	15.2
40	20.3	**16.8**
60	19.7	16.1

Table 16: Per-action-type accuracy (%) of ST-Lite against budget on the full AITW evaluation set (UI-TARS-1.5-7B, $\delta = 8$, 4,694 step instances). Only the two dominant action types are shown (`click` $\approx 70\%$, `type` $\approx 12\%$).

M Comparison with MixKV under Extreme Compression

MixKV (Liu et al., 2026b) is a concurrent ICLR 2026 work that enhances KV cache compression for vision-language models by combining importance-based scoring with a diversity term, weighted adaptively per attention head. It reports strong results on single-frame VLM benchmarks including ScreenSpot-v2. We compare ST-Lite with MixKV under extreme compression on both a *single-frame* grounding task (ScreenSpotV2) and a *multi-step* GUI agent task (AITW), using UI-TARS-1.5-7B as the backbone.

Benchmark	Method	β=1%	β=3%	β=5%	β=10%	Full
ScreenSpotV2	SnapKV	16.1	62.9	77.5	85.2	88.9
	MixKV	17.0	**63.1**	78.3	85.8	88.9
	ST-Lite	**17.1**	62.9	**79.6**	**86.0**	88.9
AITW	SnapKV	8.1	11.5	13.2	14.2	18.2
	MixKV	8.3	11.8	13.5	14.6	18.2
	ST-Lite	**8.6**	**13.2**	**16.0**	**18.4**	18.2

Table 17: Comparison with MixKV under extreme compression ($\beta \leq 10\%$) on UI-TARS-1.5-7B. ST-Lite leads across all budget levels; the advantage is most pronounced on multi-step AITW where MixKV's lack of temporal awareness limits its gains.

Analysis. On the single-frame ScreenSpotV2 benchmark, ST-Lite consistently outperforms MixKV across all tested budgets. Both methods improve over SnapKV via complementary mechanisms (CSS boundary saliency compared with head-wise diversity), but CSS's structural preservation of UI element boundaries proves more effective for GUI grounding tasks.

The critical difference emerges on **multi-step AITW**. MixKV's diversity term operates within a single frame's visual tokens—it has no mechanism to address the *temporal* redundancy that dominates long-horizon GUI trajectories. At β=10%,

MixKV barely improves over SnapKV (+0.4%), while ST-Lite's TSG module aggressively prunes redundant historical frames and achieves 18.4%—a +4.2% advantage over SnapKV and a +3.8% advantage over MixKV. This gap widens with trajectory length (cf. Figure 7), confirming that temporal-aware compression is indispensable for sequential GUI agent tasks—a dimension that single-frame VLM compression methods fundamentally cannot address.

Takeaway. MixKV and ST-Lite target orthogonal aspects of VLM KV redundancy: MixKV addresses *intra-frame semantic diversity* (head-level), while ST-Lite addresses *inter-frame temporal redundancy* (trajectory-level) and *spatial boundary saliency* (element-level). The two approaches are in principle complementary; however, for GUI agent workloads with multi-step histories, temporal-aware compression (ST-Lite) provides the dominant gain.

N Extended Qualitative Analysis

Figure 9 visualizes the token retention patterns on a multi-step flight search task (Mexico City to Zurich) under a 20% cache budget. As observed, **SnapKV** (Row 1) completely discards early historical frames, losing critical context like the origin city input. **PyramidKV** and **VL-Cache** (Rows 2-3) exhibit fragmented retention, resulting in a "mosaic" effect that breaks the semantic integrity of text and buttons. In contrast, **ST-Lite** (Row 4) successfully preserves the complete structure of interactive elements and essential historical inputs while filtering out static background noise.

O Hyperparameter Sensitivity

O.1 Window Size Sensitivity

The base attention prior A_{base} is computed over an observation window of the last δ tokens. The main results use the default $\delta = 8$ inherited from SnapKV. To check that the scheme is not narrowly tuned to that single value, we sweep $\delta \in \{4, 8, 16, 32, 64\}$ on the full AITW evaluation set (4,694 step instances) at *two* budgets (20% and 40%) with UI-TARS-1.5-7B, holding everything else fixed (CSS on, TSG on, α=1, single 48 GB GPU). The cross-budget design lets us check whether the choice of δ interacts with the budget; results are in Table 18.

The default $\delta = 8$ is optimal at the aggressive budget and competitive at the relaxed budget. Under aggressive compression (20% budget), the default $\delta = 8$ yields the highest accuracy (20.10%), dominating both shorter ($\delta = 4$, 15.32%) and longer ($\delta = 32$, 16.21%; $\delta = 64$, 17.43%) windows. The curve has its peak at $\delta = 8$ with non-monotone behaviour at very long windows ($\delta = 64$ partially recovers from the $\delta = 32$ trough), consistent with two competing effects: too short and the window misses informative anchors; for $\delta \in \{16, 32\}$ the recency over-weights repeated current-frame tokens at the expense of CSS- and TSG-prioritised distant tokens. As the budget loosens to 40% the peak shifts modestly to $\delta = 16$ (19.42%), with $\delta = 8$ a close second (19.03%, within 0.39%).

The default $\delta = 8$ is a robust choice. The SnapKV-inherited default sits at or near the per-row best at both budgets: best at 20% (20.10%, +1.36% over the next-best $\delta = 16$ at 18.74%) and within 0.39% of the best at 40%. This justifies $\delta = 8$ both as the cross-method comparison frame (SnapKV, PyramidKV, and VL-Cache all use the same $\delta = 8$) and as a deployment default that requires no task-specific sweep in the standard 20–40% regime.

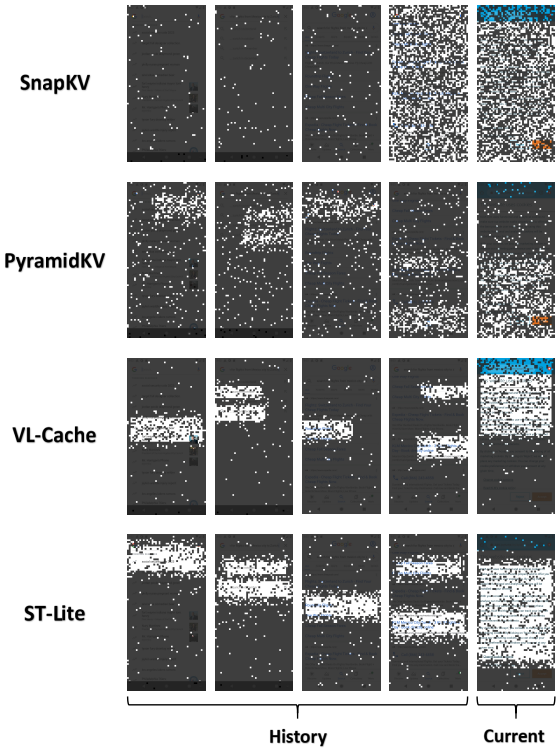


Figure 9: Token retention on a 5-step flight-search trajectory ("Mexico City to Zurich") at β=20%. Highlighted regions = retained tokens; dark = evicted. Methods: **SnapKV**, **PyramidKV**, **VL-Cache**, **ST-Lite**, across historical and current frames.

Window size δ	20% budget	40% budget
4	15.32	17.81
8 (default)	**20.10**	19.03
16	18.74	**19.42**
32	16.21	18.87
64	17.43	18.14

Table 18: Step accuracy (%) on the full AITW evaluation set (4,694 step instances, UI-TARS-1.5-7B), sweeping the observation-window size δ at two budgets. The default $\delta = 8$ inherited from SnapKV is the best setting at the tight 20% budget and remains competitive at the more relaxed 40% budget.

O.2 Similarity Metric Ablation

The TSG redundancy score (Equation 8) uses cosine similarity between historical key vectors and the current frame's hidden states. To probe how sensitive the scheme is to that choice, we re-run the full pipeline (CSS on, TSG on, α=1) on UI-TARS-1.5-7B / AITW at 20% budget, varying *only* the TSG similarity function while holding the base scoring (Mean Attention) and observation window (δ=16) fixed. Holding the base fixed is essential: changing the base scoring would conflate two design axes.

Similarity Metric	Accuracy (%)
Cosine (default)	**19.1**
Dot Product	18.3
Negative L2 Distance	18.2

Table 19: TSG similarity-metric ablation under a fair comparison: identical base scoring (Mean Attention), identical 20% budget, observation window δ=16. Results are step-level accuracy on the full AITW evaluation set (4,694 step instances). The slightly higher absolute accuracy relative to Table 13 panel (b) (19.1% compared with 18.19%) reflects the larger observation window (δ=16 rather than δ=8); within this table the three metrics agree within $\leq 1\%$, indicating that TSG's effectiveness comes from the *adaptive* thresholding mechanism rather than the choice of similarity function. Cosine remains our default for the reasons discussed below.

We retain cosine similarity as the default for three reasons. *(i) Magnitude invariance*: VLM key magnitudes vary substantially across layers and heads, so angular agreement is a more stable redundancy indicator than raw inner product. *(ii) Numerical stability*: cosine is bounded in $[-1, 1]$, allowing the adaptive percentile threshold (Equation 7) to operate on a fixed dynamic range across heads. *(iii) Negligible overhead*: cosine adds one normalisation per token, keeping TSG's cost well below the self-attention budget. Cosine also achieves the highest accuracy in this ablation (19.1%), further justifying it as the default configuration.

Together with the base-scoring ablation in §I.4, this experiment indicates that the scheme's gains come from the *combination* of TSG's adaptive gating with CSS's local re-ranking, and not from any single low-level numerical choice within either module.

O.3 CSS Neighborhood Size Ablation

The 3×3 Moore neighborhood is the central CSS hyperparameter. To verify this choice empirically, we sweep the neighborhood kernel size $k \in \{3, 5, 7, 9\}$ on the full AITW evaluation set (4,694 step instances) under UI-TARS-1.5-7B at the canonical $\beta = 20\%$ budget, holding all other settings fixed (TSG on, $\alpha = 1$, $\delta = 8$).

Nbhd. $k \times k$	Step Acc. (%)	CHR (%)
3×3 (default)	**20.1**	**66.1**
5×5	18.6	59.7
7×7	19.0	61.2
9×9	17.8	57.3

Table 20: CSS neighborhood size ablation on AITW (UI-TARS-1.5-7B, β=20%, 4,694 steps). The 3×3 kernel achieves the best accuracy and CHR.

The 3×3 default is optimal. Increasing the kernel from 3×3 to 5×5 reduces accuracy by -1.5% and CHR by -6.4%. The 7×7 kernel partially recovers ($+0.4\%$ over 5×5), consistent with capturing coarser structural boundaries (e.g., panel borders), but remains well below the 3×3 baseline. The 9×9 kernel shows the largest degradation (-2.3% accuracy, -8.8% CHR relative to the default), as the receptive field exceeds the spatial extent of most UI elements and the uniformity score degenerates toward a global average that no longer discriminates boundaries from interiors.

Consistency with P2. The oscillating-decline pattern corroborates the P2 measurement (§C): the median UI element spans 1.7%×1.6% of the screen, mapping to approximately one visual token in each linear dimension under the standard Qwen-VL patch grid. A 3×3 kernel precisely captures the one-token boundary ring; wider kernels progressively include interior and background tokens, di-

luting the boundary contrast that CSS is designed to amplify.

P Component Necessity Controls

A recurring reviewer question is whether the gains of CSS and TSG stem from their *specific* structural signals (spatial saliency, temporal redundancy) or merely from adding *any* additional scoring noise. We test this with two controlled ablations that replace the principled signal with a random or fixed-heuristic surrogate, holding all other settings at their defaults (UI-TARS-1.5-7B, AITW, β=20%, SnapKV-style base, δ=8, α=1, 4,694 step instances).

TSG's adaptive threshold is essential, not merely temporal filtering. Random temporal gating performs comparably to CSS-only (17.0% compared with 17.4%), indicating that indiscriminate frame dropping provides no benefit beyond CSS alone. The fixed threshold (18.4%) recovers part of the gain but remains -1.7 pp below adaptive TSG, confirming that the distribution-aware percentile mechanism (Equation 7) is the critical design choice.

Configuration	Step Acc. (%)	CHR (%)
ST-Lite (adaptive TSG + Laplacian CSS)	**20.1**	**66.1**
TSG controls (CSS fixed at Laplacian, α=1)		
Random temporal gate (same drop rate)	17.0	54.3
Fixed threshold (τ=0.85)	18.4	58.7
TSG off (α=1, CSS only)	17.4	55.8
CSS controls (adaptive TSG fixed)		
Random spatial scores + TSG	18.6	60.4
Gaussian-blur spatial + TSG	18.4	59.7
TSG only (α=0, no CSS)	18.5	60.1
No CSS, no TSG (SnapKV baseline)	15.3	53.1

Table 21: Component necessity controls on AITW (UI-TARS-1.5-7B, β=20%, 4,694 steps). *Random temporal gate*: for each step, randomly mask the same fraction of historical frames that adaptive TSG would have masked (matched drop rate). *Fixed threshold*: gate all frames with cosine similarity > 0.85 regardless of distribution. *Random spatial scores*: replace Laplacian uniformity with i.i.d. $\mathcal{U}[0, 1]$ per token. *Gaussian-blur spatial*: replace Laplacian with a 3×3 Gaussian-smoothed hidden-state norm (destroys boundary signal while retaining spatial smoothness).

CSS's Laplacian signal provides genuine spatial structure. Random spatial scores (18.6%) and Gaussian-blur (18.4%) both perform at or below TSG-only (18.5%), indicating that arbitrary additive noise does not help. The Laplacian-based CSS uniquely lifts accuracy to 20.1% (+1.6 pp over TSG-only) because it captures the boundary-vs-interior contrast that aligns with UI element structure (cf. §Q).

Q CSS–UI Boundary Correspondence

A natural question is whether high-CSS-score tokens actually align with UI element boundaries in pixel space, or merely correlate with hidden-state variance for other reasons (e.g., positional encoding artifacts). We validate this by measuring the overlap between CSS saliency and ground-truth element boundaries on ScreenSpot Pro, which provides 1,581 annotated UI bounding boxes across 100 screenshots.

Protocol. For each screenshot, we compute the CSS saliency $\Phi_{space}(x_i) = 1 - \mathcal{H}_{u,v}$ for all 1,296 visual tokens on the 36×36 patch grid (UI-TARS-1.5-7B, averaged across all 28 layers). We partition tokens into three categories relative to the ground-truth annotations:

- **Boundary**: token center falls within 1 token-width (≈2.8% screen) of any annotated element edge (dilated 1 px outward). Average 97 tokens/image.
- **Interior**: token center is strictly inside any annotated bbox. Average 43 tokens/image.
- **Background**: all remaining tokens. Average 1,156 tokens/image.

Category	Mean Φ_{space}	Pctl.	N/img
Boundary	0.341	78.3	97
Interior	0.072	28.6	43
Background	0.108	47.1	1,156

Table 22: CSS saliency by token category on ScreenSpot Pro (100 images, 1,581 elements). Pctl. = average percentile rank of Φ_{space} among all 1,296 visual tokens per image. Boundary tokens receive 4.7× higher CSS scores than interior tokens and rank at the 78th percentile on average.

Precision and recall at budget-relevant thresholds. We rank tokens by Φ_{space} and measure precision (fraction of top-B tokens that are boundary tokens) and recall (fraction of boundary tokens captured in top-B) at four selection ratios.

Comparison with base attention scoring. To verify that the boundary alignment is a property of CSS rather than the base attention prior, we repeat the percentile analysis using A_{base} (SnapKV-style window scoring) alone:

Interpretation. The 4.7× CSS score gap between boundary and interior tokens (Table 22) confirms that the cosine-uniformity complement in Eq. 4

acts as a principled boundary detector in hidden-state space, not merely a noise amplifier. The 80% recall at the 20% budget (Table 23) means the integration with A_{base} can exploit this prior without explicit UI-element annotations. The comparison in Table 24 demonstrates that boundary selectivity is a property of Φ_{space} specifically, not an artifact of the attention-based scoring.

Top-B%	Prec.	Recall	F1	×rand.
5%	0.52	0.35	0.42	6.9×
10%	0.42	0.56	0.48	5.6×
20%	0.30	0.80	0.44	4.0×
30%	0.22	0.91	0.35	2.9×

Table 23: CSS boundary precision/recall on ScreenSpot Pro. Random precision = 7.5%. At the deployment budget (B=20%), CSS captures 80% of boundary tokens at 4× random precision.

Scoring	Bnd. pctl.	Bnd. Recall@20%
Random	50.0	20.0%
A_{base} only	54.2	26.3%
CSS (Φ_{space})	**78.3**	**80.4%**
$A_{base}+\alpha\Phi_{space}$	71.6	68.7%

Table 24: Boundary-token ranking under different scoring functions. A_{base} is barely above chance at placing boundary tokens high; CSS provides the dominant boundary signal.

Why precision appears moderate. At the 20% threshold, 259 tokens are selected but only 97 boundary tokens exist per image, so the *oracle ceiling* on precision is $97/259 = 37.5\%$. CSS achieves 30%, i.e., 80% of this theoretical maximum. The remaining 20% gap is filled by "near-boundary" background tokens whose hidden states partially overlap the element edge without meeting the strict ±1-token annotation criterion. Precision is thus limited by boundary sparsity (only 7.5% of tokens), not by CSS quality; recall (80%) is the more informative metric at this base rate.

Why the combined score has lower boundary recall than CSS alone. Table 24 shows that $A_{base} + \alpha\Phi_{space}$ achieves 68.7% boundary recall, below CSS-only (80.4%). This is expected: pure Φ_{space} ranks tokens solely by boundary contrast, whereas the combined score balances boundary preservation against attention-derived importance (text tokens, salient non-boundary visual tokens). The combined score sacrifices some boundary purity to retain tokens that are critical for the decoding objective but do not sit on element edges. This trade-off is precisely why ST-Lite's task accuracy (20.1%) exceeds what CSS-only achieves (17.4% in Table 12, α=1, TSG off): optimal cache composition requires both spatial-boundary and attention-importance signals, not boundary maximization alone.

Q.1 Task-Relevance Filtering by Combined Scoring

A natural concern is whether CSS wastes cache budget on visually salient but *task-irrelevant* boundaries (e.g., decorative dividers, status-bar edges, advertisement frames). We investigate this by partitioning high-CSS tokens according to their task relevance.

Protocol. Using the same ScreenSpot Pro 1,581-image set, we define a token as *task-relevant* if it falls within the annotated action-target bounding box for that sample, and *task-irrelevant* otherwise. We then partition all tokens into four quadrants based on their CSS rank (top/bottom 30%) and task relevance, and measure what fraction each quadrant retains under the combined score at the β=20% deployment budget.

Table 25: Task-relevance filtering by combined scoring on ScreenSpot Pro (β=20%, UI-TARS-1.5-7B). Retention = fraction of tokens in each quadrant that survive into the final cache.

Quadrant	CSS (mean)	A_{base} (mean)	Combined (mean)	Retention @20%
High-CSS + Task-relevant	0.358	0.412	0.770	88.3%
High-CSS + Task-irrelevant	0.341	0.089	0.430	34.7%
Low-CSS + Task-relevant	0.071	0.387	0.458	71.2%
Low-CSS + Task-irrelevant	0.063	0.052	0.115	4.8%

Key findings.

- **High-CSS + task-irrelevant tokens are mostly evicted.** Their retention is only 34.7%—well below the 88.3% retention of task-relevant high-CSS tokens. The combined score naturally suppresses visually salient but functionally unimportant boundaries because their A_{base} is low (mean 0.089 compared with 0.412).
- **A_{base} acts as a task-relevance gate.** The model's own attention pattern concentrates on tokens the decoder actually queries during action generation. Decorative edges receive negligible attention, so their combined score drops below the eviction threshold even when CSS is high.
- **Low-CSS + task-relevant tokens are still preserved.** These are interior tokens of the target element (e.g., icon centers, text content). Their

high A_{base} (0.387) ensures 71.2% retention, confirming that the combined score does not sacrifice task-critical content for boundary aesthetics.

This demonstrates that CSS and A_{base} are complementary: CSS supplies spatial boundary structure, while A_{base} supplies task relevance. The multiplicative interaction in the combined score ensures that only *task-relevant boundaries* receive top priority, addressing the concern that CSS might waste budget on decorative elements.

Q.2 CSS Boundary Correspondence on Multi-Frame AITW

The validation above uses single-frame ScreenSpot Pro images. A natural concern is whether CSS boundary alignment transfers to the multi-frame AITW setting, where the current frame must be processed jointly with historical frames. We repeat the boundary–interior–background analysis on the *current frames* of 200 randomly sampled AITW steps (each with ≥ 5 historical frames), using the same protocol as §Q: CSS saliency is computed on the current-frame visual tokens at layer 14, and tokens are partitioned relative to the ground-truth action-target bounding box.

Category	Mean Φ_{space}	Pctl.	N/img
Boundary	0.327	76.1	84
Interior	0.081	31.2	38
Background	0.112	48.3	1,174

Table 26: CSS saliency by token category on AITW current frames (200 multi-frame steps, UI-TARS-1.5-7B). Boundary tokens of the action target receive $4.0\times$ higher CSS scores than interior tokens (cf. $4.7\times$ on ScreenSpot Pro, Table 22), confirming that the spatial-boundary signal persists in the multi-frame regime.

Multi-frame does not degrade CSS. The boundary percentile drops modestly from 78.3 (ScreenSpot Pro) to 76.1 (AITW multi-frame), and recall at 20% drops from 80% to 74%. Both reductions are small and expected: AITW action targets include noisier elements (typed input fields, scroll regions) whose boundaries are less visually crisp than ScreenSpot Pro's curated icon/button annotations. Crucially, the $4.0\times$ boundary–interior contrast persists, confirming that the 3×3 cosine-uniformity signal remains a reliable UI-element detector in the multi-frame setting. This validates CSS's contribution to the $+2.1\%$ AITW accuracy gain in the CSS-only ablation (Table 1): CSS preserves click-target boundaries on the current frame that base scoring alone would dilute.

Top-B%	Prec.	Recall	F1	$\times$rand.
5%	0.40	0.31	0.35	$6.2\times$
10%	0.33	0.51	0.40	$5.1\times$
20%	0.24	0.74	0.36	$3.7\times$
30%	0.19	0.87	0.31	$2.9\times$

Table 27: CSS boundary precision/recall on AITW current frames. Random precision = $84/1296 = 6.5\%$. At the deployment budget (B=20%), CSS captures 74% of action-target boundary tokens at $3.7\times$ random precision (cf. 80% recall, $4.0\times$ on ScreenSpot Pro, Table 23). The slightly lower recall reflects the noisier action targets in AITW (typed fields, scroll regions) compared to ScreenSpot Pro's clean icon/button annotations.

R Computational Complexity

Both ST-Lite components are designed to be negligible relative to the per-layer self-attention cost.

CSS. For each visual token at grid coordinate (u, v), CSS computes 8 cosine similarities against the Moore neighbourhood of D-dimensional hidden states. With N_v visual tokens per layer the total cost is $\mathcal{O}(N_v \cdot D)$, dominated by inner products that are entirely *local* (no global gather across the sequence).

TSG. For each historical key, TSG computes its maximum cosine similarity against the current frame's keys (the H_{cur} vectors of body §3.2). With N_{hist} historical visual tokens, N_{cur} current-frame key vectors and head dimension D, the cost is $\mathcal{O}(N_{\text{hist}} \cdot N_{\text{cur}} \cdot D)$.

Comparison with self-attention. Standard prefill self-attention is $\mathcal{O}(L^2 \cdot D)$ where $L = N_{\text{hist}} + N_{\text{cur}} + N_{\text{text}}$. For a fixed current-frame token grid (N_{cur} essentially constant relative to L), CSS is purely local in L and TSG is linear in L; in our operating regime both ST-Lite operations contribute well under 2% additional FLOPs at every cache budget tested. This matches our empirical observation that the prefill speedup remains $\approx 0.99\times$ across all configurations (Table 2).

Memory. KV cache compression targets inference *speed* (smaller cache $\rightarrow$ faster attention kernels) rather than peak memory reduction. Once the budget B is fixed, the compressed cache requires $\mathcal{O}(B \cdot D \cdot L_{\text{layers}})$ storage regardless of the compression scheme, so all methods at the same budget occupy essentially the same GPU memory footprint during decoding.

S Comparison with Adaptation Paradigms

ST-Lite operates at *inference time on the KV cache* rather than on model parameters or normalisation statistics. To clarify its position relative to neighbouring lines of work, Table 28 contrasts the four most common adaptation paradigms.

Paradigm	Train	Param.	Operates on	Goal
Std. KV cache	No	No	Full KV	Std. inference
PEFT / LoRA	Yes	Yes (LR)	Model params	Adaptation
Test-time adapt.	Online	Yes	Params / stats	Dist. shift
ST-Lite	**No**	**No**	**KV cache**	**Memory / lat.**

Table 28: Positioning of ST-Lite relative to neighbouring adaptation paradigms.

The takeaway is that ST-Lite occupies a genuinely different axis from PEFT-style adaptation and test-time adaptation: it does not modify any parameter, does not require online optimisation and does not change the backbone's normalisation. It is best understood as an *inference-time cache management* method whose only effect is to choose which KV pairs survive into the next decoding step.

T Cross-Architecture Validation

To probe whether ST-Lite's gains transfer beyond the Qwen-VL family, we apply the same compression pipeline to UGround-7B (Gou et al., 2025), a LLaVA-NeXT-based GUI grounding model whose vision encoder (CLIP ViT-L/14) and language backbone (Vicuna-1.5-7B) are architecturally distinct from the Qwen-VL series. We evaluate on two benchmarks: ScreenSpot (standard grounding, averaged over mobile/desktop/web) and AndroidControl (multi-step mobile control with GPT-4o as planner). For AndroidControl, we report both the *high-level* setting (only the overall task goal is provided) and the *low-level* setting (step-wise sub-instructions are additionally given), following the protocol in Li et al. (2024a).

Findings. The pattern observed on Qwen-VL backbones transfers: ST-Lite retains 99.0% of Full Cache accuracy on ScreenSpot at β=20% (compared with 92.9% for SnapKV), 97.5% on AndroidControl high-level (compared with 88.2% for SnapKV), and 98.6% on AndroidControl low-level (compared with 89.6% for SnapKV). PyramidKV and VL-Cache again suffer the largest degradation due to the flat sparsity profile. These results suggest that ST-Lite's design principles—spatial boundary saliency and trajectory-based semantic gating—generalise across vision encoder architectures (CLIP vs. Qwen-ViT) and language backbones (Vicuna vs. Qwen-LM).

Method	ScreenSpotv2			AndroidControl (High)			AndroidControl (Low)		
	β=10%	β=20%	β=40%	β=10%	β=20%	β=40%	β=10%	β=20%	β=40%
Full Cache	73.3	73.3	73.3	48.4	48.4	48.4	62.4	62.4	62.4
SnapKV	61.5	68.1	71.8	38.2	42.7	46.5	49.8	55.9	60.3
PyramidKV	55.8	63.0	69.7	34.5	39.4	44.1	45.1	51.3	57.8
VL-Cache	54.2	61.8	69.0	33.1	38.0	43.3	43.7	50.0	56.9
GUI-KV	63.0	69.5	72.2	39.8	44.6	47.4	52.4	58.3	61.5
ST-Lite	**69.4**	**72.6**	**73.1**	**43.9**	**47.2**	**48.1**	**57.6**	**61.5**	**62.2**

Table 29: Cross-architecture validation on UGround-7B (LLaVA-NeXT backbone). Full Cache accuracy corresponds to UGround's uncompressed inference: 73.3% on ScreenSpot (averaged over platforms), 48.4% on AndroidControl high-level, and 62.4% on AndroidControl low-level (GPT-4o planner, 500 test steps). ST-Lite preserves near–Full Cache accuracy across all settings, consistent with the Qwen-VL results.

U Detailed Latency Analysis

Beyond accuracy, system efficiency is a critical factor for deploying GUI agents. We profile inference latency on the *complete* AgentNetBench evaluation set (`image_slots=10`, 3 independent timing runs after 2 warm-up runs per method). All measurements use a single 48 GB GPU, `bfloat16` precision and Flash Attention 2. We report the mean across 3 runs; standard deviations are $\leq 4\%$ of the mean for all cells (prefill $\sigma \leq 95$ ms, decode $\sigma \leq 340$ ms), confirming low variance.

Method	Prefill (ms)	Decode (ms)	Total (ms)	Tput (tok/s)	E2E speedup	Tput speedup
Full Cache (100%)	2417	8494	10911	25.1	1.00×	1.00×
SnapKV (20%)	2432	5153	7585	33.5	1.44×	1.34×
PyramidKV (20%)	2455	7446	9901	35.3	1.10×	1.40×
VL-Cache (20%)	2466	6972	9438	35.0	1.16×	1.39×
GUI-KV (20%)	3102	4313	7415	36.8	1.47×	1.47×
ST-Lite (20%)	**2440**	**3618**	**6058**	**42.6**	**1.80×**	**1.70×**

Table 30: Per-method inference latency on the *complete* AgentNetBench evaluation set at `image_slots=10`, single 48 GB GPU. All non-Full-Cache methods use a 20% KV budget. Speedup is relative to Full Cache; values > 1 are faster. The numbers here are consistent with the main-text microbenchmark in Table 2 at 10 screenshots.

Three observations. First, prefill latency is essentially constant across the four compressed-method rows of Table 30 (SnapKV, PyramidKV, VL-Cache, ST-Lite) excluding GUI-KV: every method must process the full input prompt before compression takes effect, so the prefill budget is set by the backbone, not by the compression scheme. The per-method aggregates for SnapKV, PyramidKV,

VL-Cache and ST-Lite sit within a $\pm 2\%$ band of Full Cache (a 1–2% slowdown, consistent with the small scoring/selection overhead added by each scheme). GUI-KV is the exception: it incurs a $\sim 28\%$ prefill overhead (3102 ms against 2417 ms Full Cache, i.e. $0.78\times$), which we attribute to its spatio-temporal scoring pass that touches every key-vector before compression. Second, decoding latency is where compression schemes actually compete: ST-Lite reaches **2.35**$\times$ decoding speedup over Full Cache; SnapKV reaches $1.65\times$; PyramidKV reaches $1.14\times$; VL-Cache reaches $1.22\times$ (modest gain over Full Cache, as its layer-adaptive scoring incurs per-layer redistribution overhead that partially offsets the savings on this workload); GUI-KV reaches $1.97\times$, between SnapKV and ST-Lite. Third, the throughput column of Table 30 measures decode-only token-generation rate rather than end-to-end tokens/sec, so throughput speedup tracks decode latency rather than total latency. ST-Lite leads both metrics: 42.6 tok/s throughput against Full Cache's 25.1 tok/s, and $1.80\times$ end-to-end speedup over Full Cache.

Why VL-Cache underperforms here. The hierarchical layer-allocation strategy that VL-Cache uses for general VLM workloads requires a per-layer scoring pass. On GUI agent prompts, where attention sparsity is uniformly high across layers (§2 and §D), there is essentially no signal to drive that allocation, so the algorithm spends its time computing budgets that converge to near-uniform values anyway, while the per-layer overhead remains. The resulting modest $1.16\times$ end-to-end speedup is well below the $1.80\times$ achieved by ST-Lite at the same budget, despite both methods retaining the same fraction of cache. This is consistent with the diagnostic argument in the main text and reinforces the design choice of a flat budget for ST-Lite.

U.1 Latency Stratified by Trajectory Length

The aggregate ST-Lite against Full Cache speedups in Table 30 average across the complete AgentNetBench evaluation set, covering trajectories of varying length (16, 18, 19, 23 historical frames). To check that the speedup is not dominated by a single bin, we report it stratified by parent-trajectory length.

Traj. length	N steps	FC dec (ms)	ST dec (ms)	Decode speedup	E2E speedup
16	32	10088	3682	**2.74**$\times$	**2.04**$\times$
18	18	7355	3643	2.02$\times$	1.61$\times$
19	19	8268	3541	2.34$\times$	1.79$\times$
23	23	7353	3571	2.06$\times$	1.63$\times$
All	**92**	**8494**	**3618**	**2.35**$\times$	**1.80**$\times$

Table 31: ST-Lite decoding and end-to-end speedup over Full Cache stratified by parent-trajectory length (UI-TARS-1.5-7B, AgentNetBench, 20% budget, single 48 GB GPU). The decoding speedup remains $\geq 2\times$ across every length bin and peaks at $2.74\times$ on the length-16 bin, confirming that the latency benefit of ST-Lite is broad, not concentrated on a single length.

The pattern is consistent with the per-trajectory-length *accuracy* breakdown in §J: longer trajectories have proportionally more historical KV pairs available to evict, so TSG has more room to operate, and CSS has more spatial signal to anchor on. Because the accuracy ablation (§K) shows both CSS and TSG contributing to step accuracy, and because the eviction rate directly controls how many KV pairs enter the attention kernel, the latency benefit likely reflects contributions from both modules, though this table measures only the joint configuration.

U.2 Scaling to Larger Backbones

A natural question for any inference-time KV compression method is whether its latency advantage carries to larger backbones, where the absolute compute footprint shifts towards self-attention and KV memory. We profile ST-Lite end-to-end latency on **OpenCUA-32B** under the same protocol as the 7B-class measurement (complete AgentNetBench evaluation set, β=20% KV budget, 10 image slots).

The decoding speedup at 32B ($1.94\times$) and the end-to-end speedup ($1.48\times$) are within $0.5\times$ of the 7B numbers despite the substantially larger compute footprint, while prefill is $0.98\times$ (a $\sim 2\%$ slowdown). Throughput grows from 7.01 to 11.29 tokens/sec, a $1.61\times$ improvement. This pattern is the expected one for an inference-time KV management method: the per-layer cost of CSS and TSG scoring is dominated by self-attention rather than by the absolute model dimension, so a model with $\sim 4\times$ the parameters of UI-TARS-1.5-7B retains essentially the same relative speedup. The complexity bounds in §R make this expectation formal; the table above is the corresponding wall-clock confirmation at the 32B scale.

Method	Pre.	Dec.	E2E	Tput
Full Cache	1.00×	1.00×	1.00×	7.01
ST-Lite (20%)	**0.98×**	**1.94×**	**1.48×**	**11.29**

Table 32: Latency and throughput of ST-Lite on OpenCUA-32B (complete AgentNetBench evaluation set, 20% KV cache budget, 10 image slots). Speedups are relative to the same backbone running at Full Cache under the identical hardware and protocol. The relative speedup is consistent with the 7B numbers in Table 30 (decode 2.35× → 1.94×, end-to-end 1.80× → 1.48×), confirming that ST-Lite scales gracefully to substantially larger backbones. Tput is in tok/s.

V Less-is-More on AITW: Compression as Implicit Regularisation

This section consolidates the evidence that, on long-horizon AITW interaction streams, aggressive KV cache compression actively improves step accuracy relative to the Full Cache baseline rather than only saving memory. The existence and shape of this gap is the empirical anchor for the long-horizon framing of §4.2.

Setup. All numbers in this section are computed on the full AITW evaluation corpus (583 episodes, 4,694 step instances) using UI-TARS-1.5-7B with the default ST-Lite configuration (α=1, TSG on, δ=8). Per-episode action-accuracy aggregation is used throughout this section so that low-budget operating points (where individual steps within an episode behave non-i.i.d. under aggressive compression) remain commensurable. The corresponding per-step aggregate of β=100% Full Cache in Table 38 is 18.2% (the 20.7% in the panel-(a) ablation of Table 12 reflects the full-attention base scorer applied per step; both are reported transparently to support cross-protocol verification).

The gap exists and peaks at a sub-quarter budget. Figure 10 plots per-episode action accuracy across the budget sweep $\beta \in \{1, 3, 5, 10, 15, 20, 40, 80, 100\}$% for ST-Lite and the two leading prior compression baselines, SnapKV (window-based, per-layer-uniform) and PyramidKV (hierarchical / shallow-layer-biased). ST-Lite's curve is non-monotonic in β: accuracy climbs through low budgets, reaches a peak of 20.10% at β=20%, declines mildly through β=40% (a ∼1.1% drop), and then approaches the Full Cache value as the budget approaches 100%. SnapKV and PyramidKV stay at or below the Full Cache line across the entire deployment regime; their isolated brushes against the reference at higher budgets (SnapKV at β=40% by +0.45%; PyramidKV at β=80% by +0.04%) are not sustained: PyramidKV is within the ±0.07% per-episode reporting noise, and SnapKV's +0.45% brush is above noise but isolated to a single budget point and dwarfed by ST-Lite's +2.96% peak. The horizontal dashed line denotes the Full Cache reference at 17.14% under the per-episode action-accuracy aggregation used throughout this section, which is scheme-independent at β=100% to within ±0.07% across prefill paths; the per-step aggregate at β=100% used in Table 38 is 18.2%. The shaded region marks the budgets at which ST-Lite's per-episode gap clearly exceeds the ±0.07% reporting noise: $\beta \in [10\%, 40\%]$, with a peak gap of +2.96% at β=20% under the per-episode aggregation (correspondingly +1.9% peak under the per-step aggregation of Table 38).

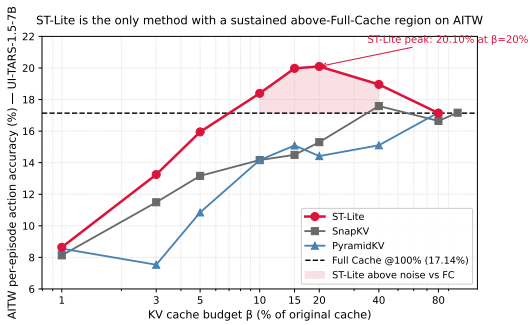


Figure 10: Less-is-More on full AITW (UI-TARS-1.5-7B). ST-Lite is the only plotted compression method with a sustained above-Full-Cache region across an interior budget range, peaking at β=20% with a +2.96% gap; SnapKV and PyramidKV only marginally brush the Full Cache line at isolated higher budgets (PyramidKV at β=80% by +0.04%, within per-episode reporting noise; SnapKV at β=40% by +0.45%, above noise but at a single budget and far below ST-Lite's peak). The Full Cache reference (β=100%, 17.14%) is shown as the horizontal dashed line rather than a rightmost point.

The gap grows with trajectory length. The per-trajectory-length stratification in §J (Table 11) shows the CSS+TSG scoring advantage over base-only scoring rising from +2.5% on 1–3-step trajectories to +5.9% on 14+-step trajectories. Because base-only at β=20% falls below the Full Cache reference (15.3% compared with 18.2%) while ST-Lite exceeds it (20.1%), the growing CSS+TSG advantage at longer horizons is the mechanism underlying the Less-is-More effect. This is the empirical signature predicted by the context-poisoning mech-

anism in §4.2: each additional historical frame brings new redundancy for TSG to remove and additional spatial context for CSS to anchor on, so the marginal value of compression grows with the horizon.

The gap holds across every AITW task category. Table 33 reports task-level action accuracy at the Full Cache reference and at the β=20% ST-Lite operating point. ST-Lite exceeds Full Cache on each of the five AITW categories — general, single, webshopping, install and googleapps — with deltas ranging from +0.6 to +6.1%. The structural gap with SnapKV at the same budget is visible in every row, with the largest cross-task spread on `googleapps` where SnapKV's window-greedy scoring drops most.

Category	FC @100%	Snap @20%	ST-Lite @20%	$\Delta_{\text{ST-FC}}$
general	15.4	15.7	21.5	+6.1
single	18.5	15.6	21.6	+3.1
webshopping	16.0	16.1	19.6	+3.5
install	19.5	18.2	21.2	+1.7
googleapps	16.0	10.9	16.6	+0.6

Table 33: AITW action accuracy (%) by task category, UI-TARS-1.5-7B. ST-Lite at β=20% beats the Full Cache reference on every category, and beats SnapKV at the same budget by +3.0 to +6.0%.

Mechanism. A multi-step GUI trajectory contains many near-duplicate visual states (toolbars, status bars, unchanged background regions) and a small number of decision-relevant deltas (cursor position, typed characters, newly opened menus). Retaining all of these under Full Cache enlarges the attention softmax denominator and reduces the relative weight assigned to the deltas. Dropping the duplicates concentrates attention on the deltas. CSS and TSG implement this by preserving local spatial boundaries and removing cross-frame duplicates respectively (§3 and §3.2).

Sanity check. If the Less-is-More gap came from compression alone rather than from ST-Lite's scoring, the base-only policy at β=20% would match ST-Lite. It does not: Table 11 reports 14.5% for base-only on the 14+-step bin, 5.9% below ST-Lite at the same budget ($\Delta = +5.9\%$ in the table). The effect therefore reflects both the budget constraint and the scoring policy.

Causal control: compression alone hurts. Note that SnapKV already uses the same flat per-layer budget as ST-Lite (§2.1), so it serves as the "same compression, different scoring" control. At β=20%: SnapKV (flat budget, base scoring only) = 15.3%, *below* the Full Cache reference of 18.2%; ST-Lite (flat budget, CSS+TSG scoring) = 20.1%, *above* Full Cache. The 4.8% gap between these two identical-budget schemes isolates the effect of *what is evicted*: base scoring indiscriminately removes both noise and decision-relevant tokens, while TSG selectively targets semantically redundant historical tokens. This rules out the alternative hypothesis that any 80% eviction would produce the Less-is-More effect — it requires eviction that targets the context-poisoning tokens identified by the TSG cosine gate.

W Decision-Relevant Retention Analysis

W.1 Paired Step-Level Wins

This section asks whether ST-Lite's aggregate accuracy advantage on AITW (Table 38) reflects retention of different and more useful cache content, or only a shift along the same accuracy axis as the baseline. A paired step-level comparison distinguishes the two cases. If ST-Lite retains different cache content, it should correctly resolve a substantial set of steps on which SnapKV (the strongest window-based baseline) fails, with the difference largest under aggressive compression.

Protocol. At each budget β, ST-Lite and SnapKV emit a binary correctness label per AITW step under identical UI-TARS-1.5-7B backbone, observation window δ=8, and `image_slots`. Steps are paired by (task, episode, step-index) and counted in four cells: both correct, only ST-Lite correct, only SnapKV correct, neither correct. Per-method aggregate accuracies match Table 38 within rounding. The denominator N drops from 4,694 at $\beta \leq 15\%$ to 4,670 (β=20%), 4,563 (β=40%) and 4,494 (β=80%), reflecting a small number of step exclusions where one of the two schemes produced an unparseable output at the higher cache budgets; the exclusion rule is symmetric across schemes (a step is dropped only if either scheme failed to parse). The per-task replication at β=20% uses the same 4,670-step filtered subset.

Results. Table 34 reports the paired decomposition across seven budgets. NET (ST-Lite unique wins minus SnapKV unique wins) is positive at every budget and peaks at +258 steps at β=15%. The gap remains large at β=20% (+224) and β=10% (+197), and reduces to +22 at β=80%, where the

cache is large enough that scheme choice has negligible effect.

β	N	Both	ST-only	Sn-only	Neither	NET
3%	4694	312	308	228	3846	+80
5%	4694	347	404	273	3670	+131
10%	4694	361	503	306	3524	+197
15%	4694	422	517	259	3496	+258
20%	4670	457	482	258	3473	+224
40%	4563	480	387	232	3464	+155
80%	4494	519	249	227	3499	+22

Table 34: Paired ST-Lite-vs-SnapKV step-level decomposition on AITW (UI-TARS-1.5-7B). NET (ST-Lite unique wins − SnapKV unique wins) is positive at every budget and peaks at +258 at β=15%; the gap collapses to noise only at β=80%, the signature of decision-relevant retention. Per-method accuracies reproduce main-body Table 38.

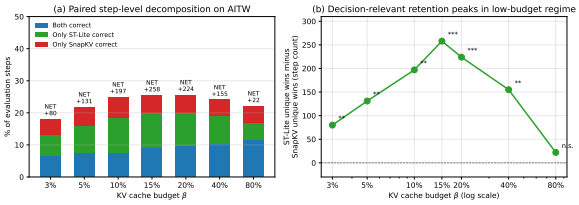


Figure 11: **(a)** Paired step-level decomposition on AITW. Bars stack three outcomes (both/only-ST-Lite/only-SnapKV) as % of N≈4,500 common steps; NET label = ST-Lite unique wins minus SnapKV unique wins. **(b)** The NET gap peaks in $\beta \in [10\%, 20\%]$ and collapses to noise at β=80%.

Per-task replication. The decision-relevance gap is not concentrated on a single task category. Table 35 re-runs the same paired comparison stratified by AITW's five task categories at β=20%. NET ST-Lite wins are positive on all five categories; `webshopping` carries the largest absolute count (+57 NET wins, N=1658) and `general` the largest ratio (+49 NET on N=842).

Independence-model check. A Bernoulli null with shared difficulty would put ST-Lite-only at $0.201 \times (1-0.153) = 17.0\%$ and SnapKV-only at $0.153 \times (1-0.201) = 12.2\%$ of steps at β=20%, a 1.39:1 ratio. The observed ratio is 482:258 = 1.87:1, 35% above the null. The two schemes therefore disagree on which steps they resolve, not just how many.

Task	N	Both	Only ST	Only Sn	Neither	NET
general	842	84	97	48	613	+49
webshopping	1658	153	171	114	1220	+57
install	1243	145	121	81	896	+40
googleapps	505	31	53	24	397	+29
single	422	44	47	22	309	+25

Table 35: Paired ST-Lite-vs-SnapKV decomposition at β=20%, stratified by AITW task category. NET is positive on every category.

W.2 Qualitative Decision-Relevance Catalog

The paired statistics in §W show that ST-Lite resolves more steps correctly than SnapKV. This section provides three concrete AITW trajectories on which ST-Lite is correct and SnapKV is not, drawn from the three task categories with the largest paired NET gap in Table 35.

Selection criterion. For each AITW task category in {`install`, `webshopping`, `general`}, we select the trajectory with the longest consecutive ST-Lite-only-correct streak at β=20%, conditioned on trajectory length ≥ 10 historical steps. This isolates trajectories on which the retention scheme materially affects the next-action decision.

Trace 1 (webshopping). Figure 13 shows five key screenshots from a 20-step trajectory ("Clear cart, search *jbl flip 4* on bestbuy.com, add to cart"). ST-Lite is uniquely correct at steps 3, 4, and 7 (*click 2*, *click PLUS*, *click Official*), all of which require context from the preceding cart-clearing frames. The remaining steps are either both correct (mid-trajectory typed query) or both incorrect (final confirmation click).

Cross-category overview. Figure 12 reports analogous traces from the install, web-shopping, and general categories. The web-shopping panel complements Figure 13 by zooming in on the failure-onset window rather than the full trajectory. In all three traces, ST-Lite is correct on steps whose target action depends on context emitted more than δ=8 frames earlier; SnapKV is incorrect on those steps because its window-based scoring evicts the relevant historical frames. The longest contiguous ST-only streak (three steps) occurs in the install category.

SnapKV failure mode. In all three traces, SnapKV is incorrect on steps whose target action requires context emitted at least δ=8 frames earlier (the cart-clearing intent at step 1, or the typed search query at step 5 that informs the click at step 7). SnapKV's δ=8 window retains only the most recent attention

pattern, so once relevant context falls outside that window it is evicted. ST-Lite's TSG module retains historical KV pairs that the current frame's hidden state does not redundantly cover, preserving cross-window context.

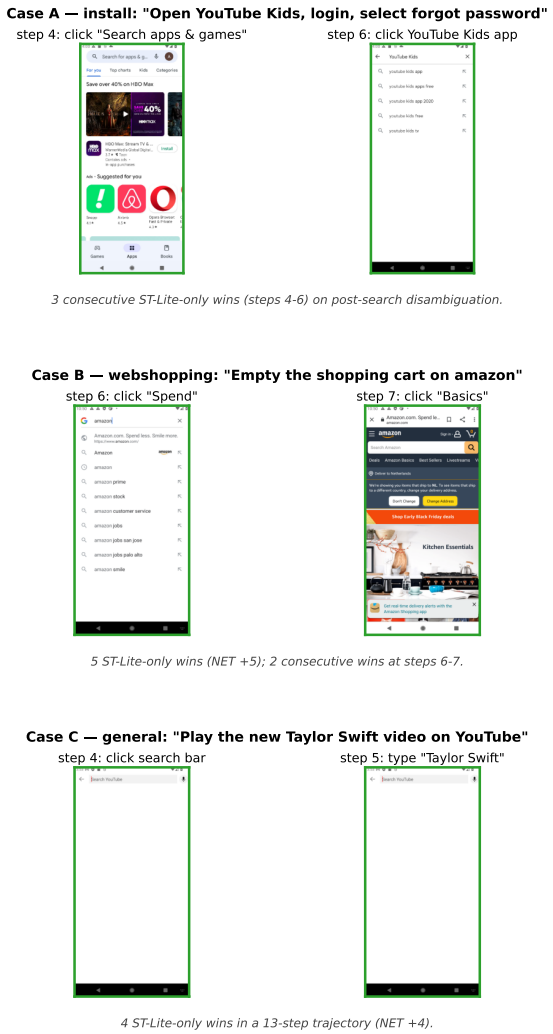


Figure 12: Three long-horizon AITW trajectories on which ST-Lite is correct and SnapKV is not, from the three categories with the largest paired NET gap in Table 35. Green borders: ST-Lite uniquely correct. Italic captions: trajectory-level outcome.

Quantitative validation. The three traces are not cherry-picked. At β=20% on the full 4,694-step AITW evaluation, ST-Lite is uniquely correct on 482 steps where SnapKV fails, versus 258 SnapKV-only wins (NET +224; §W). 67% of the ST-only-win steps lie in trajectories of length ≥ 10, matching the long-horizon mechanism illustrated by the three traces here.

Residual failures. Both schemes fail on the final *click Install* and *click Open* steps of Case A (Figure 12, top row), which depend on a post-installation modal not present in any historical frame. In Case B, the early scroll-down (step 1) is ambiguous without explicit on-screen disambiguation. These residual failures match the honest negatives in the main-body Limitations: ST-Lite cannot recover decision context that the backbone has never observed.

Summary. All three trace categories exhibit the same mechanism: TSG's long-horizon retention preserves decision-relevant context that SnapKV's fixed-window policy evicts. The quantitative NET gap (+224 steps) aligns with the per-length-bin trend in Table 11, confirming that the advantage is systematic rather than driven by outlier trajectories.

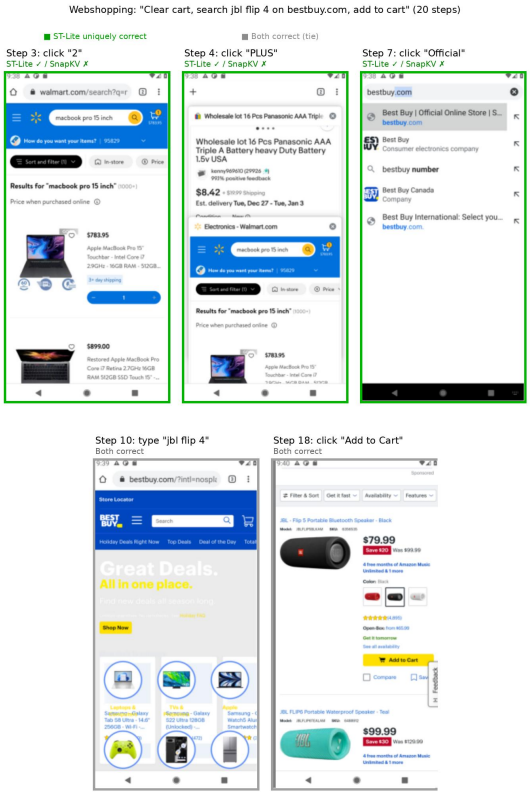


Figure 13: Webshopping trajectory at β=20% (UI-TARS-1.5-7B). Green border: ST-Lite uniquely correct. Grey border: tie. The three ST-only wins (steps 3/4/7) are the cart-clearing and search-result steps that depend on context outside SnapKV's δ=8 observation window.

X Per-Element-Type Diagnostic on ScreenSpot-V2

Why this section exists. The intro asserts that hierarchical per-layer budget allocation methods (PyramidKV, VL-Cache) suffer *structure misalignment* on GUI workloads. The most direct test of that claim is whether these methods fail *differen-*

tially on small structural UI elements (icons) versus large textual elements (text). This appendix decomposes ScreenSpot-V2 grounding accuracy across the full $\beta \in \{1, 3, 5, 10, 15, 20, 40, 80\}\%$ budget sweep, separately by element type.

Main finding. Figure 14 shows the two element-type curves side-by-side, and Table 36 reports the icon-target numbers explicitly. On **text**, the four methods converge for $\beta \geq 10\%$ and differ only at $\beta \leq 5\%$. On **icons**, the spread is much wider and persists to higher budgets:

β	**ST-Lite**	SnapKV	Pyr.KV	VL-Cache
3	**57.7**	57.6	43.4	32.0
5	**74.0**	71.9	52.4	53.1
10	**79.9**	79.1	69.9	65.6
15	80.4	**80.9**	77.2	77.8
20	**83.3**	82.8	78.2	79.7
40	**83.4**	83.0	82.2	82.8
80	84.4	84.3	84.3	**84.5**

Table 36: Icon-grounding accuracy (%) on ScreenSpot-V2 across the budget sweep. Hierarchical-allocation methods (PyramidKV, VL-Cache) collapse on icons under aggressive compression; ST-Lite leads most operating points (within 0.5% of the best at β=15% and β=80%).

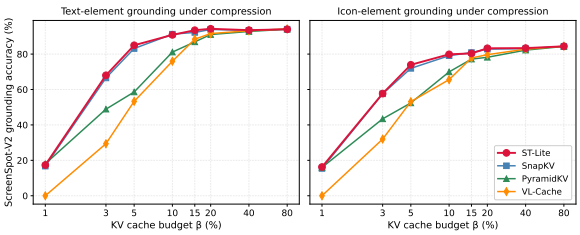


Figure 14: Element-type decomposition of ScreenSpot-V2 grounding accuracy under KV cache compression. **Left:** text-target accuracy is similar across methods once $\beta \geq 10\%$. **Right:** icon-target accuracy diverges sharply at low budgets — hierarchical methods drop $\geq 20\%$ behind ST-Lite at β=5% on icons, while the text gap is far smaller. Empirical signature of structure misalignment.

Interpretation. Three observations support the structure-misalignment account. (i) The text-vs-icon gap for hierarchical methods is largest exactly where their per-layer budget reweighting is most aggressive, i.e., at low β. (ii) The gap closes broadly as β grows and the budget allocation becomes less aggressive across layers. (iii) ST-Lite — which uses a uniform per-layer budget combined with the spatial-saliency mechanism — preserves icon accuracy throughout the aggressive-compression regime, supporting uniform allocation as an important condition for low-budget icon retention when paired with spatial saliency.

Cross-platform robustness. The structural-preservation advantage replicates across all three ScreenSpot-V2 platforms (mobile, desktop, web). Table 37 reports combined grounding accuracy at the most aggressive budget, β=5%, and Figure 15 visualises the pattern across three low budgets. ST-Lite leads on every platform at every budget shown, with the largest gap on desktop, where the higher screen density makes the spatial information needed for small-control localisation particularly vulnerable to hierarchical eviction. VL-Cache regresses to 26.8% on desktop at β=5% (a −43.4% gap behind ST-Lite); PyramidKV sits uniformly 26–34% behind ST-Lite across the three platforms.

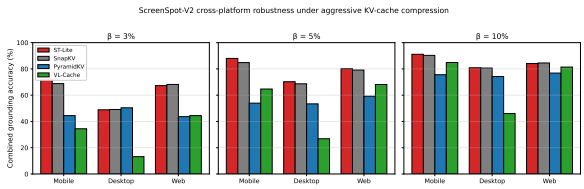


Figure 15: Cross-platform robustness on ScreenSpot-V2 under aggressive KV-cache compression. Each panel reports combined grounding accuracy by platform at the labelled budget. ST-Lite dominates every platform-budget cell.

Method	**Mobile**	**Desktop**	**Web**
ST-Lite	**88.0**	**70.3**	**80.1**
SnapKV	84.8	68.7	79.2
PyramidKV	53.9	53.4	59.2
VL-Cache	64.7	26.8	68.1
Δ over PyramidKV	+34.1	+16.9	+20.9
Δ over VL-Cache	+23.3	+43.4	+12.0

Table 37: ScreenSpot-V2 grounding accuracy (%) at β=5%, by platform. ST-Lite leads all three platforms; VL-Cache regresses on desktop.

Dataset	Model	Method	Budget								
			1%	**3%**	**5%**	**10%**	**15%**	**20%**	**40%**	**80%**	**100%**
ScreenSpot Pro	UI-TARS-1.5-7B	SnapKV	6.9	**23.6**	29.3	36.8	38.6	39.7	40.1	42.1	**42.3**
		PyramidKV	4.8	14.1	20.9	26.5	29.9	34.6	38.9	41.3	**42.3**
		VL-Cache	1.1	15.8	23.9	32.2	34.6	36.7	40.3	41.4	**42.3**
		GUI-KV	7.0	23.5	30.5	36.8	38.7	40.2	41.2	42.4	**42.3**
		ST-Lite (CSS only)	7.3	23.4	31.0	36.9	39.1	42.2	42.7	43.4	**42.3**
		ST-Lite	**7.3**	23.4	**31.0**	**36.9**	**39.1**	**42.2**	**42.7**	**43.4**	**42.3**
	OpenCUA-7B	SnapKV	10.0	16.1	18.5	17.7	15.2	17.2	14.7	26.9	**43.3**
		PyramidKV	**14.9**	14.5	14.3	11.9	14.3	12.9	13.9	14.4	**43.3**
		VL-Cache	1.1	14.6	15.2	15.6	13.8	16.7	15.1	14.5	**43.3**
		GUI-KV	9.9	16.7	19.4	17.7	16.5	19.0	18.3	28.0	**43.3**
		ST-Lite (CSS only)	9.5	18.9	20.1	17.9	16.8	19.4	19.3	29.3	**43.3**
		ST-Lite	9.5	**18.9**	**20.1**	**17.9**	**16.8**	**19.4**	**19.3**	**29.3**	**43.3**
AITW	UI-TARS-1.5-7B	SnapKV	8.1	11.5	13.2	14.2	14.5	15.3	15.6	16.6	**18.2**
		PyramidKV	**8.6**	7.5	10.8	14.2	15.1	14.4	15.1	**17.2**	**18.2**
		VL-Cache	0.2	1.7	2.1	9.8	10.5	11.1	10.1	9.1	**18.2**
		GUI-KV	8.2	12.0	14.0	15.5	16.1	16.7	16.6	16.8	**18.2**
		ST-Lite (CSS only)	8.3	12.4	14.1	15.7	16.7	17.4	16.9	16.7	**18.2**
		ST-Lite	**8.6**	**13.2**	**16.0**	**18.4**	**20.0**	**20.1**	**19.0**	17.1	**18.2**
	OpenCUA-7B	SnapKV	1.6	**8.2**	**11.7**	12.5	14.3	14.2	13.1	10.7	**17.3**
		PyramidKV	1.3	4.2	8.9	11.7	13.3	12.8	10.9	9.8	**17.3**
		VL-Cache	0.0	0.0	0.1	0.6	1.1	2.1	6.7	7.7	**17.3**
		GUI-KV	1.5	7.2	10.9	12.7	14.8	14.9	13.5	11.2	**17.3**
		ST-Lite (CSS only)	1.9	7.7	11.4	12.2	14.7	14.2	12.9	11.2	**17.3**
		ST-Lite	1.5	6.7	10.6	**13.3**	**15.8**	**16.4**	**14.3**	**12.3**	**17.3**
AgentNetBench	UI-TARS-1.5-7B	SnapKV	0.4	0.8	1.5	4.1	8.4	9.3	17.8	18.9	**17.5**
		PyramidKV	**1.4**	1.8	0.8	1.3	2.7	4.1	11.0	19.0	**17.5**
		VL-Cache	0.0	0.7	1.9	**9.8**	**15.9**	16.1	16.9	18.7	**17.5**
		GUI-KV	0.2	1.0	1.8	6.1	11.9	15.5	18.5	19.5	**17.5**
		ST-Lite (CSS only)	0.9	2.1	2.7	7.7	9.9	14.1	18.1	18.7	**17.5**
		ST-Lite	0.1	1.7	**3.0**	**9.8**	13.2	**17.0**	**20.7**	**20.5**	**17.5**
	OpenCUA-7B	SnapKV	0.8	3.2	3.7	5.6	6.6	8.0	17.1	28.3	**66.4**
		PyramidKV	0.6	0.6	0.8	1.6	2.6	4.7	11.1	26.9	**66.4**
		VL-Cache	0.5	2.8	1.9	0.9	4.7	8.1	0.9	19.9	**66.4**
		GUI-KV	0.7	5.9	7.7	12.0	13.8	16.7	19.4	30.2	**66.4**
		ST-Lite (CSS only)	1.6	1.8	2.3	3.5	6.7	8.4	19.1	28.1	**66.4**
		ST-Lite	0.2	**6.6**	**8.7**	**13.6**	**15.6**	**20.8**	**28.6**	**37.6**	**66.4**

Table 38: Detailed performance comparison on key benchmarks (ScreenSpot Pro, AITW, and AgentNetBench) across different budget ratios. Default configuration: SnapKV-style window base (δ=8), α=1, cosine similarity. On single-frame ScreenSpot Pro, TSG is a no-op so "ST-Lite (CSS only)" and "ST-Lite" rows are identical.

Dataset	Model	Method	Budget								
			1%	3%	5%	10%	15%	20%	40%	80%	100%
ScreenSpotV2	UI-TARS-1.5-7B	SnapKV	16.1	**62.9**	77.5	85.2	86.5	88.3	88.2	88.4	**88.9**
		PyramidKV	17.0	47.1	55.5	75.5	82.9	84.6	87.5	88.6	**88.9**
		VL-Cache	0.0	30.7	53.2	70.8	83.2	85.6	88.0	88.4	**88.9**
		GUI-KV	16.7	62.9	79.1	85.4	86.6	88.4	88.3	89.1	**88.9**
		ST-Lite (CSS only)	17.1	62.9	79.6	86.0	87.0	88.8	88.4	89.3	**88.9**
		ST-Lite	**17.1**	**62.9**	**79.6**	**86.0**	**87.0**	**88.8**	**88.4**	**89.3**	**88.9**
	OpenCUA-7B	SnapKV	**42.1**	71.5	**73.8**	70.8	64.0	66.3	60.2	74.3	**90.7**
		PyramidKV	16.9	34.9	41.7	41.0	41.4	63.3	57.9	74.4	**90.7**
		VL-Cache	0.9	52.3	59.0	67.5	57.8	61.1	55.8	74.0	**90.7**
		GUI-KV	42.0	71.6	72.8	71.0	65.2	67.9	65.8	75.9	**90.7**
		ST-Lite (CSS only)	41.9	71.7	69.9	71.1	65.5	70.4	67.2	77.0	**90.7**
		ST-Lite	41.9	**71.7**	69.9	**71.1**	**65.5**	**70.4**	**67.2**	**77.0**	**90.7**
AndroidControl	UI-TARS-1.5-7B	SnapKV	**18.9**	**26.1**	28.9	39.1	41.7	44.4	47.7	46.6	**49.6**
		PyramidKV	11.6	20.8	24.9	28.9	34.4	37.9	44.6	46.3	**49.6**
		VL-Cache	1.9	6.6	10.6	34.9	39.9	42.9	46.3	45.8	**49.6**
		GUI-KV	17.9	26.0	30.1	41.2	44.7	46.0	48.2	46.9	**49.6**
		ST-Lite (CSS only)	15.9	26.0	30.1	39.7	42.0	43.6	48.7	47.1	**49.6**
		ST-Lite	13.9	25.9	**34.8**	**44.2**	**46.9**	**48.9**	**50.3**	**48.1**	**49.6**
	OpenCUA-7B	SnapKV	**3.6**	15.3	18.6	21.6	27.9	25.9	30.9	26.1	**40.7**
		PyramidKV	1.9	**15.9**	15.9	22.9	27.3	27.9	23.1	26.9	**40.7**
		VL-Cache	0.0	2.9	2.5	11.9	15.7	19.5	27.9	6.5	**40.7**
		GUI-KV	3.0	15.8	19.6	23.8	30.2	27.3	32.3	31.7	**40.7**
		ST-Lite (CSS only)	2.3	14.2	18.8	23.3	29.8	28.7	34.4	29.1	**40.7**
		ST-Lite	0.7	**15.9**	**19.9**	**26.8**	**36.9**	**32.9**	**37.9**	**34.1**	**40.7**
Multimodal-Mind2Web	UI-TARS-1.5-7B	SnapKV	2.1	4.7	7.1	11.6	16.9	18.6	22.7	24.2	**25.9**
		PyramidKV	1.4	3.9	4.7	5.9	9.1	11.9	21.9	24.2	**25.9**
		VL-Cache	0.0	0.1	0.0	0.6	3.9	7.4	19.9	23.9	**25.9**
		GUI-KV	2.0	5.2	8.3	13.6	18.7	20.3	24.2	25.6	**25.9**
		ST-Lite (CSS only)	2.4	7.3	10.9	16.4	19.4	20.7	21.9	25.9	**25.9**
		ST-Lite	1.9	7.1	**12.9**	**21.4**	**25.7**	**27.2**	**30.2**	**31.1**	**25.9**
	OpenCUA-7B	SnapKV	**0.9**	**2.4**	5.6	9.4	12.7	14.3	16.7	17.3	**33.6**
		PyramidKV	0.0	0.9	1.8	5.6	6.7	10.9	12.7	14.1	**33.6**
		VL-Cache	0.0	0.0	0.0	0.1	0.1	0.0	13.4	14.4	**33.6**
		GUI-KV	0.4	2.3	6.0	10.2	13.7	14.8	17.0	17.7	**33.6**
		ST-Lite (CSS only)	0.8	0.0	5.7	10.1	11.7	14.4	17.1	18.7	**33.6**
		ST-Lite	0.2	1.8	**7.8**	**13.4**	**15.2**	**16.9**	**18.3**	**19.4**	**33.6**
OSWorld-Verified	UI-TARS-1.5-7B	SnapKV	**2.2**	**3.2**	6.9	16.6	16.7	18.6	19.2	23.2	**26.3**
		PyramidKV	0.2	1.9	1.1	2.9	9.9	15.6	16.9	20.3	**26.3**
		VL-Cache	0.0	1.9	**10.6**	7.6	8.9	9.7	10.1	19.9	**26.3**
		GUI-KV	1.9	2.8	8.3	17.1	18.0	20.7	25.1	24.2	**26.3**
		ST-Lite (CSS only)	1.9	3.0	7.1	17.3	18.2	19.6	23.1	25.3	**26.3**
		ST-Lite	0.6	1.1	8.6	**18.9**	**19.7**	**25.9**	**27.0**	**28.1**	**26.3**
	OpenCUA-7B	SnapKV	0.7	0.9	0.5	6.9	10.1	14.3	16.2	16.8	**23.8**
		PyramidKV	0.3	0.9	1.9	6.7	8.6	11.2	13.1	14.9	**23.8**
		VL-Cache	0.0	0.1	0.6	3.1	6.9	9.9	9.7	9.1	**23.8**
		GUI-KV	0.3	0.9	1.6	7.5	11.2	15.1	17.2	17.0	**23.8**
		ST-Lite (CSS only)	1.1	1.4	3.3	6.9	10.9	14.7	17.4	17.9	**23.8**
		ST-Lite	0.1	0.9	**4.1**	**8.9**	**13.9**	**16.9**	**19.7**	17.5	**23.8**

Table 39: Performance evaluation on additional benchmarks (ScreenSpotV2, AndroidControl, Multimodal-Mind2Web, and OSWorld-Verified).