# VERSIONSHIFT: EVALUATING AND ADAPTING PRIOR EXPERIENCE ACROSS SOFTWARE VERSIONS FOR GUI AGENTS

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

GUI agents interact with graphical user interfaces (GUIs) to perform tasks in real-world software environments, yet these interfaces continuously evolve through software updates, changing layouts, element locations, and task workflows. Such evolution makes **cross-version experience adaptation** essential: prior experience may remain useful after an update, but can also become outdated or misleading. However, selective adaptation of prior experience across software versions remains underexplored, and existing benchmarks provide limited support for controlled evaluation of such experience transfer. To systematically study this problem, we introduce **VERSIONSHIFT**, a benchmark that holds task intent fixed while pairing executions across chronological software versions. Through semantic alignment of paired trajectories, VERSIONSHIFT characterizes source-experience validity at each target step using **REUSE, REGROUND, and REPLAN** annotations, distinguishing directly executable source actions, applicable semantics that require new grounding, and target steps with no applicable source action. VERSIONSHIFT contains 312 tasks and 1,248 trajectories across 53 applications and websites spanning Mobile, Desktop, and Web. Evaluation on VERSIONSHIFT shows that historical trajectories can improve action prediction while also introducing substantial negative transfer, with beneficial and harmful effects coexisting within the same model. To retain useful experience while limiting negative transfer, we propose **VAEA**, a validity-aware experience adaptation approach that learns to select applicable source steps and estimate direct-reuse validity. VAEA uses the predicted applicable source steps and reuse-validity score to conditionally reuse source actions, reground selected semantics, or replan when no source step is applicable, without requiring cross-version annotations at inference. Experiments show that VAEA improves action prediction over controlled experience-use baselines and consistently reduces history-induced negative transfer, while remaining competitive with other memory methods. We will release the benchmark and code to facilitate future research.

## 1 INTRODUCTION

Graphical user interfaces (GUIs) provide a general interaction layer through which agents can operate diverse applications without relying on application-specific APIs (Zhang et al., 2025). Recent advances in multimodal large language models (MLLMs) have further improved visual understanding, grounding, and action generation, enabling increasingly capable GUI agents (Nguyen et al., 2025).

However, real-world GUIs continuously evolve through software updates, changing layouts, element locations, and task workflows. Such evolution creates a key challenge: prior experience may remain valid after an update, but can also become outdated or misleading (Moran et al., 2018). Figure 1 illustrates this challenge with VLC: a theme-setting task grows from three steps in VLC v1.7 to five steps in VLC v2.5 after the relevant option is moved to a newly introduced Interface page. The earlier trajectory is therefore only partially reusable: opening the menu remains directly executable (S1 $\rightarrow$ T1), opening the setting preserves its semantics but requires regrounding (S2 $\rightarrow$ T2), while the newly introduced intermediate steps have no corresponding source actions and require replanning (T3 and T4). Effective cross-version adaptation thus requires agents to determine which parts of prior experience remain reliable.



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

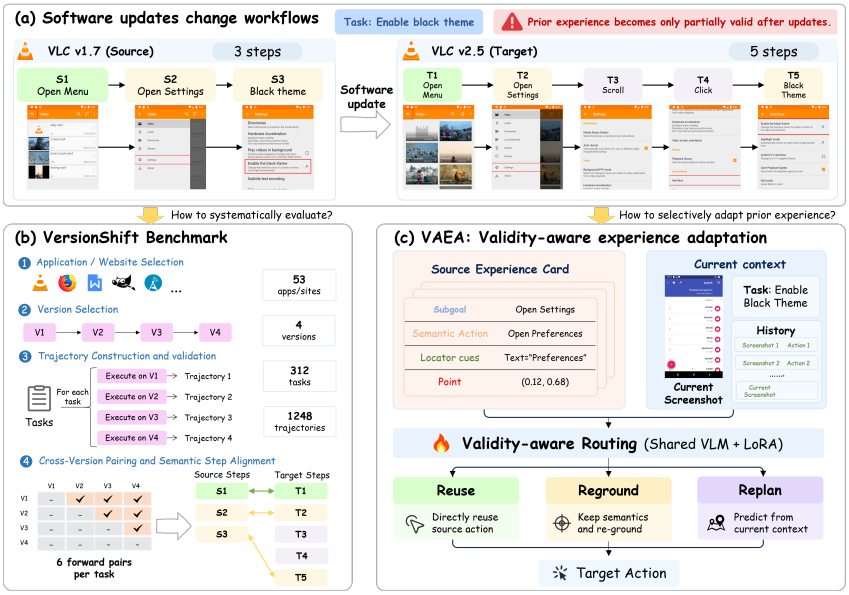


Figure 1: **Cross-version experience adaptation with VERSIONSHIFT and VAEA. (a)** Software updates change task workflows, leaving prior experience only partially valid. **(b)** VERSIONSHIFT evaluates cross-version experience validity using same-task trajectories, forward version pairs, and semantic step alignment. **(c)** VAEA combines source experience with the current context and adaptively determines whether to REUSE, REGROUND, or REPLAN for target action prediction.

Existing work has begun to address interface evolution from several perspectives. TransBench studies cross-version GUI grounding (Lu et al., 2025), while TimeWarp evaluates Web agents under temporal interface variation and explores cross-version training (Ishmam & Marino, 2026). MAGNET uses memory to preserve functional and procedural knowledge across changes in mobile interfaces (Sun et al., 2026). However, existing settings do not explicitly characterize which parts of a specific prior execution remain valid under a particular version transition. Selective adaptation of such experience remains underexplored, motivating a more specific question: *how should a GUI agent selectively adapt and reuse prior experience as software evolves?*

To study this question, we introduce **VERSIONSHIFT**, a benchmark that pairs same-task executions across chronological software versions. We semantically align paired trajectories and annotate source-experience validity at each target step as **REUSE, REGROUND, or REPLAN**, capturing the different levels at which prior experience remains usable. The benchmark contains 312 tasks and 1,248 trajectories across 53 applications and websites on Mobile, Desktop, and Web. Our evaluation reveals heterogeneous cross-version transfer: historical trajectories improve action prediction for some agents but cause net degradation for others. Beneficial and harmful effects can also coexist within the same model, showing that aggregate gains can conceal history-induced errors.

To retain useful experience while limiting negative transfer, we propose **VAEA**, a validity-aware experience adaptation approach that predicts *which source steps remain applicable* and *whether a selected source action can be directly reused*. These two predictions jointly determine how source experience is used: VAEA conditionally reuses compatible source actions, regrounds selected semantics without stale localization cues, or replans from the current target context when no source step is applicable. Experiments show that VAEA improves action prediction over experience-use baselines while reducing history-induced negative transfer. Our contributions are listed as follows:

1. We introduce **VERSIONSHIFT**, a cross-version GUI benchmark that pairs same-task executions across chronological versions on Mobile, Desktop, and Web. Semantic step alignment and validity annotations enable evaluation of what remains reliable from prior experience.
2. We characterize cross-version experience transfer across diverse GUI agents, showing that historical experience can have both beneficial and harmful effects within the same model. This analysis exposes harmful transfer that aggregate performance gains can conceal.

Table 1: **Comparison with representative GUI-agent benchmarks.** M, D, and W denote Mobile, Desktop, and Web, respectively. Cross-version indicates explicit evaluation across interface or software versions. Traj. Pairs indicates whether executions of the same task are paired across versions for source–target experience transfer.

| Benchmark | Apps / Sites | Tasks / Queries | Trajectories | Eval. Type | Platforms | Cross-version | Traj. Pairs |
|---|---|---|---|---|---|---|---|
| Mind2Web (Deng et al., 2023) | 137 | 2,350 | 2,350 | Task | W | ✗ | ✗ |
| AITW (Rawles et al., 2023) | 357+ | 30,378 | 715,142 | Task | M | ✗ | ✗ |
| AndroidWorld (Rawles et al., 2025) | 20 | 116 | – | Task | M | ✗ | ✗ |
| OSWorld (Xie et al., 2024) | 9 | 369 | – | Task | D | ✗ | ✗ |
| TransBench (Lu et al., 2025) | 81 | 22,000+ | – | Grounding | M/W | ✓ | ✗ |
| TimeWarp (Ishmam & Marino, 2026) | 3 | 231 | 757 | Task | W | ✓ | ✗ |
| **VERSIONSHIFT** | **53** | **312** | **1,248** | **Task** | **M/D/W** | ✓ | ✓ |

3. We propose **VAEA**, which learns to select applicable source steps and estimate reuse validity to control information retained from historical experience. Experiments and diagnostic analyses show improved action prediction and reduced history-induced negative transfer.

## 2 RELATED WORK

**Experience Adaptation for GUI Agents.** Recent work has explored how GUI agents can store, retrieve, and reuse interaction experience. AWM and HMT organize past executions into reusable workflows or hierarchical memories to improve task generalization (Wang et al., 2024; Tan et al., 2026). UI-Mem studies hierarchical experience transfer across tasks and applications, while AndroT-Mem focuses on anchored state memory for long-horizon GUI interaction (Xiao et al., 2026; Shi et al., 2026). CoMEM explores scalable trajectory compression through continuous memory representations (Wu et al., 2025). MAGNET further considers evolving mobile interfaces and preserves stable semantic and procedural knowledge across interface changes (Sun et al., 2026).

However, these methods do not explicitly characterize which parts of a specific source execution remain valid under a particular version transition. In contrast, we first study this source-experience validity through controlled cross-version trajectory pairs, enabling step-level analysis of what remains transferable as software evolves.

**GUI Benchmarks.** Existing GUI benchmarks span Web, Mobile, and Desktop environments, including Mind2Web, WebArena, AndroidWorld, and OSWorld (Deng et al., 2023; Zhou et al., 2024; Rawles et al., 2025; Xie et al., 2024), while OpenCUA and TongUI extend evaluation across multiple platform families (Wang et al., 2026; Zhang et al., 2026). Recent work has further begun to study interface evolution: TransBench evaluates cross-version transfer for GUI grounding (Lu et al., 2025), and TimeWarp studies Web-agent robustness under temporally varying interfaces (Ishmam & Marino, 2026). Different from these settings, VERSIONSHIFT focuses on transfer from a specific prior execution by pairing multi-step trajectories of the same task across chronological versions on Mobile, Desktop, and Web, and characterizing source-experience validity at each target step.

## 3 VERSIONSHIFT: CROSS-VERSION EXPERIENCE ADAPTATION BENCHMARK

Cross-version experience adaptation asks which parts of prior experience remain valid as software evolves. Studying this problem requires separating changes in source-experience validity from changes in task intent. VERSIONSHIFT therefore pairs executions of the same task across chronological software versions while keeping the task specification fixed, enabling controlled evaluation of cross-version experience transfer and step-level validity.

### 3.1 BENCHMARK OVERVIEW

VERSIONSHIFT isolates software evolution by keeping the application or website, task intent, initial-state specification, and success criterion fixed while varying the software version. It covers 53 applications and websites across Mobile, Desktop, and Web, with 312 tasks. Each task is executed on four chronologically ordered versions, yielding 1,248 validated trajectories and 4,191 executable

actions. The four trajectories induce six older-to-newer source–target pairs per task, resulting in 1,872 forward trajectory pairs. We focus on forward transfer, reflecting the practical setting in which experience from an earlier version is reused after a software update. Table 1 compares VERSIONSHIFT with representative benchmarks. To construct these pairs and their validity annotations, we first collect and validate one successful trajectory for each task–version instance, then pair trajectories across versions, align their steps by semantic subgoal, and annotate source-experience validity at each target step. The following subsections describe these two stages.

### 3.2 TRAJECTORY CONSTRUCTION

We first construct one validated trajectory for each task–version instance through application and task selection, trajectory collection, and quality control.

**Application, version, and task selection.** We select applications with four reproducible releases or dated snapshots that can be restored to stable initial states. For each application, we define tasks with consistent intent and observable success criteria across all four versions. Each task uses the same natural-language instruction, initial-state specification, and success criterion across versions. Tasks whose intended goal cannot be consistently reproduced across all four versions due to transient content, unavailable external services, or version-specific functionality are excluded.

**Trajectory collection.** For each task–version pair, we reset the environment to the prescribed initial state and manually record one successful execution. We preserve semantically equivalent execution paths when possible to reduce arbitrary demonstrator variation, but do not force identical action sequences across versions when an update naturally changes the required workflow. Each step records the observation, action, and target-element metadata, while platform-specific events are normalized into a shared action space. Accordingly, VERSIONSHIFT evaluates transferability along controlled execution paths rather than exhaustively covering all valid solutions.

**Quality control.** Each trajectory undergoes automated checks for schema validity, coordinate ranges, completeness, and associated artifacts, followed by manual verification of the initial state, action sequence, target elements, and terminal success. Invalid or ambiguous executions are corrected or recollected, and only tasks with four complete validated trajectories are retained.

### 3.3 CROSS-VERSION PAIRING AND VALIDITY ANNOTATION

The validated trajectories are then converted into directional cross-version pairs and step-level validity annotations. Software updates can insert, remove, split, or merge interaction steps, making source and target actions unreliable to align by trajectory position. We therefore align steps according to their semantic subgoals and roles in completing the task. The alignment supports one-to-one, one-to-many, many-to-one, and unmatched correspondences. For each source–target pair, we annotate each target step according to whether and how aligned source experience remains applicable:

- **REUSE**: A source action remains semantically appropriate and directly executable on the target interface without modifying its action parameters. For example, a menu item remains at the same location and can be clicked directly.
- **REGROUND**: Source action semantics remain applicable, but localization or spatial parameters must be updated on the target interface. For example, the same control is preserved but moved to a different location.
- **REPLAN**: No source action is applicable to the current target step, including cases where workflow changes introduce new actions or invalidate a previously relevant action. For example, an update introduces a new navigation step that has no source counterpart.

These labels characterize source-experience validity rather than the magnitude of visual change alone. An interface can change while preserving the executability of a particular action; conversely, a visually similar interface can require a different action or workflow. Across all forward pairs, REUSE, REGROUND, and REPLAN account for 51.2%, 24.1%, and 24.7% of source-conditioned target-step instances, respectively. Additional details on benchmark construction, annotation, and statistics are provided in Appendix A.

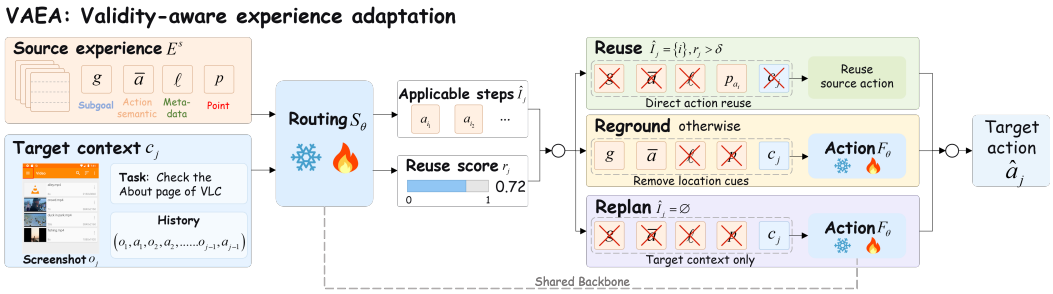


Figure 2: **VAEA: validity-aware experience adaptation.** At each target step, the routing mode $S_\theta$ selects applicable source steps $\hat{\mathcal{I}}_j$ and estimates a reuse score $r_j$ from the source experience and target context. These outputs determine whether to directly REUSE a source action, or invoke the action-generation mode $F_\theta$ to REGROUND selected source semantics or REPLAN from the target context alone. $S_\theta$ and $F_\theta$ are two modes of the same VLM with a shared LoRA adapter.

## 4 VAEA: VALIDITY-AWARE EXPERIENCE ADAPTATION

VERSIONSHIFT shows that source experience can remain valid at different levels after software evolution: a source action may remain directly executable, retain useful semantics but require new grounding, or become inapplicable. VAEA handles these cases through two validity decisions: *which source steps remain semantically applicable* and *whether an applicable source action can still be directly executed*. Accordingly, VAEA REUSEs a valid source action, REGROUNDs applicable semantics on the current interface, or REPLANs when no source step is applicable.

### 4.1 SOURCE EXPERIENCE REPRESENTATION

To support these decisions without retaining potentially stale source observations, VAEA represents each source step with a compact experience card containing transferable semantics and information for direct reuse. Given a successful source trajectory $\tau^s = \{(o_i^s, a_i^s)\}_{i=1}^n$, each step is represented as:

$$e_i^s = (g_i^s, \bar{a}_i^s, \ell_i^s, p_i^s), \tag{1}$$

where $g_i^s$ describes the step-level subgoal and $\bar{a}_i^s$ abstracts the executed action into transferable semantics, capturing what should be accomplished independent of source-side location. $\ell_i^s$ describes the source action's target control using metadata such as its text, name, and role, helping determine whether a corresponding control still exists on the target interface. For pointer actions, $p_i^s \in [0,1]^2$ stores the normalized source location used for direct reuse, with $p_i^s = \varnothing$ otherwise. The resulting source experience bank is $E^s = \{e_i^s\}_{i=1}^n$.

### 4.2 VALIDITY-AWARE ADAPTATION

At target step $j$, the current target context is $c_j = (u, o_j^t, a_{<j}^t)$, where $u$ is the task instruction, $o_j^t$ is the current observation, and $a_{<j}^t$ is the target-side action history. Given $c_j$ and the source card bank $E^s$, VAEA first determines what source experience remains applicable before deciding how it should affect the target action.

**Source-experience routing.** The routing mode $S_\theta$ determines whether and how source experience should be used. It predicts

$$(\hat{\mathcal{I}}_j, r_j) = S_\theta(c_j, E^s), \tag{2}$$

where $\hat{\mathcal{I}}_j$ contains source steps whose semantics remain applicable to the current target state, and $r_j$ is a confidence score indicating whether a selected source action remains directly executable without modification. These outputs separate semantic applicability from direct executability. If $\hat{\mathcal{I}}_j = \varnothing$, no source step is applicable and VAEA REPLANs. Otherwise, the selected source experience provides applicable semantics; $r_j$ determines whether the source action is directly REUSEd or its semantics must be REGROUNDed on the target interface.

**Conditional experience adaptation.** The target action is determined by

$$\hat{a}_j = \begin{cases} F_\theta(c_j), & \hat{\mathcal{I}}_j = \varnothing, \\ \mathrm{Reuse}(e_i^s), & \hat{\mathcal{I}}_j = \{i\},\ r_j \geq \delta, \\ F_\theta\Big(c_j, \{(g_i^s, \bar{a}_i^s)\}_{i \in \hat{\mathcal{I}}_j}\Big), & \text{otherwise}, \end{cases} \tag{3}$$

corresponding to REPLAN, REUSE, and REGROUND, respectively. For REPLAN, the action-generation mode $F_\theta$ predicts the target action without source experience. For REUSE, a unique compatible pointer action with sufficiently high reuse validity directly retains its source location $p_i^s$. For REGROUND, $F_\theta$ receives only the selected source subgoals and action semantics, preserving semantic guidance while discarding source-side control metadata and localization information that may have become stale.

Thus, $S_\theta$ decides *whether and how* prior experience should be used, whereas $F_\theta$ generates a target-side action whenever direct reuse is not selected. This design directly mirrors the three validity cases characterized by VERSIONSHIFT while allowing the appropriate mode to be inferred from the current target state.

### 4.3 MULTITASK TRAINING AND INFERENCE

$S_\theta$ and $F_\theta$ are two instruction-conditioned modes of the same VLM backbone with a shared LoRA adapter (Hu et al., 2021). Routing examples supervise applicable source-card selection and reuse prediction, while action examples supervise target-side action generation. Both modes are optimized with standard causal language-modeling losses:

$$\mathcal{L}_{\mathrm{VAEA}} = \mathcal{L}_{\mathrm{route}}^{\mathrm{LM}} + \mathcal{L}_{\mathrm{act}}^{\mathrm{LM}}. \tag{4}$$

The backbone is frozen and only the shared LoRA adapter is optimized. Cross-version correspondences supervise source-step applicability, reuse-validity labels supervise direct-reuse prediction, and recorded target actions supervise action generation.

At inference, VAEA requires only the current target context $c_j$ and source card bank $E^s$. Each target step first invokes $S_\theta$ to predict $\hat{\mathcal{I}}_j$ and $r_j$. Direct REUSE requires no additional model call, whereas REGROUND and REPLAN invoke $F_\theta$ to generate the target action. Cross-version alignments, validity annotations, target actions, and target-element annotations are used only for supervision and are unavailable at inference. Implementation details are provided in Appendix B.

## 5 EXPERIMENTS

### 5.1 EXPERIMENTAL SETUP

**Evaluation protocol.** We first evaluate existing agents without benchmark-specific training on the full VERSIONSHIFT benchmark, covering all 53 applications and websites and 312 tasks. For trainable methods, we instead use application-disjoint train, development, and test splits containing 207, 50, and 55 tasks from 35, 9, and 9 applications, respectively. The development split is used only for model selection and reuse-threshold calibration, and all final results are reported on the held-out test split. No test trajectories or annotations are used for training or model selection.

**Baselines and implementation.** We first evaluate representative closed-source models, open-source VLMs (Bai et al., 2025b;a), and specialized GUI agents (Wu et al., 2026; Lin et al., 2025; Xu et al., 2024; Xie et al., 2026; Zhou et al., 2026; Qin et al., 2025) without benchmark-specific fine-tuning. **Target-Only** uses only the current target context, while **+Raw History** additionally receives an unadapted trajectory from an earlier version, isolating the effect of directly transferring prior experience.

For controlled adaptation experiments, we use Qwen3-VL-4B and Qwen3-VL-8B as backbones (Bai et al., 2025a) and compare baselines that progressively control how source experience is introduced, represented, and selected. **Target-only** is trained only on target-version examples, while **Experience Replay** additionally trains on source-version examples but uses no source experience at inference. **Raw History** conditions on the complete source trajectory, whereas **Compact History** conditions

on its compact experience cards. Their **+Two-Pass** variants add a second model pass to control for the inference computation of VAEA. **Supervised Selection + Actor** learns source-step selection from correspondence supervision and generates actions from the selected coordinate-free semantics, isolating source-step selection without validity-aware adaptation. Finally, **VAEA** selectively REUSEs, REGROUNDs, or REPLANs source experience according to its predicted validity.

For comparison with existing memory-based methods, we additionally evaluate AWM (Wang et al., 2024) on Web and MAGNET (Sun et al., 2026) on Mobile, following their original platform scopes and official implementations under the VERSIONSHIFT evaluation protocol. Within each backbone, all controlled methods use the same data split, target-side supervision, action scorer, and optimization protocol. Further implementation details are provided in Appendix C.

**Metrics.** We report **Step accuracy (Step)**, which measures next-action correctness at each executable target step, and **Trajectory accuracy (Traj.)**, which counts a target trajectory as correct only when all executable target actions are correctly predicted under the recorded observations and reference action histories. We further measure **Positive Transfer (PTR)** and **Negative Transfer (NTR)** relative to the corresponding Target-only reference $T$. For a method $M$,

$$\mathrm{PTR}(M,T)=\frac{\#\{k:T_k \text{ incorrect},\ M_k \text{ correct}\}}{N},\qquad \mathrm{NTR}(M,T)=\frac{\#\{k:T_k \text{ correct},\ M_k \text{ incorrect}\}}{N}. \tag{5}$$

where $k$ indexes paired target-step decisions and $N$ is their total number. Since PTR and NTR are computed over the same instances, their difference gives the net change in Step accuracy, $\Delta\mathrm{Step}=\mathrm{PTR}-\mathrm{NTR}$.

### 5.2 CROSS-VERSION EXPERIENCE TRANSFER IN EXISTING GUI AGENTS

We first examine whether existing GUI agents can benefit from a successful trajectory collected on an earlier software version. For each model, we compare **Target-Only** with **+Raw History** under identical target states and model parameters. Table 2 reveals highly heterogeneous cross-version transfer: Raw History improves Step accuracy for only 5 of the 13 evaluated models, with changes ranging from +21.48 to -28.18. Moreover, positive and negative transfer can coexist within the same model. For example, Qwen3-VL-8B achieves a PTR of 16.92% but an NTR of 10.56%, showing that its overall improvement masks a substantial number of newly introduced errors. These results indicate that historical trajectories contain useful procedural knowledge but are not uniformly reliable across software versions, motivating selective adaptation of prior experience.

### 5.3 CROSS-VERSION EXPERIENCE ADAPTATION WITH VAEA

We next evaluate whether VAEA improves cross-version experience adaptation beyond the controlled alternatives defined above. Table 3 compares all methods under the same training and evaluation protocol.

**Historical experience is useful but not uniformly reliable.** Raw History substantially improves over Target-only, confirming that prior trajectories contain transferable information, while its nontrivial NTR shows that direct conditioning also introduces new errors. Compact History remains weaker, indicating that compact experience representation alone does not resolve this reliability problem.

**VAEA provides gains beyond representation, selection, and extra inference.** VAEA achieves the highest Step and Traj. accuracy on both backbones while consistently reducing NTR relative to Raw History. The Two-Pass baselines remain below VAEA, indicating that the gains are not explained solely by additional inference computation. VAEA also outperforms Supervised Selection + Actor, showing that selecting applicable source steps alone is insufficient; distinguishing direct reuse from target-side regrounding provides additional benefit. Together, these results support validity-aware adaptation over indiscriminate use of cross-version experience.

### 5.4 COMPARISON WITH PLATFORM-SPECIFIC MEMORY BASELINES

We further compare VAEA with platform-specific memory baselines: AWM on Web and MAGNET on Mobile. Comparisons are reported separately within each platform, using Qwen3-VL-8B with the same evaluation states and source–target pairs.

Table 2: **Cross-version experience transfer in existing GUI agents.** Target-Only uses the current target context, while +Raw History additionally receives an unadapted trajectory from an earlier version. $\Delta$ Step, PTR, and NTR quantify net, positive, and negative transfer, respectively.

| Category | Model | Target-Only | | +Raw History | | Paired Transfer | | |
|---|---|---|---|---|---|---|---|---|
| | | Step ↑ | Traj. ↑ | Step ↑ | Traj. ↑ | $\Delta$ Step ↑ | PTR ↑ | NTR ↓ |
| **Closed-source** | Claude Sonnet 5 | 13.48 | 1.23 | 22.67 | 2.14 | +9.19 | 13.92 | 4.73 |
| | GPT-5.6-Terra | **49.83** | 16.35 | **71.30** | **42.41** | **+21.48** | **31.36** | 9.89 |
| **Foundation VLM** | Qwen2.5-VL-3B | 23.12 | 3.63 | 12.46 | 3.63 | -10.66 | 6.48 | 17.14 |
| | Qwen2.5-VL-7B | 30.24 | 5.45 | 28.00 | 7.85 | -2.24 | 12.11 | 14.35 |
| | Qwen3-VL-4B | 35.90 | 5.45 | 32.31 | 14.53 | -3.60 | 7.52 | 11.12 |
| | Qwen3-VL-8B | 31.27 | 12.71 | 37.62 | 15.76 | +6.35 | 16.92 | 10.56 |
| **GUI agent** | GUI-Actor-3B | 3.19 | 0.00 | 3.78 | 0.00 | +0.60 | 3.19 | 2.59 |
| | ShowUI-2B | 2.57 | 1.23 | 1.55 | 0.91 | -1.02 | 0.77 | **1.80** |
| | Aguvis-7B | 42.78 | 16.35 | 14.60 | 5.45 | -28.18 | 2.57 | 30.75 |
| | JEDI-3B | 37.89 | 11.81 | 23.29 | 9.08 | -14.60 | 7.55 | 22.15 |
| | MAI-UI-2B | 3.09 | 0.00 | 8.77 | 1.82 | +5.68 | 8.25 | 2.57 |
| | MAI-UI-8B | 43.47 | **18.16** | 32.48 | 8.49 | -10.99 | 11.86 | 22.85 |
| | UI-TARS-1.5-7B | 26.38 | 2.40 | 22.67 | 7.59 | -3.71 | 14.10 | 17.80 |

Table 3: **Selective cross-version experience adaptation.** We compare VAEA with controlled baselines that isolate source-side training, history conditioning, compact experience representation, additional inference, and supervised source-step selection.

| Backbone | Method | Performance | | Transfer | | |
|---|---|---|---|---|---|---|
| | | Step ↑ | Traj. ↑ | $\Delta$ Step ↑ | PTR ↑ | NTR ↓ |
| Qwen3-VL-4B | Target-only | 63.57 | 37.88 | – | – | – |
| | Experience Replay | 66.15 | 40.00 | +2.58 | 7.82 | 5.24 |
| | Raw History | 76.12 | 48.79 | +12.55 | 17.96 | 5.41 |
| | Raw History + Two-Pass | 77.58 | 52.12 | +14.00 | 18.73 | 4.73 |
| | Compact History | 70.45 | 40.61 | +6.87 | 12.89 | 6.01 |
| | Compact History + Two-Pass | 76.98 | 52.12 | +13.40 | 17.78 | 4.38 |
| | Supervised Selection + Actor | 78.09 | 53.94 | +14.52 | 18.30 | 3.78 |
| | **VAEA** | **79.21** | **54.55** | **+15.64** | **18.90** | **3.26** |
| Qwen3-VL-8B | Target-only | 68.99 | 43.94 | – | – | – |
| | Experience Replay | 70.10 | 45.45 | +1.12 | 6.19 | 5.07 |
| | Raw History | 77.32 | 52.12 | +8.33 | **15.98** | 7.65 |
| | Raw History + Two-Pass | 78.44 | 53.33 | +9.45 | 14.86 | 5.41 |
| | Compact History | 73.71 | 47.88 | +4.73 | 11.08 | 6.36 |
| | Compact History + Two-Pass | 78.09 | 53.33 | +9.11 | 13.62 | 4.51 |
| | Supervised Selection + Actor | 79.73 | 54.55 | +10.74 | 14.52 | 3.78 |
| | **VAEA** | **80.76** | **56.06** | **+11.77** | 15.03 | **3.26** |

On Web, AWM improves over Raw History, demonstrating the benefit of structured workflow memory. VAEA further improves Step accuracy while reducing NTR, indicating fewer history-induced errors. On Mobile, VAEA and MAGNET achieve comparable Step accuracy: MAGNET yields slightly higher PTR, whereas VAEA achieves lower NTR and higher Traj. accuracy. These results show that VAEA is competitive with specialized memory methods while achieving lower negative transfer on both evaluated platforms.

### 5.5 Ablation Study

We conduct fixed-checkpoint ablations on the same trained model, modifying either the routing policy or the source information exposed to the action branch. Table 5 shows that VAEA benefits from

Table 4: **Comparison with platform-specific memory baselines.** AWM is evaluated on Web and MAGNET on Mobile. All methods are evaluated under the same source–target protocol.

| Platform | Method | Performance | | Transfer | | |
|---|---|---|---|---|---|---|
| | | Step ↑ | Traj. ↑ | Δ Step ↑ | PTR ↑ | NTR ↓ |
| **Web** | Target-only | 53.91 | 35.42 | – | – | – |
| | Raw History | 61.74 | 48.96 | +7.83 | **16.09** | 8.26 |
| | Compact History | 61.74 | 46.88 | +7.83 | 10.00 | **2.17** |
| | AWM | 64.35 | 53.13 | +10.43 | 14.35 | 3.91 |
| | **VAEA** | **66.52** | **55.21** | **+12.61** | 15.22 | 2.61 |
| **Mobile** | Target-only | 79.49 | 48.00 | – | – | – |
| | Raw History | 82.54 | 55.33 | +3.05 | **11.53** | 8.47 |
| | Compact History | 77.97 | 43.33 | -1.53 | 5.93 | 7.46 |
| | MAGNET | **85.93** | 58.00 | **+6.44** | 10.85 | 4.41 |
| | **VAEA** | 84.92 | **59.33** | +5.42 | 7.97 | **2.54** |

Table 5: **Fixed-checkpoint ablations of VAEA.** Panel A modifies the routing policy while keeping source selection and action generation fixed, and Panel B modifies the source information exposed to the same trained model. Oracle variants use benchmark annotations only for diagnostic analysis.

| Variant | Qwen3-VL-4B | | | | Qwen3-VL-8B | | | |
|---|---|---|---|---|---|---|---|---|
| | Step ↑ | Traj. ↑ | PTR ↑ | NTR ↓ | Step ↑ | Traj. ↑ | PTR ↑ | NTR ↓ |
| *Panel A: Routing-policy interventions* | | | | | | | | |
| Always Reuse | 72.42 | 46.67 | 17.53 | 8.68 | 74.31 | 49.39 | 14.77 | 9.45 |
| Always Reground | 76.80 | 52.12 | 17.96 | 4.73 | 78.26 | 53.94 | 14.35 | 5.07 |
| Always Replan | 69.93 | 42.73 | 10.74 | 4.38 | 73.37 | 46.67 | 8.93 | 4.55 |
| **VAEA** | **79.21** | **54.55** | **18.90** | **3.26** | **80.76** | **56.06** | **15.03** | **3.26** |
| *Oracle Routing* | 80.84 | 57.88 | 20.19 | 2.92 | 82.13 | 59.39 | 15.89 | 2.75 |
| *Panel B: Source-information interventions* | | | | | | | | |
| Raw source representation | 74.83 | 48.79 | 17.62 | 6.36 | 76.98 | 51.21 | 15.03 | 7.04 |
| All semantic cards | 77.58 | 52.73 | 18.91 | 4.90 | 79.38 | 54.55 | 15.29 | 4.90 |
| Selected cards + old locator | 75.69 | 49.39 | 19.34 | 7.22 | 77.58 | 52.12 | 16.24 | 7.65 |
| **VAEA** | **79.21** | **54.55** | **18.90** | **3.26** | **80.76** | **56.06** | **15.03** | **3.26** |
| *Oracle Source Alignment* | 80.41 | 56.67 | 19.76 | 2.92 | 81.87 | 57.88 | 15.63 | 2.75 |

both adaptive routing and selective source information use. No fixed routing policy matches full VAEA: Always Reuse introduces the most negative transfer, Always Reground is the strongest fixed policy, and Always Replan discards useful source information. On the representation side, replacing semantic cards with raw source information or exposing all cards degrades performance, indicating that both semantic abstraction and source-step selection help reduce interference. Restoring old localization cues increases positive transfer but also introduces substantially more negative transfer, supporting the need to distinguish REUSE from REGROUND. Finally, the remaining gains from Oracle Routing and Oracle Source Alignment indicate additional headroom, with routing accounting for the larger gap in this diagnostic setting.

## 6 CONCLUSION

Software evolution changes not only GUI appearance and workflows, but also the validity of prior interaction experience. To study this problem, we introduce **VERSIONSHIFT**, a benchmark that pairs same-task executions across chronological software versions. We semantically align paired trajectories and annotate source-experience validity at each target step as REUSE, REGROUND, or REPLAN, capturing the different levels at which prior experience remains usable. Evaluation on VERSIONSHIFT reveals that historical experience can provide transferable knowledge while also introducing errors that aggregate performance gains may conceal. To retain useful experience while limiting negative transfer, we propose **VAEA**, which explicitly models source-experience validity to selectively reuse, reground, or discard prior experience. Evaluations show improved action prediction and reduced negative transfer over other baselines. These results highlight experience validity as an important consideration for GUI agents operating across evolving software.

## AI USE STATEMENT

In this work, we used generative AI tools for writing-related tasks, including language translation, improving manuscript clarity, and correcting vocabulary and grammatical issues. We reviewed all AI-assisted content to ensure that the final manuscript accurately reflects the authors' intended meaning. We take full responsibility for the final content of this work.

## REFERENCES

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

Shuai Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Sibo Song, Kai Dang, Peng Wang, Shijie Wang, Jun Tang, et al. Qwen2.5-vl technical report. *arXiv preprint arXiv:2502.13923*, 2025b.

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

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.

Md Farhan Ishmam and Kenneth Marino. Timewarp: Evaluating web agents by revisiting the past. *arXiv preprint arXiv:2603.04949*, 2026.

Kevin Qinghong Lin, Linjie Li, Difei Gao, Zhengyuan Yang, Shiwei Wu, Zechen Bai, Stan Weixian Lei, Lijuan Wang, and Mike Zheng Shou. Showui: One vision-language-action model for gui visual agent. In *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 19498–19508. IEEE, 2025.

Yuheng Lu, Qian Yu, Hongru Wang, Zeming Liu, Wei Su, Yanping Liu, Yuhang Guo, Maocheng Liang, Yunhong Wang, and Haifeng Wang. Transbench: Breaking barriers for transferable graphical user interface agents in dynamic digital environments. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 12464–12478, 2025.

Atif M Memon. Automatically repairing event sequence-based gui test suites for regression testing. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 18(2):1–36, 2008.

Atif M Memon and Mary Lou Soffa. Regression testing of guis. *ACM SIGSOFT software engineering notes*, 28(5):118–127, 2003.

Kevin Moran, Cody Watson, John Hoskins, George Purnell, and Denys Poshyvanyk. Detecting and summarizing gui changes in evolving mobile apps. In *Proceedings of the 33rd ACM/IEEE international conference on automated software engineering*, pp. 543–553, 2018.

Dang Nguyen, Jian Chen, Yu Wang, Gang Wu, Namyong Park, Zhengmian Hu, Hanjia Lyu, Junda Wu, Ryan Aponte, Yu Xia, et al. Gui agents: A survey. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 22522–22538, 2025.

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

Chris Rawles, Sarah Clinckemaillie, Yifan Chang, Jonathan Waltz, Gabrielle Lau, Marybeth Fair, Alice Li, William Bishop, Wei Li, Folawiyo Campbell-Ajala, et al. Androidworld: A dynamic benchmarking environment for autonomous agents. In *International Conference on Learning Representations*, volume 2025, pp. 406–441, 2025.

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

Yibo Shi, Jungang Li, Linghao Zhang, Zihao Dongfang, Biao Wu, Sicheng Tao, Yibo Yan, Chenxi Qin, Weiting Liu, Zhixin Lin, et al. Androtmem: From interaction trajectories to anchored memory in long-horizon gui agents. *arXiv preprint arXiv:2603.18429*, 2026.

Libo Sun, Jiwen Zhang, Siyuan Wang, and Zhongyu Wei. Magnet: Towards adaptive gui agents with memory-driven knowledge evolution. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 28181–28206, 2026.

Yunteng Tan, Zhi Gao, and Xinxiao Wu. Enhancing web agents with a hierarchical memory tree. *arXiv preprint arXiv:2603.07024*, 2026.

Xinyuan Wang, Bowen Wang, Dunjie Lu, Junlin Yang, Tianbao Xie, Junli Wang, Jiaqi Deng, Xiaole Guo, Yiheng Xu, Chen Wu, et al. Opencua: Open foundations for computer-use agents. *Advances in Neural Information Processing Systems*, 38:139756–139806, 2026.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. *arXiv preprint arXiv:2409.07429*, 2024.

Jiayang Wu, Wensheng Gan, Zefeng Chen, Shicheng Wan, and Philip S Yu. Multimodal large language models: A survey. In *2023 IEEE international conference on big data (bigdata)*, pp. 2247–2256. IEEE, 2023.

Qianhui Wu, Kanzhi Cheng, Rui Yang, Chaoyun Zhang, Jianwei Yang, Huiqiang Jiang, Jian Mu, Baolin Peng, Bo Qiao, Reuben Tan, et al. Gui-actor: Coordinate-free visual grounding for gui agents. *Advances in Neural Information Processing Systems*, 38:15101–15128, 2026.

Wenyi Wu, Kun Zhou, Ruoxin Yuan, Vivian Yu, Stephen Wang, Zhiting Hu, and Biwei Huang. Auto-scaling continuous memory for gui agent. *arXiv preprint arXiv:2510.09038*, 2025.

Han Xiao, Guozhi Wang, Hao Wang, Shilong Liu, Yuxiang Chai, Yue Pan, Yufeng Zhou, Xiaoxin Chen, Yafei Wen, and Hongsheng Li. Ui-mem: Self-evolving experience memory for online reinforcement learning in mobile gui agents. *arXiv preprint arXiv:2602.05832*, 2026.

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

Tianbao Xie, Jiaqi Deng, Xiaochuan Li, Junlin Yang, Haoyuan Wu, Jixuan Chen, Wenjing Hu, Xinyuan Wang, Yuhui Xu, Zekun Wang, et al. Scaling computer-use grounding via user interface decomposition and synthesis. *Advances in Neural Information Processing Systems*, 38, 2026.

Yiheng Xu, Zekun Wang, Junli Wang, Dunjie Lu, Tianbao Xie, Amrita Saha, Doyen Sahoo, Tao Yu, and Caiming Xiong. Aguvis: Unified pure vision agents for autonomous gui interaction. *arXiv preprint arXiv:2412.04454*, 2024.

Bofei Zhang, Zirui Shang, Zhi Gao, Wang Zhang, Rui Xie, Xiaojian Ma, Tao Yuan, Xinxiao Wu, Song-Chun Zhu, and Qing Li. Tongui: Internet-scale trajectories from multimodal web tutorials for generalized gui agents. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 12367–12375, 2026.

Chaoyun Zhang, Shilin He, Liqun Li, Si Qin, Yu Kang, Qingwei Lin, Saravan Rajmohan, and Dongmei Zhang. Api agents vs. gui agents: Divergence and convergence. *arXiv preprint arXiv:2503.11069*, 2025.

Hanzhang Zhou, Panrong Tong, Xu Zhang, Quyu Kong, Chenglin Cai, Tianyu Xia, Gongjie Zhang, Jianan Zhang, Long Li, Long Chen, et al. Qwen-ui-agent technical report: Toward next-generation real-world centric foundation gui agents. *arXiv preprint arXiv:2607.28227*, 2026.

Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, et al. Webarena: A realistic web environment for building autonomous agents. In *International Conference on Learning Representations*, volume 2024, pp. 15585–15606, 2024.

# A VERSIONSHIFT BENCHMARK DETAILS

This section provides additional statistics and annotation details for VERSIONSHIFT. Unless otherwise stated, a step denotes an executable GUI action excluding the terminal `finish` record. A forward source–target pair contains executions of the same task on two versions, where the earlier version serves as the source and the later version serves as the target.

## A.1 SCALE AND PLATFORM COVERAGE

VERSIONSHIFT spans 53 applications and websites and 312 tasks across Mobile, Web, and Desktop. The complete platform list is shown in Figure 3.

Table 6 summarizes platform-level statistics. Mobile and Desktop contain longer trajectories than Web, with average lengths of 4.19, 3.83, and 1.93 executable steps, respectively. The shorter average Web trajectories mainly result from the higher variability of web environments across versions, where maintaining consistent task executions is more challenging and often leads to shorter reproducible interaction sequences.

## A.2 ADAPTATION COMPOSITION ACROSS PLATFORMS

Table 7 summarizes the distribution of step-level adaptation modes across platforms. Overall, REUSE, REGROUND, and REPLAN account for 51.2%, 24.1%, and 24.7% of paired target-step instances, respectively. The distribution varies across platforms: Mobile and Desktop are dominated by REUSE, whereas Web contains substantially more REGROUND steps, reflecting more frequent localization changes across Web versions.

## A.3 ACTION DISTRIBUTION ACROSS PLATFORMS

Table 8 summarizes the executable action distribution across platforms. Pointer actions dominate all three platforms, accounting for 77.2–88.8% of actions. The remaining action types reflect platform-specific interaction patterns: Desktop contains substantially more Keyboard operations, Web has a higher proportion of Scroll actions, while Text input appears mainly on Mobile.

## A.4 ANNOTATION RELIABILITY

Figure 4 shows the annotation flow of VERSIONSHIFT, and Figure 5 shows the annotation software interface. All benchmark annotations were produced following predefined labeling guidelines. Each trajectory was first automatically validated for executability and task success. Cross-version adaptation labels were then assigned by comparing source and target interfaces, executed actions, and semantic subgoals. Each assigned REUSE, REGROUND, or REPLAN label was manually verified, and ambiguous cases were revisited before inclusion. The annotation process was conducted by one trained annotator. Therefore, we emphasize rule-based consistency and verification against executable evidence rather than inter-annotator agreement.

## A.5 DATA COLLECTION EXAMPLE

Figure 6 illustrates a representative data instance in VERSIONSHIFT. For each task, we record the complete execution trajectory together with step-level screenshots, actions, UI metadata, and saved interface artifacts. Each executable step is further associated with semantic annotations such as its subgoal. Across software versions, corresponding steps are linked to describe how the same interaction changes, including element identity, localization, and other interface properties.




Figure 3: Applications and websites included in VERSIONSHIFT.

Table 6: **VersionShift statistics by platform.** Steps count executable actions across all recorded trajectories. Avg. and Range describe executable actions per trajectory.

| Platform | Apps | Tasks | Traj. | Pairs | Steps | Avg. | Range |
|---|---|---|---|---|---|---|---|
| Mobile | 17 | 114 | 456 | 684 | 1,910 | 4.19 | 1–11 |
| Web | 24 | 99 | 396 | 594 | 766 | 1.93 | 1–6 |
| Desktop | 12 | 99 | 396 | 594 | 1,515 | 3.83 | 1–16 |
| **Total** | **53** | **312** | **1,248** | **1,872** | **4,191** | **3.36** | **1–16** |

# B VAEA IMPLEMENTATION DETAILS

This section provides implementation details of VAEA beyond the method description in the main paper. We describe source experience construction, training-instance organization, routing supervision and inference, and the optimization configuration.

## B.1 SOURCE EXPERIENCE CONSTRUCTION

**Trajectory normalization.** Each recorded trajectory is converted into a platform-independent step sequence. An action step contains its step index, action type, normalized pointer location when available, non-spatial action arguments, and observable control metadata. Pointer coordinates are normalized to the $[0, 1]^2$ frame of the corresponding screenshot.

**Automatic subgoal generation.** For each source action, we generate a concise semantic subgoal using GPT-4o-mini. The annotation input contains only source-side information, including the source action metadata, interface representation (XML for Mobile/Desktop and HTML, URL, and title information for Web), and the source screenshot. The model is instructed to describe the immediate user intent of each action rather than low-level coordinates or selectors:

```
You are annotating GUI trajectories for a version-shift
benchmark.
Generate one semantic subgoal for each action step.  A
subgoal should describe the immediate user intent of that
step, not low-level coordinates or selectors.  Use a concise
English imperative phrase, usually 4--12 words.  Do not
invent states unsupported by the action metadata, interface
representation, or screenshot.  Return strict JSON only.
```

Table 7: **Step-level adaptation modes in VersionShift.** Statistics are aggregated over all forward source–target version pairs.

| Mode | Mobile | Web | Desktop | Overall |
|---|---|---|---|---|
| REUSE | 1,772 (62.4%) | 157 (13.0%) | 1,318 (57.4%) | 3,247 (51.2%) |
| REGROUND | 447 (15.7%) | 839 (69.6%) | 240 (10.5%) | 1,526 (24.1%) |
| REPLAN | 621 (21.9%) | 210 (17.4%) | 738 (32.1%) | 1,569 (24.7%) |
| **Total** | **2,840** | **1,206** | **2,296** | **6,342** |

Table 8: **Executable action distribution across platforms.** Each cell reports action count and percentage within the corresponding platform. Navigation actions are included in Pointer.

| Action | Mobile | Web | Desktop | Total |
|---|---|---|---|---|
| Pointer | 1,697 (88.8%) | 615 (80.3%) | 1,169 (77.2%) | 3,481 |
| Text | 123 (6.4%) | 9 (1.2%) | 0 (0.0%) | 132 |
| Keyboard | 0 (0.0%) | 0 (0.0%) | 344 (22.7%) | 344 |
| Scroll | 90 (4.7%) | 142 (18.5%) | 2 (0.1%) | 234 |
| **Total** | **1,910** | **766** | **1,515** | **4,191** |

Invalid outputs or missing subgoals are replaced by a deterministic fallback generated from the action type and available control labels. The annotation VLM is used only during offline preprocessing and is never queried during VAEA inference.

**Experience cards.** The implementation serializes the source experience defined in Section 4. A representative serialized card is:

```
{
 "source_step_index": 1,
 "subgoal": "Open the Settings page",
 "action_semantics": "Open the Settings page",
 "action_type": "tap",
 "locator_evidence": {
  "old_control": {
   "text": "Settings",
   "resource_id": "menu_settings",
   "class_name": "TextView"
  }
 },
 "copy_action": {
  "action_type": "tap",
  "tap_point": [0.34, 0.49]
 }
}
```

`subgoal`, `action_semantics`/`action_type`, `locator_evidence`, and `copy_action` correspond to $g_i^s$, $\bar{a}_i^s$, $\ell_i^s$, and the direct-reuse action represented by $p_i^s$, respectively. Locator evidence and source coordinates are used only for routing and direct reuse, and are removed before REGROUND action generation. Direct reuse is supported only for valid `click`/`tap` actions; otherwise, `copy_action:null`. Cards are constructed from source-side information only, while cross-version annotations and target actions are used only for training supervision. Full source screenshots are not retained.

### B.2 TRAINING INSTANCE CONSTRUCTION

VAEA is trained as a two-task model with the prefixes `<ADAPT>` and `<ACT>` using the same GUI backbone and LoRA adapter. Both tasks use only the current target screenshot as Image 0; source screenshots are never provided.

**`ADAPT` instances.** For each source-conditioned target step, the ADAPT input contains the task instruction, target history, current screenshot, and source experience cards. The output is:

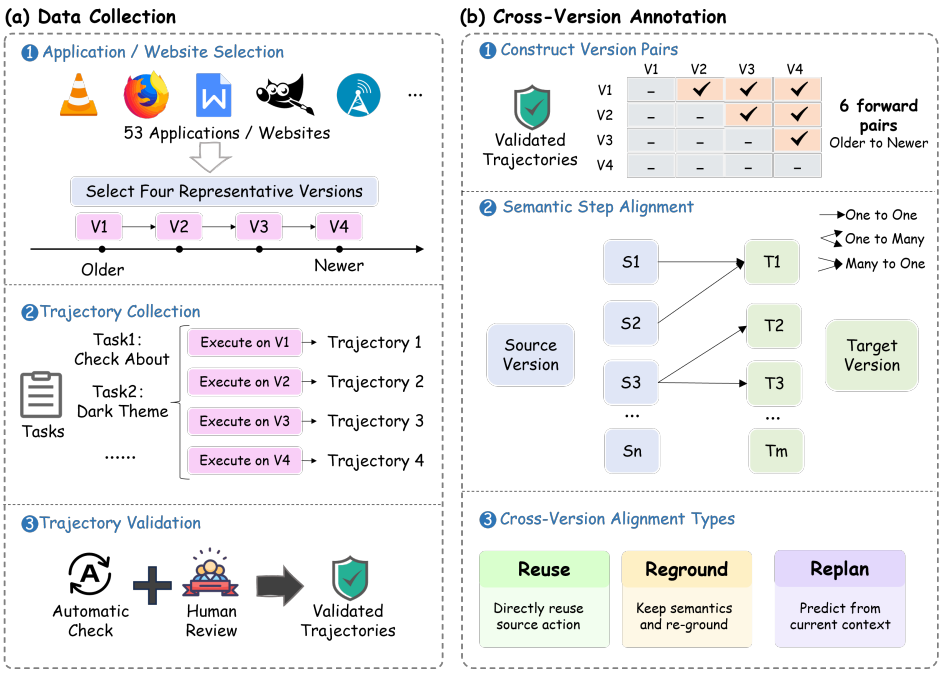


Figure 4: **VersionShift benchmark construction.** **(a)** We collect and validate successful executions of the same task across four chronological software versions. **(b)** Cross-version trajectories are paired, aligned by semantic subgoals, and annotated with labels REUSE, REGROUND, and REPLAN.

`{"source_steps":[indices],"reuse":true/false}`.

The selected indices identify source steps whose semantics remain applicable to the current target step. The reuse field supervises whether a uniquely selected `copy_action` remains valid at its normalized source location; its `true`/`false` logits are used to derive the reuse score $r_j$ at inference.

The ADAPT prompt is:

```
<ADAPT>
Select source steps whose action semantics are applicable
to the immediate next action on current Image 0.  Return []
if none apply.  Then predict whether the single selected
copy_action would execute the correct action on Image 0
at its unchanged normalized location.  Set reuse=false
for empty or multiple selection, or when copy_action
is unavailable.  Output exactly JSON in this order:
{"source_steps":[indices],"reuse":true or false}.  No
explanation.
EVIDENCE_JSON:<compact evidence JSON>
```

**ACT instances.** The Actor receives the current target context and selected source semantics without localization information:

```
{
 "source_step_index": 1,
 "subgoal": "Open the Settings page",
 "action_semantics": "Open the Settings page",
 "action_type": "tap"
}
```

Both `locator_evidence` and `copy_action` are removed before action generation, requiring the Actor to reground the retained semantics on Image 0.

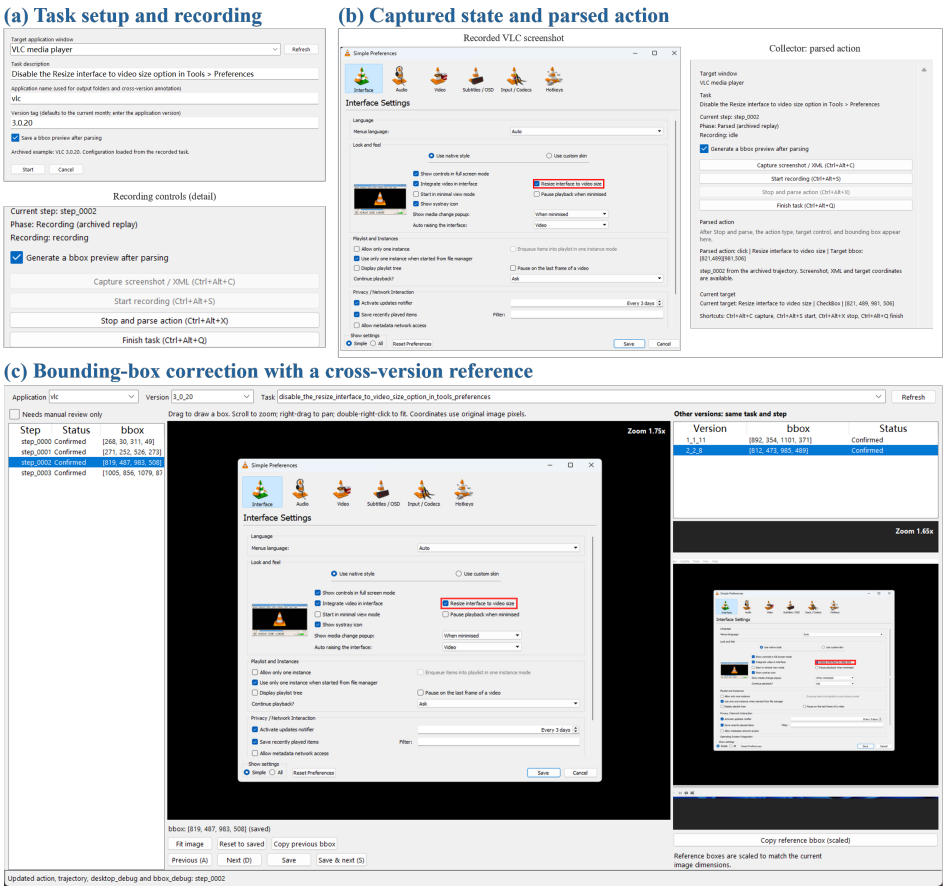


Figure 5: **VersionShift data collection and annotation interface.**

The ACT prompt is:

```
<ACT>
You are a GUI agent.  Select exactly one next executable
action for the task on the current target screen.  Treat
source-version experience as a semantic workflow hint, not
a replay script.  Ground the next action independently on
Image 0 and never copy source numeric coordinates.  Return
one JSON object only following the platform-specific action
schema.
```

We construct an ACT instance for every target step with applicable source experience, including cases whose routing supervision permits direct reuse. Thus, a step can fall back to REGROUND when its predicted reuse score does not satisfy the inference threshold. When no source experience is applicable, ACT receives no experience card and learns REPLAN. We additionally include target-only Actor anchors to reduce dependence on source experience availability.

All model-visible inputs exclude supervision-only information, including target actions, routing labels, and cross-version correspondence annotations.

### B.3 ROUTING SUPERVISION AND INFERENCE

The routing mode $S_\theta$ predicts applicable source-card indices $\hat{\mathcal{I}}_j$ and a Boolean reuse field. Source indices are supervised by source–target semantic alignment, while the reuse label indicates whether a uniquely selected pointer action remains valid at its original normalized location.




Figure 6: **Example of data collection and annotation in VERSIONSHIFT.** (a) A complete task trajectory, (b) the recorded state, action, and UI metadata for one step, and (c) the corresponding semantic and cross-version annotations.

At inference, the reuse confidence is derived from the language-model logits:

$$r_j = \frac{\exp z_j(\texttt{true})}{\exp z_j(\texttt{true}) + \exp z_j(\texttt{false})}. \tag{6}$$

An empty selection yields REPLAN; a unique selection with a valid `copy_action` and $r_j \geq \delta$ yields REUSE; all other non-empty selections yield REGROUND.

### B.4 OPTIMIZATION DETAILS

We implement ADAPT and ACT using a single Qwen3-VL backbone with a shared LoRA adapter. All base-model parameters remain frozen, with LoRA applied to the `q_proj` and `v_proj` modules.

## C BASELINE IMPLEMENTATION DETAILS

This section describes the baseline implementations used in our experiments. For the controlled Qwen3-VL comparisons, all trainable methods use the same application-disjoint split, backbone family, action protocol, action scorer, LoRA configuration, and optimizer budget. Source-conditioned methods may contain multiple instances for the same target action due to different source trajectories, but introduce no additional unique target-action supervision. Because source-conditioned inputs differ in token and image length, we use update-matched training rather than exact FLOP matching. No method has access to target-side annotations or cross-version correspondence labels during inference.

**Target-only SFT.** Target-only SFT is trained on 2,025 unique target-side state–action examples. The model receives only the target instruction, screenshot, and action history and predicts the next target action. This baseline measures performance without historical experience.

**Experience Replay SFT.** Experience Replay SFT augments Target-only SFT with 868 source-version state–action examples from the training split. Source examples are treated as ordinary action-prediction samples without trajectory context and are not provided at inference. This baseline measures the effect of parametric exposure to source-version experience.

Table 9: **VAEA optimization configuration shared by the 4B and 8B backbones.**

| Configuration | Value |
|---|---|
| Precision | BF16 |
| LoRA rank / scale / dropout | $16/32/0.05$ |
| Trainable modules | `q_proj`, `v_proj` |
| Batch size | 16 |
| Learning rate | $1 \times 10^{-4}$ |
| Warmup updates | 89 |
| Maximum sequence length | 16,384 |
| Training seeds | $\{0\}$ |

**Raw History SFT.** Raw History SFT uses 4,053 source-conditioned examples. The model receives the target state together with the complete source trajectory, including source actions and screenshots, and directly predicts the target action without selection or adaptation. This baseline measures direct conditioning on historical trajectories.

**Compact History SFT.** Compact History SFT replaces the raw trajectory with coordinate-free semantic experience cards, retaining action semantics and non-spatial arguments while removing source screenshots, coordinates, and localization cues. This baseline tests whether semantic compression alone is sufficient for cross-version adaptation.

**Two-Pass variants.** Raw History + Two-Pass and Compact History + Two-Pass add a second verification pass to their respective first-pass predictions. The second pass can revise the initial action, controlling for the additional inference computation used by VAEA without explicit validity-aware adaptation.

**Supervised Selection + Actor.** This baseline uses the same experience cards as VAEA and correspondence supervision to select applicable source steps. The Actor always generates the target action from the selected coordinate-free semantics, without validity-aware direct reuse. This isolates the contribution of source-step selection.

**Platform-specific memory baselines.** We additionally evaluate AWM on Web and MAGNET on Mobile following their intended platform settings and official implementations, with adaptations required by the VERSIONSHIFT protocol.

- **AWM.** Following the original AWM method, we induce parameterized workflows from source experience and provide them as additional context for action prediction. We adapt its observation and action interfaces to VersionShift: workflows are constructed from the paired source-version trajectory, and the actor predicts screenshot-grounded actions in the benchmark's canonical action space. Thus, workflow induction and context augmentation follow the original method, while the experience scope and input/output interfaces are adapted to our evaluation protocol.
- **MAGNET.** Following the original MAGNET method, we retain procedural memory, stationary memory, the planner–actor architecture, and MiniCPM-Embedding-based semantic retrieval. We modify the stationary-memory construction stage by using the provided source-version element annotations instead of OmniParser detections. Specifically, annotated bounding boxes provide element crops, while element attributes and source actions provide functional descriptions, supplemented by Qwen when necessary. We also restrict memory construction to the paired source trajectory and freeze memory during target evaluation. These changes adapt MAGNET to the offline comparison; its online memory-evolution mechanism is not evaluated.

**Common training and evaluation protocol.** All controlled trainable baselines use BF16 training, the same LoRA parameterization, effective batch size of 16, learning rate of $1 \times 10^{-4}$. Predictions use the same action canonicalization and scorer, and all methods are evaluated on identical target states within each evaluation split.

Table 10: **Application-disjoint data splits.** All splits include the six forward version pairs per task. Traj. denotes source–target trajectory pairs. Target Steps count unique target action steps, whereas Paired Steps count target steps under their corresponding source–target pairs. Terminal `finish` records are excluded.

| Split | Apps | Tasks | Traj. | Target Steps | Paired Steps |
|---|---|---|---|---|---|
| Train | 35 | 207 | 1,242 | 2,025 | 4,053 |
| Dev | 9 | 50 | 300 | 567 | 1,125 |
| Test | 9 | 55 | 330 | 583 | 1,164 |

# D Additional Experimental Results

We partition VersionShift into application-disjoint training, development, and test splits. The development split is used exclusively for model selection and reuse-threshold calibration, while the held-out test split is used only for final evaluation.

**Analysis across adaptation types.** The central hypothesis of VAEA is that source experience requires different adaptation behaviors after software evolution. We therefore compare VAEA with fixed policies that always follow Reuse, Reground, or Replan. Oracle Routing uses benchmark adaptation annotations to select the corresponding fixed adaptation branch for each instance and serves only as a diagnostic oracle; these annotations are unavailable during inference.

Table 11 shows that each fixed policy performs best on its corresponding adaptation type: Always Reuse on Reuse, Always Reground on Reground, and Always Replan on Replan. This confirms that no single adaptation behavior is optimal across different types of version changes.

Without access to adaptation labels at inference, VAEA achieves the highest overall Step accuracy among non-oracle methods on both backbones. The remaining gap to Oracle Routing indicates further headroom in learned routing decisions.

**Quality of the `ADAPT` predictions.** We further examine the intermediate predictions of the `ADAPT` stage. Because VAEA does not explicitly predict a three-way adaptation label, the adaptation branch is derived from its routing output. A unique selection with a valid `copy_action` and $r_j \geq \delta$ yields Reuse; a non-empty selection without direct reuse yields Reground; and an empty selection yields Replan.

Table 12 reports source-step selection, direct-reuse quality, and derived branch accuracy. Selection Exact reaches 76.8% and 80.4% for Qwen3-VL-4B and Qwen3-VL-8B, respectively, while the higher F1 scores indicate that non-exact selections often still recover relevant source experience.

Direct reuse achieves high precision but moderate recall, indicating a conservative routing policy that avoids uncertain direct copying. Such cases instead fall back to Reground or Replan. The remaining gap in branch accuracy suggests further headroom in source-step selection and reuse-validity estimation.

**Sensitivity to the direct-reuse threshold.** Direct reuse involves a trade-off between coverage and reliability. We sweep the reuse threshold $\delta$ on the development split while keeping source selection and fallback action generation fixed, and freeze the selected threshold for test evaluation.

As $\delta$ increases, reuse coverage decreases while precision improves. For Qwen3-VL-4B and Qwen3-VL-8B, the selected operating points are $\delta = 0.90$ and $\delta = 0.95$, respectively, reaching reuse precision of 95.0% and 96.1%. The different operating points suggest that reuse confidence should be calibrated separately for each backbone.

Effectively disabling direct reuse retains strong performance, indicating that semantic transfer and target-side grounding account for much of the gain. High-confidence direct reuse nevertheless provides a modest additional improvement.

**Source-version gap analysis.** We group all six forward source–target pairs by their ordinal version gap: $d = 1$ includes $v_1 \to v_2$, $v_2 \to v_3$, and $v_3 \to v_4$; $d = 2$ includes $v_1 \to v_3$ and $v_2 \to v_4$; and

Table 11: **Behavior across benchmark-annotated adaptation types.** Step accuracy is reported separately on REUSE, REGROUND, and REPLAN instances. Fixed policies follow a single adaptation branch, while VAEA selects the branch adaptively.

| Backbone | Policy | Reuse ↑ | Reground ↑ | Replan ↑ | Overall ↑ |
|---|---|---|---|---|---|
| Qwen3-VL-4B | Always Reuse | **87.81** | 68.37 | 34.78 | 72.42 |
| | Always Reground | 84.84 | **80.95** | 49.13 | 76.80 |
| | Always Replan | 74.84 | 65.99 | **61.30** | 69.93 |
| | **VAEA** | 87.66 | 78.91 | 56.09 | **79.21** |
| | *Oracle Routing* | 87.81 | 80.95 | 61.30 | 80.84 |
| Qwen3-VL-8B | Always Reuse | **88.75** | 71.43 | 37.83 | 74.31 |
| | Always Reground | 86.09 | **82.31** | 51.30 | 78.26 |
| | Always Replan | 79.06 | 68.71 | **63.48** | 73.37 |
| | **VAEA** | 88.59 | 80.27 | 59.57 | **80.76** |
| | *Oracle Routing* | 88.75 | 82.31 | 63.48 | 82.13 |

Table 12: **Diagnostic quality of the `ADAPT` stage on the held-out test split.** Selection Exact measures exact recovery of the applicable source-step set, while Selection F1 gives partial credit for overlapping selections. Reuse precision and recall evaluate direct-reuse decisions using the operating point selected on the development split. Branch Acc. measures the accuracy of the derived REUSE/REGROUND/REPLAN branch.

| Backbone | Selection Exact ↑ | Selection F1 ↑ | Reuse Prec. ↑ | Reuse Recall ↑ | Branch Acc. ↑ |
|---|---|---|---|---|---|
| Qwen3-VL-4B | 76.8 | 84.7 | 95.7 | 58.9 | 78.6 |
| Qwen3-VL-8B | 80.4 | 87.9 | 96.3 | 63.7 | 81.9 |

$d = 3$ contains $v_1 \to v_4$. For each gap, we report the average over the corresponding forward pairs. The gap reflects only version ordering and does not directly measure the magnitude of interface change.

Table 14 shows a consistent trend across both backbones: larger ordinal gaps are associated with lower Raw History Step accuracy and higher NTR. At the same time, the gain of VAEA over Raw History increases from $d = 1$ to $d = 3$. These results suggest that validity-aware adaptation becomes increasingly beneficial for more temporally distant source versions, where Raw History also exhibits higher negative transfer.

**Cross-platform performance.** Finally, we examine whether the overall gain of VAEA is concentrated on a particular platform. Table 15 shows that VAEA improves over Raw History on Mobile, Web, and Desktop for both backbones, indicating that its benefit is not driven by a single environment. The largest consistent gain appears on Web, while the magnitude of improvement varies across platforms and backbone sizes.

The NTR breakdown further shows that the effect of source information differs across platforms. For example, Compact History substantially reduces NTR on Web, while VAEA achieves higher Step accuracy with consistently low NTR across both backbones. These results suggest that validity-aware adaptation provides complementary benefits across different platform-specific interface changes.

## E LIMITATIONS

While VERSIONSHIFT provides a systematic benchmark for studying cross-version experience adaptation in GUI agents, several limitations remain and will be addressed in future extensions.

**Offline evaluation.** VERSIONSHIFT currently evaluates agents on pre-collected target states rather than through fully online interaction with each software version. While this design enables reproducible comparison across agents and source–target pairs, it does not capture error propagation and

Table 13: **Development-set sensitivity to the direct-reuse threshold.** Reuse coverage is the fraction of paired steps that directly reuse a source pointer action, while reuse precision measures the correctness of accepted reuse decisions. Bold denotes the operating point selected on the development split.

| Backbone | $\delta$ | Step ↑ | Traj. ↑ | Reuse coverage | Reuse precision ↑ |
|---|---|---|---|---|---|
| Qwen3-VL-4B | 0.00 | 72.80 | 44.67 | 61.2 | 71.6 |
| | 0.50 | 76.44 | 49.33 | 32.8 | 85.1 |
| | 0.70 | 77.69 | 51.33 | 24.6 | 89.9 |
| | 0.80 | 78.22 | 52.00 | 18.1 | 94.1 |
| | **0.90** | **78.84** | **52.67** | **12.4** | **95.0** |
| | 0.95 | 78.67 | 52.00 | 7.8 | 97.7 |
| | 1.00 | 78.04 | 51.33 | 0.0 | – |
| Qwen3-VL-8B | 0.00 | 71.73 | 46.00 | 69.4 | 68.1 |
| | 0.50 | 76.98 | 50.67 | 36.3 | 82.8 |
| | 0.70 | 78.67 | 52.67 | 27.7 | 89.4 |
| | 0.80 | 79.56 | 53.33 | 19.6 | 93.6 |
| | 0.90 | 79.91 | 54.00 | 14.0 | 94.9 |
| | **0.95** | **80.09** | **54.67** | **9.1** | **96.1** |
| | 1.00 | 79.56 | 54.00 | 0.0 | – |

Table 14: **Cross-version transfer across source-version gaps.** Results are averaged over all paired steps with the same ordinal version gap $d$. We report Raw History Step accuracy and NTR, together with VAEA Step accuracy and its gain over Raw History.

| Backbone | Gap $d$ | Raw Step ↑ | Raw NTR ↓ | VAEA Step ↑ | VAEA Gain ↑ |
|---|---|---|---|---|---|
| Qwen3-VL-4B | 1 | 82.05 | 2.74 | 83.13 | +1.09 |
| | 2 | 77.20 | 4.92 | 80.40 | +3.20 |
| | 3 | 69.06 | 8.72 | 73.85 | +4.79 |
| Qwen3-VL-8B | 1 | 83.70 | 4.35 | 84.79 | +1.09 |
| | 2 | 78.84 | 7.08 | 82.04 | +3.20 |
| | 3 | 69.57 | 11.45 | 75.56 | +5.98 |

recovery during interaction. We will extend VERSIONSHIFT with online evaluation to better assess cross-version adaptation in interactive settings.

**Version coverage.** Each task in VERSIONSHIFT currently covers four chronologically ordered versions, representing only a subset of the continuous evolution of real-world software. We will expand the benchmark with longer version histories and denser temporal coverage to capture more diverse evolution patterns.

**Scope of cross-version tasks.** To enable controlled comparison, VERSIONSHIFT focuses on tasks whose intent remains consistent and whose initial conditions can be reproduced across versions. This excludes some more disruptive forms of software evolution, such as removed functionality, version-specific capabilities, and tasks heavily dependent on dynamic external content or services. We will broaden the task scope in future versions of the benchmark to cover these more challenging cross-version scenarios.

Table 15: **Cross-platform breakdown.** Step denotes step-level accuracy and NTR denotes negative transfer relative to the corresponding Target-only model. Overall Step is computed on the full test split and matches the aggregate result reported in the main paper.

| Backbone | Method | Mobile | | Web | | Desktop | | Overall |
|---|---|---|---|---|---|---|---|---|
| | | Step ↑ | NTR ↓ | Step ↑ | NTR ↓ | Step ↑ | NTR ↓ | Step ↑ |
| Qwen3-VL-4B | Target-only | 74.24 | – | 48.70 | – | 55.23 | – | 63.57 |
| | Raw History | 80.17 | 5.08 | 60.00 | 5.65 | 79.94 | 5.81 | 76.12 |
| | Compact History | 75.08 | 6.27 | 56.96 | 4.78 | 71.51 | 6.40 | 70.45 |
| | Raw History + Two-Pass | 81.53 | 4.75 | 62.61 | 3.04 | 80.81 | 5.81 | 77.58 |
| | **VAEA** | **83.39** | **3.39** | **64.78** | **2.61** | **81.69** | **3.49** | **79.21** |
| Qwen3-VL-8B | Target-only | 79.49 | – | 53.91 | – | 61.05 | – | 68.99 |
| | Raw History | 82.54 | 8.47 | 61.74 | 8.26 | 78.78 | 5.81 | 77.32 |
| | Compact History | 77.97 | 7.46 | 61.74 | **2.17** | 74.42 | 7.27 | 73.71 |
| | Raw History + Two-Pass | 83.90 | 6.27 | 64.78 | 3.91 | 78.20 | **4.94** | 78.44 |
| | **VAEA** | **84.92** | **2.54** | **66.52** | 2.61 | **83.14** | **4.94** | **80.76** |