Not All Graphs Are Maps: A Task-Agnostic Exploration Framework for Mobile GUI Navigation

Adamo Orsini[1,2*], **Seyed Mehdi Ayyoubzadeh**[3*], **Sadman Rakib Pinon**[1,2]
Hannuo Zhang[1,2], **Koosha Gholipour Baradari**[1], **Yuanhao Yu**[3]
Xinxin Zuo[1], **Yang Wang**[1,2]

[1]Concordia University [2]Mila – Quebec AI Institute [3]McMaster University

{adamo.orsini,sadmanrakib.pinon,hannuo.zhang,koosha.gholipourbaradari}@mail.concordia.ca

s.mehdi.ayyoubzadeh@gmail.com dr.yhyu@outlook.com

{xinxin.zuo,yang.wang}@concordia.ca

Abstract

Modern mobile GUI agents can perceive visible screens and execute local actions, but they still lack persistent app-specific navigation knowledge: where functions live, which pages contain them, and which screens are equivalent across repeated visits. We introduce TAG-Nav, a Task-Agnostic Graph Navigation framework that wraps existing GUI-agent backbones with reusable destination memory. Offline exploration is distilled into compact records of app functions, target pages, and waypoint routes before any evaluated instruction is seen. When a future task's destination is covered, TAG-Nav retrieves the destination and provides route guidance without task-specific demonstrations. This separates mobile GUI *destination navigation* from full task completion: many tasks share the prefix of reaching the right page, while final actions may enter content, send data, make purchases, delete information, or change sensitive settings. To evaluate route finding without downstream execution confounds, we introduce DestBench, a navigation-only protocol that asks whether an agent reaches the correct pre-action destination screen. DestBench enables a controlled graph-memory study comparing no-memory agents with random, automated BFS, and clean graph memory. Graph-derived memory improves strict navigation success rate of UI-TARS-1.5-7B from 52.4% without graph memory to 58.3% with automated BFS memory, and improves Qwen-VL-Max success rate from 59.5% to 71.4% on the same conditions. Together, TAG-Nav and DestBench[1] show that destination coverage and route cleanliness strongly shape graph-memory value, making graph quality a measured variable rather than an assumed benefit of exploration.

[*]These authors contributed equally.

[1]https://tag-nav-authors.github.io/tag_nav/

1 Introduction

Modern multimodal GUI agents perceive screens and execute visible actions well, but they lack a persistent map of the apps they operate (Zhang et al., 2025a; Wen et al., 2024; Qin et al., 2025). When asked to reach a deep setting, account-management page, privacy control, or app-specific tool, the agent must re-infer where the function lives and how to navigate there from the current screen on every task. Many user requests share the prefix of arriving at a destination before any task-specific (and often sensitive or irreversible) final action: reaching an account-privacy screen is reusable navigation, while toggling the control itself is task-specific execution.

We study mobile GUI *destination navigation* as a problem distinct from full task completion, focused on whether the agent reaches the correct pre-action destination screen.

The standard response is to explore the app offline and retrieve from the resulting graph (Wen et al., 2024; Guan et al., 2025; Xie et al., 2025; Chen et al., 2025b), but exploration does not automatically yield a useful map: a large random graph can have low target coverage and noisy routes, while a small clean graph can be highly navigable. Two graph-construction properties decide whether explored memory actually helps: *target coverage* of the destinations users request, and *route cleanliness* from the app root to those destinations. We argue that graphs satisfying both, built by task-agnostic breadth-oriented exploration and post-processed into destination-anchored deduplicated routes, are the design that turns offline exploration into navigation memory that wraps any visual GUI actor.

We instantiate this design in TAG-Nav, a **T**ask-**A**gnostic **G**raph **Nav**igation framework that wraps existing visual GUI agents with destination mem-

Findings of the Association for Computational Linguistics: IJCNLP-AACL 2026, pages 3772–3794
November 6-10, 2026 ©2026 Association for Computational Linguistics

ory without retraining. TAG-Nav explores apps with task-agnostic local breadth-first search (Local-BFS) and post-processes the result into compact destination records (target page summary, intents, waypoint route, transition hints); at inference time the matching record is injected as plain text into an unmodified backbone such as UI-TARS (Qin et al., 2025). The retrieved unit is a destination plus its route, not a per-task trajectory replay, so TAG-Nav can be evaluated under different graph-construction conditions while holding the actor and task set fixed.

To evaluate this design on real apps, we introduce DestBench, a navigation-only protocol on real Android apps that judges whether the agent reaches the correct pre-action destination screen, with graph-aware target coverage and route cleanliness diagnostics. End-to-end task benchmarks would otherwise conflate memory quality with downstream execution, blurring the properties we want to measure. On 84 DestBench tasks spanning six apps, TAG-Nav achieves higher observed navigation success than a no-memory baseline across two GUI backbones (Table 3). Controlled comparisons against random-exploration and clean manual-graph conditions examine the roles of target coverage and route cleanliness (Table 6).

Our contributions are:

- **TAG-Nav**, a task-agnostic destination-memory module that wraps an unmodified visual GUI actor: offline exploration becomes compact destination records (target page, waypoint route, transition hints) injected as plain text at inference.
- **DestBench**, a navigation-only page-reaching protocol on real Android apps with graph-aware diagnostics for target coverage and route cleanliness.

2 Related Work

Mobile GUI Agents and Visual Grounding. Mobile GUI agents translate natural language instructions and visual observations into interface actions. Smartphone agents and Android automation systems showed that language models can operate mobile apps through screenshots, UI trees, and action histories (Zhang et al., 2025a; Wen et al., 2024; Wang et al., 2024, 2025b). More recent general-purpose GUI backbones such as CogAgent and UI-TARS improve perception, grounding, and multi-step action generation across GUI tasks (Hong et al., 2024; Qin et al., 2025; Wang et al., 2025a). Parallel work on visual grounding and screen parsing, including SeeClick, UGround, and OmniParser, shows that locating the correct on-screen element is itself a central bottleneck for visual GUI agents (Cheng et al., 2024; Gou et al., 2025; Lu et al., 2024). TAG-Nav is complementary to these efforts: it does not replace the actor or the grounding model, but supplies app-specific destination memory that tells an existing actor where to go. Table 1 summarizes how TAG-Nav differs from prior graph- and retrieval-based systems on two axes that anchor the rest of this section: *what is retrieved online* and *whether the memory wraps an unmodified actor*; a per-system breakdown (stored memory, retrieved unit, coupling) is given in Appendix G.

Systems	What is retrieved online	Actor
AutoDroid, AppAgent (Wen et al., 2024; Zhang et al., 2025a)	Per-element hints or app docs	No
KG-RAG, GUI-explorer (Guan et al., 2025; Xie et al., 2025)	Trajectory pairs or current-screen cues	No
PG-Agent, GraphPilot (Chen et al., 2025b; Yu et al., 2026)	Page-graph guidelines or one-shot plans	No
TAG-Nav	**Destination page + waypoint route**	**Yes**

Table 1: Positioning of TAG-Nav against representative graph- and memory-augmented GUI agents. Two axes capture the main difference: the *unit of memory* retrieved online, and whether that memory wraps an *unmodified* stepwise visual actor. Detailed per-system positioning (exploration mode, graph structure, actor pipeline) is deferred to Appendix G.

Mobile-Agent Benchmarks. Mobile-agent benchmarks have expanded from UI-control datasets to interactive task-completion environments. Android in the Wild and AndroidControl study large-scale supervised UI-control data with high- and low-level instructions (Rawles et al., 2023; Li et al., 2024), while AndroidLab provides a reproducible Android training and evaluation framework (Xu et al., 2025b). AndroidWorld, SPA-Bench, Mobile-Bench, and MobileWorld evaluate increasingly realistic mobile-agent task completion, including dynamic environments, multi-step planning, API/tool use, and user interaction (Rawles et al., 2025; Chen et al., 2025a; Deng et al., 2024; Kong et al., 2026). Other recent

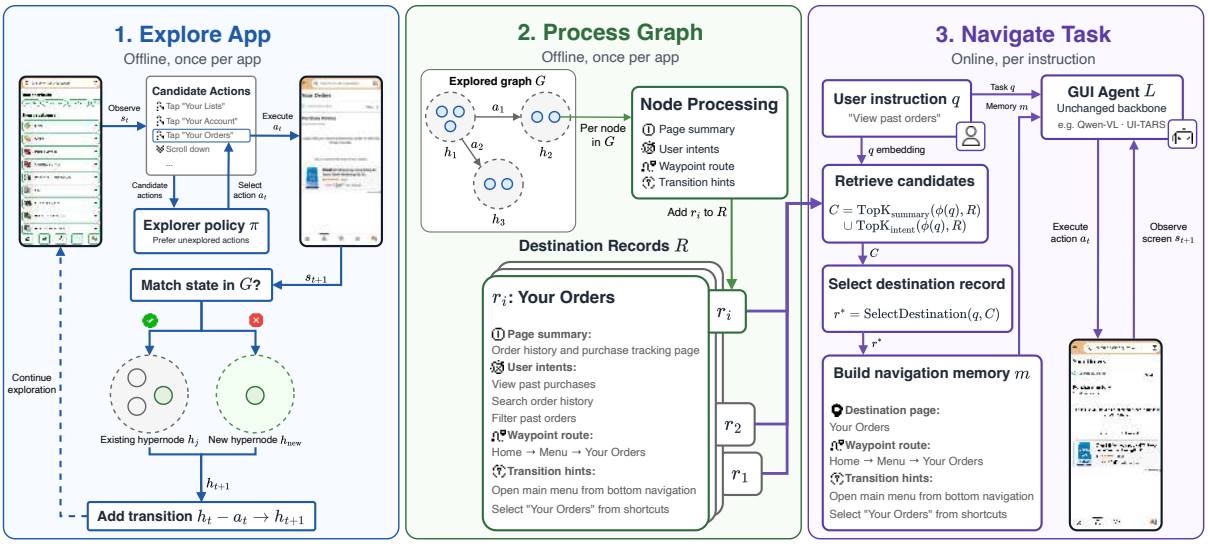


Figure 1: Overview of TAG-Nav on an Amazon instruction. Offline exploration builds a hypernode app graph, post-processing distills indexed states into destination records, and online navigation retrieves candidate records, selects r^*, and formats the selected record as destination memory for an unchanged GUI agent. The figure writes r_i for the formal record $r_{h,n}$ with $i = (h, n)$ and uses s_t only as schematic exploration-state notation; the formal online/evaluation trajectory uses screenshots x_t.

benchmarks stress cross-app navigation, multiple valid paths, noisy apps, and online procedural judging (Lu et al., 2025; Xu et al., 2025a; Im et al., 2026; Chai et al., 2026). These benchmarks are valuable, but their success criteria usually mix navigation with downstream execution: typing user-specific content, changing app state, sending information, selecting items, or satisfying procedural completion checks. Because they sample task completion across many apps, they also provide limited visibility into how well app-specific memory covers the functional regions of a single target app. DestBench instead isolates page-reaching navigation by asking whether the agent reaches the pre-action destination screen for a given task. This makes it possible to evaluate graph-derived navigation memory through target coverage, target retrieval, path existence, route cleanliness, and strict page matching rather than only final task success.

App Exploration and App-Specific Memory. Several systems build app-specific knowledge before task execution. Earlier UI-guided testing tools such as DroidBot built app state-transition models through device interaction, providing an important predecessor to app exploration for automation (Li et al., 2017). AutoDroid explores Android apps, builds a UI transition graph, and turns view-hierarchy/accessibility states into simplified HTML prompts and simulated-task app memory (Wen et al., 2024). Its memory is structured around explored UI trees, elements, and actions rather than visual screen perception, making it a strong early example of exploration-based app memory. AppAgent and AppAgent-v2 use autonomous or manual exploration to build app documents for later task execution (Zhang et al., 2025a; Li et al., 2025). Their memory is element- and document-centered: per-element notes consulted on the current screen, not destination memory keyed to a target page. GUI-Xplore uses pre-recorded exploration videos and converts them into GUI transition graphs (Sun et al., 2025). GUI-Xplore is primarily evaluated as an offline app-understanding setup over those videos, whereas TAG-Nav studies graph memory under live device interaction with a stepwise actor. GUI-explorer autonomously explores apps and mines transition-aware knowledge for dynamic guidance (Xie et al., 2025). Across this family, exploration-derived memory is largely treated as an assumed benefit; what varies between systems is the exploration policy and the memory format, not a controlled comparison of when the explored graph yields useful navigation memory. Our narrower question is when exploration becomes reusable navigation memory: whether the graph contains destination pages, whether retrieval selects the right target, and whether the recovered route is clean enough to guide a future page-reaching task.

Graph and Retrieval-Augmented GUI Agents. Graph and retrieval memories are increasingly common in GUI-agent systems. KG-RAG transforms UI transition graphs into a vector database of intent-trajectory pairs and retrieves relevant tra-

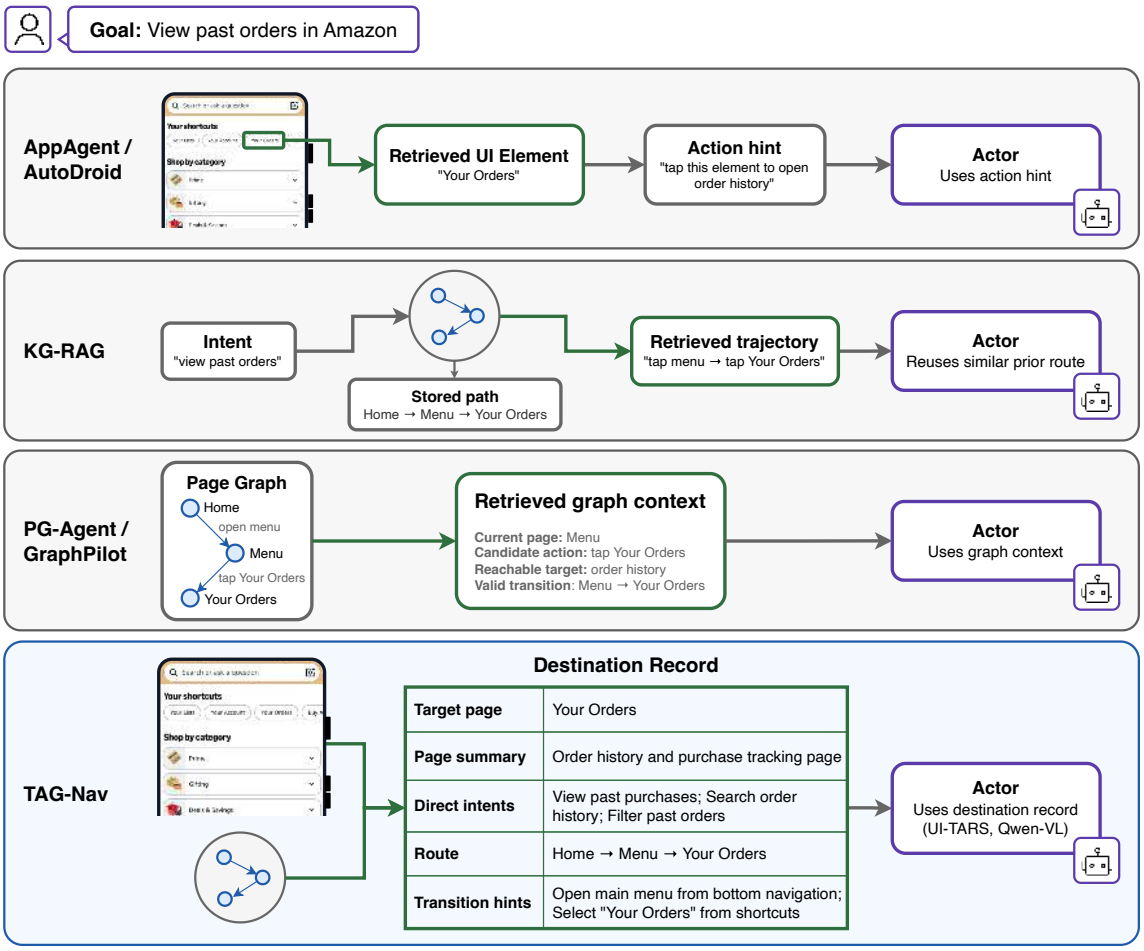


Figure 2: Memory units retrieved at inference by representative graph- and memory-augmented GUI agents. AutoDroid and AppAgent retrieve a per-element hint anchored to a specific UI control; KG-RAG retrieves an intent-trajectory pair (a flat replay of how a similar past task was completed); PG-Agent and GraphPilot retrieve page-level guideline text. TAG-Nav instead retrieves a structured *destination record* with a named target, page summary, route, and transition hints, and injects it into an unmodified visual actor (UI-TARS, Qwen-VL). The categorical difference between TAG-Nav and prior work is the *shape* of the retrieved memory unit, not the presence of memory.

jectories at inference time (Guan et al., 2025). PG-Agent reconstructs page graphs from episodes and retrieves page-graph guidelines for multi-agent planning (Chen et al., 2025b). GraphPilot constructs app-specific knowledge graphs and uses them for low-latency action-sequence generation (Yu et al., 2026). HyMEM studies graph-structured self-evolving memory for GUI agents (Zhu et al., 2026), while AndroTMem focuses on anchored within-task memory in long-horizon Android tasks (Shi et al., 2026). The closest systems differ in what memory they retrieve and how it is consumed: KG-RAG retrieves intent-trajectory pairs for action decisions, GUI-explorer retrieves transition-aware current-screen knowledge for dynamic guidance, PG-Agent retrieves screen-conditioned page-graph guidelines for multi-agent planning, and GraphPilot uses a knowledge graph to plan near one-shot action sequences with a validator. TAG-Nav differs along two axes that prior work does not exercise together (see Fig. 2): the online unit is a destination page with a compact waypoint route consumed by an unmodified stepwise visual actor, and graph quality is measured by comparing no-memory, random, automated-BFS, and clean-graph conditions under the same actor and tasks.

3 TAG-Nav

3.1 Overview

TAG-Nav is a navigation-memory module for mobile GUI agents that separates app-specific discovery from online task execution. Offline exploration builds an app graph over hypernode-clustered observations, then post-processing turns each indexed graph state into a compact destination record describing what the page is for and how it can be reached. At inference time, TAG-Nav retrieves the destination record for the user instruction, for-

mats it as a plain-text destination-memory block, and injects it into an existing GUI-agent backbone without training a new actor or replacing the actor's visual grounding module. Figure 1 summarizes the full pipeline: the offline input is an app/device configuration and the offline output is an app graph plus target-indexed records; the online input is a natural-language instruction and the online output is the executed trajectory together with the evaluator's terminal status.

3.2 Offline App Exploration

TAG-Nav represents an app as a directed graph $G = (\mathcal{H}, E)$. A hypernode $h \in \mathcal{H}$ groups observed UI states that the matcher judges to be the same screen or screen family using visual or layout evidence. Each member observation stores its screenshot, parsed UI or layout evidence when available, and metadata collected during exploration. An edge $e = (h_i, a, h_j)$ records an executable transition from source hypernode h_i to destination hypernode h_j, where a includes the action type, an action description, and available element evidence. The graph is therefore a state abstraction over repeated screenshots rather than a literal log of every frame.

Auto-BFS is our automated graph-construction condition, which builds the app graph through repeated walks from the app's initial screen. It combines walks using the local breadth-first search exploration policy (Local-BFS) with random-action walks. Local-BFS walks probe unexplored actions and verify return to the current screen before probing another action, skipping actions already explored on matched sibling nodes within the same hypernode. Observed states are matched to existing hypernodes or assigned to new ones, and transitions are recorded as edges. We treat Auto-BFS as an autonomous graph-construction condition rather than a novel graph-search algorithm. Appendix B.2 gives the walk schedule and measured exploration counts, and Appendix B.1 describes how observed states are matched to hypernodes.

In Algorithm 1, π specifies the exploration configuration described in Appendix B.2, including the probability of local expansion, depth schedule, maximum number of successful undo-and-check cycles, and rules for selecting actions. The counters d and k track walk depth and successful undo-and-check cycles, respectively. SelectAction returns an action and a flag b indicating whether it is a local probe. Local-BFS skips actions already explored on the current screen or matching actions explored on sibling screens within the same hypernode. UndoAndCheck returns the observed landing state and a flag v indicating whether return to the source screen was verified after any configured recovery attempts. MatchOrAdd matches the reset observation or adds it to G; graph updates modify $\mathcal{H}$ and E in place.

Algorithm 1 Offline Graph Memory Construction

Require: app A, device D, exploration configuration π, maximum walks N, VLM M_{mem}, embedder ϕ
Ensure: graph $G = (\mathcal{H}, E)$, memory index R

$s_0 \leftarrow \mathsf{Observe}(\mathsf{Reset}(A, D))$
$h_0 \leftarrow \mathsf{Hypernode}(s_0)$
$\mathcal{H} \leftarrow \{h_0\}$; $E \leftarrow \emptyset$; $R \leftarrow \emptyset$
$G \leftarrow (\mathcal{H}, E)$
for $w = 1, \ldots, N$ **do**
  $s \leftarrow \mathsf{Observe}(\mathsf{Reset}(A, D))$
  $h \leftarrow \mathsf{MatchOrAdd}(s, G)$
  $m \leftarrow \mathsf{SampleWalkMode}(\pi)$
  $D_w \leftarrow \mathsf{DepthLimit}(\pi, w)$; $d \leftarrow 0$; $k \leftarrow 0$
  while $d < D_w$ **do**
    $(a, b) \leftarrow \mathsf{SelectAction}(s, h, G, m, d, k, \pi)$
    $(s', \ell') \leftarrow \mathsf{ExecuteObserve}(D, a)$
    $h' \leftarrow \mathsf{MatchHypernode}(s', \ell', G)$
    if $h' = \emptyset$ **then**
      $h' \leftarrow \mathsf{Hypernode}(s', \ell')$
      $\mathcal{H} \leftarrow \mathcal{H} \cup \{h'\}$
    else
      $\mathsf{AddObservation}(h', s', \ell')$
    end if
    $E \leftarrow E \cup \{(h, a, h')\}$
    $\mathsf{MarkExplored}(s, h, a, G)$
    $\mathsf{Checkpoint}(G, s', a)$
    if b **then**
      $(s_{\text{back}}, v) \leftarrow \mathsf{UndoAndCheck}(D, a, s, h, \pi)$
      if $\neg v$ **then**
        break
      end if
      $s \leftarrow s_{\text{back}}$
    else
      $(s, h) \leftarrow (s', h')$
    end if
    Update walk counters (d, k) according to π
  end while
end for
for all $h \in \mathcal{H}, n \in \mathsf{Members}(h)$ **do**
  $P_{h,n} \leftarrow \mathsf{RootPaths}(G, h, n)$
  $(d_{h,n}, I_{h,n}, W_{h,n}, H_{h,n}) \leftarrow M_{\text{mem}}(h, n, P_{h,n})$
  $z^d_{h,n} \leftarrow \phi(d_{h,n})$; $z^I_{h,n} \leftarrow \phi(I_{h,n})$
  $R[(h, n)] \leftarrow$
    $(h, n, d_{h,n}, I_{h,n}, W_{h,n}, H_{h,n}, z^d_{h,n}, z^I_{h,n})$
end for
return G, R

3.3 Graph Construction and Post-Processing

Raw exploration graphs are not used directly as prompts: mobile apps contain dynamic content, ads, pop-ups, scroll variants, and alternate paths that would produce noisy targets and brittle action scripts if copied verbatim. TAG-Nav therefore post-processes the graph into compact retrieval records

that describe what each destination screen is for and how it can be reached.

For each indexed graph state, a vision-language model generates a concise visual page summary and a set of user intents directly satisfied by the current screen. The prompt emphasizes the active interaction layer of the final screenshot, so if a modal, drawer, menu, or permission dialog is in the foreground, the metadata describes that foreground layer rather than the dimmed background.

TAG-Nav also constructs the route field of each destination-memory record by extracting root-to-state paths from the graph, annotating the screenshots along each path with action evidence, and asking a vision-language model to distill the route into semantic waypoints and transition hints. The post-processing prompt removes loops, retries, oscillations, and irrelevant detours, so the resulting memory is expressed as page or state labels and high-level transitions, not fixed coordinates or event-log replay.

Each processed graph state (h, n) becomes a memory record $r_{h,n} = (h, n, d_{h,n}, I_{h,n}, W_{h,n}, H_{h,n}, z^d_{h,n}, z^I_{h,n})$ with the summary, intent, waypoint, hint, and embedding fields defined in Algorithm 1; we write $i = (h, n)$ and $r_i = r_{h,n}$ for compact figure notation. Figure 2 contrasts the destination-record unit with the per-element hints, intent-trajectory pairs, and page guidelines used by prior graph-RAG GUI systems (Wen et al., 2024; Guan et al., 2025; Xie et al., 2025; Chen et al., 2025b).

3.4 Target Retrieval and Memory Formatting

Given a user instruction q, TAG-Nav first retrieves candidate destination records. The instruction is embedded and compared against both the visual-summary embeddings and direct-intent embeddings for all records. Top matches from the two fields are combined so that a candidate may be retrieved because it looks like the requested page, because it directly supports the requested intent, or both.

The candidate set is then passed to a compact target-selection prompt. The selector receives the user instruction, candidate screen summaries, and direct-intent text, then chooses the graph state whose current screen best satisfies the instruction as the final destination. The selector prefers loaded destinations over loading or transition screens, the destination page over foreground modals unless requested, routes page-style queries to the visual summary and action-style queries to the intents, rejects link or menu entries that merely point at the target, and prefers specific destinations for specific queries and general help pages for generic queries (full prompts in Appendix C). This target-first retrieval step avoids a common graph-memory failure mode in which a retrieved state is semantically close to the task but is not itself the destination page; unlike a direct retrieval of a past task trace, the destination is selected first and its memory is only then formatted for injection.

TAG-Nav then formats a destination-memory block containing the original task, a final-goal description, the relevant waypoint sequence, and transition hints. The block is intentionally semantic and route-oriented: it orients the backbone toward the correct app region while leaving grounding, action selection, and local recovery to the backbone, and it never prescribes fixed coordinates, replays exploration verbatim, or asks the actor to perform the irreversible final task action.

3.5 Agent-Agnostic Online Guidance

Algorithm 2 formalizes the online actor loop: TAG-Nav launches the app, captures the current screenshot, calls the GUI backbone with the user instruction plus the retrieved destination-memory block, and executes the predicted action until the backbone emits a terminal `finished` action or the step budget is reached.

The memory interface itself is backbone-independent: the destination-memory block is plain text, and any GUI agent that consumes a task prompt and screen observation and emits mobile actions can be wrapped with the same retrieved memory without retraining. The canonical backbone in our experiments is UI-TARS-1.5-7B; Sec. 5 pairs TAG-Nav memory with different GUI actors to test this agent-agnostic property empirically. Dynamic variants that refresh or edit memory during execution are treated as controlled ablations in Appendix H.7.

4 DestBench

End-to-end task benchmarks conflate navigation with grounding, manipulation, and finalization, so an aggregate success rate cannot attribute a failure to memory.

4.1 Page-Reaching Navigation

Each DestBench instance is a tuple $(A, q, s_0, x_{\text{target}})$ where A is the target app

Algorithm 2 Online Graph-Memory Navigation

Require: instruction q, memory index R, embedder ϕ, backbone L, device D, budget T
Ensure: executed trajectory τ, terminal status y

$C_{\text{sum}} \leftarrow \mathsf{TopK}_{\text{summary}}(\phi(q), R)$
$C_{\text{int}} \leftarrow \mathsf{TopK}_{\text{intent}}(\phi(q), R)$
$C \leftarrow C_{\text{sum}} \cup C_{\text{int}}$
$r^* \leftarrow \mathsf{SelectDestination}(q, C)$
$m \leftarrow \mathsf{FormatMemory}(q, r^*)$
$\tau \leftarrow \emptyset$
for $t = 1, \ldots, T$ **do**
  $x_t \leftarrow \mathsf{Screenshot}(D)$
  $a_t \leftarrow L(q, x_t, m)$
  $\tau \leftarrow \mathsf{Append}(\tau, (x_t, a_t))$
  if $a_t =$ FINISHED **then**
    $y \leftarrow$ FINISHED
    return τ, y
  end if
  $\mathsf{Execute}(D, a_t)$
end for
$y \leftarrow$ STEPBUDGETREACHED
return τ, y

and version, q is a natural-language instruction, s_0 is a reset starting state, and x_{target} is a ground-truth screenshot of the pre-action destination screen. The agent emits a trajectory $\tau = (x_0, a_0, x_1, a_1, \ldots, x_T)$ that terminates when it issues a FINISHED action or hits the step budget. Success is judged by visual equivalence between x_T and x_{target} rather than by completion of any downstream operation: a task like "set an alarm for 7am" is reduced to "reach the new-alarm creation screen"; the time picker itself is out of scope. This is a deliberate measurement choice rather than a simplification of mobile use: the post-arrival action is where benchmarks introduce confounds (form-filling, sensitive state changes, tool or API use) that have little to do with app-specific navigation. Accordingly, DestBench scores page reaching only; its gains should not be interpreted as gains in end-to-end task completion.

4.2 Apps, Tasks, and Targets

DestBench contains tasks from 20 Android apps spanning Shopping, Productivity, Utility, Travel, Entertainment, Social Media, Food & Drink, Communication, and Local Services categories (full inventory in Appendix A.1). The current controlled graph-memory experiments use the 6-app core cohort for which target annotations and graph conditions are available. Tasks are derived from real user trajectories collected in our benchmark substrate, with each task annotated by the pre-action destination screen from its original trajectory (annotation protocol in Appendix A.3); Table 2 summarizes per-app task counts.

App	Tasks	Records	Hypernodes	Edges	Covered
Google Maps	24	415	364	1,192	20
Amazon	17	287	257	652	8
YouTube	15	229	196	483	9
Settings	9	94	84	128	7
Clock	9	93	91	322	6
Calendar	10	50	40	182	5
Total	84	1,168	1,032	2,959	55

Table 2: Per-app task distribution and core graphs in DestBench. The controlled graph-memory experiments use the six-app core cohort shown here. Graph columns describe the Auto-BFS graphs (one record per stored screen state; directed hypernode edges); Covered counts tasks whose strict target is in the graph (human-audited).

4.3 Graph-Aware Diagnostics

DestBench records per-task structural metadata that attributes failures to specific causes rather than only reporting aggregate success: *target coverage* (whether the graph contains a node visually equivalent to x_{target}), *retrieval correctness* (whether top-1 retrieval picks the correct target), *path existence* from s_0 to the retrieved target, *path cleanliness* (route length in waypoints, detours, loops, backtracks, restarts, and transition-hint count), and *judge match score* (raw 1–5 rating). A target is covered only when a record matches the pre-action destination; a parent page or a screen one action too early or late is not covered. These let us distinguish between error types, including situations when the graph missed the page, when retrieval picked the wrong target, and when the route itself was noisy.

4.4 Strict Judge and Calibration

Primary navigation results use human judgments. Strict success requires the reference destination, allowing cosmetic differences on the same functional page; near misses fail. GPT-4o (Hurst et al., 2024) cross-checks final and reference screenshots on a 1–5 scale (4–5 indicate success); on a 530-pair calibration set it agrees with human judgments on 97.5% of binary labels (Appendix A.5).

5 Experiments

5.1 Setup

Backbones and protocols. We wrap four visual GUI actors without fine-tuning: UI-TARS-1.5-7B (UI-TARS; Qin et al., 2025) and Qwen-VL-Max (Qwen; Bai et al., 2025)[2] as headline backbones, plus AgentCPM-GUI (Zhang et al., 2025c) (8B open-weight MiniCPM-V-based) and

[2]Queried through the DashScope API model alias `qwen-vl-max-latest` (April–May 2026).

MAI-UI (Zhou et al., 2025) as cross-family checks (Table 7). Benchmarks used in our experiments are DestBench (Sec. 4), DroidTask (Wen et al., 2024), and SPA-Bench (Chen et al., 2025a). For DroidTask/SPA-Bench we redefine success as reaching the pre-action destination (not the final action), reusing the benchmark-native judge (Appendix E) and reporting final numbers using human-annotated results. Only within-column `no-memory` → `+TAG-Nav` deltas are compared. Table 2 gives per-app graph sizes and target coverage (exploration effort: Appendix B.2). Online navigation uses the canonical configuration: dual-field retrieval, top-1 selection, waypoint memory (Appendix B.5).

5.2 Main Result: Cross-Backbone, Cross-Benchmark

Table 3 reports the headline result: TAG-Nav improves over no-memory in every aggregate cell. These are navigation gains, not end-to-end task-completion gains. Graph coverage is partial (Table 2), and a backbone can still reach an uncovered destination from its own app knowledge, so successes are not bounded by coverage. The aggregate masks per-app heterogeneity (Figure 3): apps whose residual difficulty is execution rather than navigation (Calendar, Settings) show zero or negative per-cell deltas, so TAG-Nav is a navigation-memory aid rather than a uniform improvement; see §7. Settings already has a strong no-memory baseline, and Calendar failures often occur at terminal date/time manipulation after reaching the relevant page. Tables 3 and 4 report paired tests (per-app uncertainty: Appendix D.2); Table 5 gives per-app DroidTask-Nav and SPA-Bench-Nav results.

Backbone	DestBench SR (%)	DroidTask-Nav SR (%)	SPA-Bench-Nav SR (%)
UI-TARS-1.5-7B	52.4 → **58.3**	63.3 → **71.3**	66.7 → **76.7**
p	0.383	0.073	0.508
Qwen-VL-Max	59.5 → **71.4**	66.0 → **70.7**[§]	76.7 → **93.3**
p	0.076	0.265	0.125
Avg. gain	+8.9	+6.4	+13.3

Table 3: Cross-backbone, cross-benchmark navigation. Cells report `no-memory` → `+TAG-Nav` in success rate (SR). DroidTask and SPA-Bench are converted to page-reaching tasks; only within-column deltas are comparable. All cells use human-annotated strict success. p: exact McNemar test; 95% confidence intervals (CIs) are in Appendix D.2. [§]The no-memory arm resized screenshots to a 1,280-pixel longest side; the +TAG-Nav arm used full resolution.

5.3 Graph Quality: Controlled Decomposition

Table 6 decomposes the automated-to-manual gap on Qwen. Row (d) keeps the automated waypoint pipeline but replaces the retrieved target with the manual destination (isolating retrieval); it does not improve over TAG-Nav (70.2 vs. 71.4); per-app drilldown in Appendix D.4. Row (f) (manual exploration) sits *below* Auto-BFS (66.7 vs. 71.4) because manual graphs trade coverage for cleanliness; it is a "clean routes, narrow coverage" condition rather than an upper bound. In a separate observational analysis of 313 archived memory-augmented runs judged by GPT-4o, success falls from 58% for routes with 1–2 waypoints to 50% for 3–4 and 38% for 5 or more (correlation analysis in Appendix D.5). Appendix D.9 varies which record fields are injected.

Graph	Success	Gain (pp)	95% CI (pp)	p
UI-TARS-1.5-7B				
No memory	44/84	–	–	–
Random	54/84	+11.9	[0.0, 23.8]	0.087
Auto-BFS	49/84	+6.0	[−4.8, 16.7]	0.383
Manual	59/84	+17.9	[8.3, 27.4]	0.0007
Qwen-VL-Max				
No memory	50/84	–	–	–
Random	54/84	+4.8	[−7.1, 16.7]	0.572
Auto-BFS	60/84	+11.9	[0.0, 23.8]	0.076
Manual	56/84	+7.1	[−4.8, 19.0]	0.345

Table 4: Paired graph-condition comparisons on DestBench. Success is the human-judged destination match count. Gains and 95% CIs are in percentage points (pp) relative to no memory for the same backbone; p: exact McNemar test (Appendix D.2). Some random- and manual-arm cells used the same app's memory from a different graph condition (Appendix D.3); without those cells, every gain remains positive.

	No memory		+ TAG-Nav	
App	UI-TARS-1.5-7B	Qwen-VL-Max	UI-TARS-1.5-7B	Qwen-VL-Max
DroidTask-Nav (12 apps, 150 tasks)				
AppLauncher	2/5	4/5	5/5	2/5
Calendar	11/17	9/17	12/17	12/17
Camera	5/15	5/15	4/15	5/15
Clock	7/12	9/12	11/12	11/12
Contacts	12/14	14/14	13/14	13/14
Dialer	7/15	12/15	6/15	12/15
FileManager	6/16	10/16	14/16	13/16
Gallery	8/9	8/9	9/9	8/9
Messenger	13/15	8/15	11/15	11/15
MusicPlayer	7/9	7/9	7/9	7/9
Notes	9/14	5/14	6/14	4/14
VoiceRecorder	8/9	8/9	9/9	8/9
Total	95/150	99/150	107/150	106/150
SR (%)	63.3	66.0	71.3	70.7
SPA-Bench-Nav (7 apps, 30 tasks)				
Airbnb	1/3	0/3	1/3	2/3
Amazon	2/3	3/3	3/3	3/3
Calendar	3/3	3/3	2/3	3/3
Chrome	2/3	3/3	3/3	3/3
Clock	6/6	3/6	6/6	5/6
Google Maps	3/6	5/6	5/6	6/6
YouTube	3/6	6/6	3/6	6/6
Total	20/30	23/30	23/30	28/30
SR (%)	66.7	76.7	76.7	93.3

Table 5: DroidTask-Nav and SPA-Bench-Nav per-app drilldowns. Tasks converted from task-completion to navigation-only goals. Human-annotated strict destination success for both benchmarks.

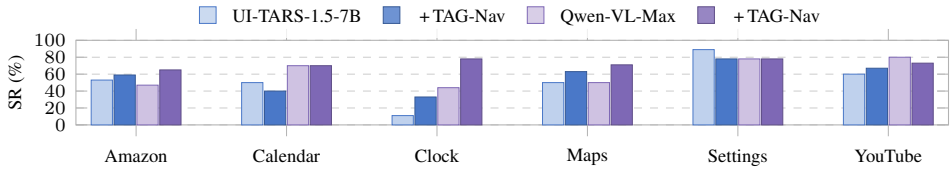


Figure 3: Per-app navigation success rate with and without TAG-Nav. Gains are largest when navigation is the bottleneck (Clock, Amazon, Google Maps) and smaller when the baseline is already strong or execution dominates (Settings, Calendar).

Condition	Targets	Routes	SR (%)
(a) No memory	–	–	59.5
(b) Random exploration	auto	noisy	64.3
(c) TAG-Nav (Auto-BFS)	auto	auto	**71.4**
(d) Manual targets + auto routes	manual	auto	70.2
(e) Auto targets + manual routes	auto	manual	**79.8**
(f) Manual exploration	manual	manual	66.7

Table 6: Graph-quality controlled decomposition on DestBench (Qwen-VL-Max). Human-annotated strict success on the 84-task core. Row (d) ran at full input resolution on a separate host. Rows (d) and (e) replace one memory field at a time with manual ground truth. Rows (b) and (f) include app cells that used the same app's memory from a different graph condition (Appendix D.3). The UI-TARS counterpart is in Appendix D.4.

5.4 Comparison to Prior Memory Systems

Table 7 compares TAG-Nav against prior memory-augmented GUI agents as a memory-format comparison on our task set and human-annotated judge, not as a claim of outperforming PG-Agent or GUI-Explorer in their released forms. PG-Agent, which needs human demonstrations, trails no-memory UI-TARS, and GUI-Explorer (GPT-4o actor) falls between UI-TARS without and with TAG-Nav, while TAG-Nav adds unsupervised exploration plus one retrieval per task and improves every wrapped backbone (adaptation rules and actors in App. E). These rows isolate the memory abstraction (custom validators and planners omitted) and do not reflect each system's published end-to-end performance.

System	Success	SR (%)
Prior memory-format adaptations		
PG-Agent†	36/84	42.9
GUI-Explorer	47/84	56.0
Backbone agents without / with TAG-Nav		
AgentCPM-GUI	28/84	33.3
+ TAG-Nav	33/84	39.3 (+6.0)
UI-TARS-1.5-7B	44/84	52.4
+ TAG-Nav	49/84	58.3 (+5.9)
MAI-UI	43/78	55.1
+ TAG-Nav	47/79	59.5 (+4.4)
Qwen-VL-Max	50/84	59.5
+ TAG-Nav	60/84	71.4 (+11.9)

Table 7: Prior memory formats and wrapped backbones on DestBench. TAG-Nav adds one-time unsupervised exploration and one retrieval call per task. All rows use human-annotated strict success. †PG-Agent additionally requires human-labeled task demonstrations.

App	No memory	+ TAG-Nav
Airbnb	7/13	10/13
Chrome	4/13	6/13
eBay	11/16	13/16
Instagram	4/13	2/13
TikTok	2/13	4/13
Total	28/68 (41.2%)	35/68 (51.5%)

Table 8: Transfer to five additional apps. UI-TARS-1.5-7B, human-judged strict destination match; separate from the 84-task core.

5.5 Transfer to Additional Apps

To test whether the benefit extends beyond the six core apps, we ran UI-TARS-1.5-7B with and without Auto-BFS memory on 68 tasks from five further apps, labeled with the same human strict rule. Memory raises success from 28/68 to 35/68 (+10.3 pp; 11 tasks improved, 4 regressed; exact McNemar $p = 0.12$), with gains on four of the five apps and a loss on Instagram (Table 8). The direction matches the core results, though the gain is not significant in this small cohort.

5.6 Failure Taxonomy

Of 48 labeled open-loop TAG-Nav failures, 22 are wrong retrievals (A), 15 near misses (B), and 11 execution failures (C); A and B together account for $\sim$77% and are addressable to the memory pipeline, so destination retrieval and target selection remain fragile (per-class notes in §7 and App. F.1).

6 Conclusion

TAG-Nav introduces a destination-memory module that wraps existing visual GUI agents without retraining, converting task-agnostic offline exploration into compact target and route records injected at inference time. Across two backbones and three benchmarks, this memory improves aggregate navigation success, and graph-quality controls associate success more with route cleanliness than with graph size. This makes app-specific memory a measurable destination map rather than a larger exploration trace.

7 Limitations

Current evidence is strongest for Android app-navigation tasks and a Qwen-VL-Max backbone. Auto-BFS graph memory still trails clean/manual memory in some conditions. Graph memory depends on app version, account/login state, UI drift, and exploration coverage. Apps with sufficiently large navigation-affecting UI updates require re-exploration before previously collected destination memory can be trusted; in a small two-app probe, full-record memory gave no benefit over no memory before or after updates (Table 26). TAG-Nav is also not plug-and-play with arbitrary task-completion benchmarks because it requires page-reaching targets, safe stopping points, and graph-aware diagnostics. Manual/clean graph results should be interpreted as a control rather than the deployment assumption. Comparisons to external benchmarks require careful protocol matching. TAG-Nav is most useful when the destination is graph-covered, the route is short and clean, and the backbone can reliably execute visible actions; coverage gaps, app-state drift, and sibling-page retrieval errors are the dominant remaining failure modes. The three behavioral failure classes documented in Table 24 (§5.6) correspond to these limits: *wrong retrieval* (Class A) reflects coverage and embedding limits, *near miss* (Class B) reflects last-step disambiguation limits that sharper target selection and cleaner routes may address, and *execution failure* (Class C) reflects step-budget exhaustion and open-loop route-following limits that per-step memory refresh may address.

Per-app heterogeneity. Gains are not uniform across apps (Figure 3): Calendar and Settings, where the residual difficulty is execution rather than navigation, show smaller or negative deltas (§5.2). We do not claim TAG-Nav improves every (app, backbone) cell.

8 Ethical Considerations

Authorized test setting and privacy. TAG-Nav exercises third-party mobile applications, so all exploration and evaluation used dedicated test devices, disposable accounts, and locally collected test data, not personal user accounts. DestBench tasks are derived from existing benchmark trajectories and our disposable-account collection; released artifacts use scrubbed or generic targets, and we do not distribute user-identifiable content. Screenshots captured during exploration may incidentally contain account-specific content; explored graphs are intended for the same app version and account state in which they were collected, and we recommend that downstream users re-collect graphs under their own accounts rather than reusing ours when account state matters. Downstream use should respect app-level terms of service and obtain authorization where required.

Dual use and release. Autonomous mobile GUI agents can in principle be repurposed for spam, automated account creation, or unauthorized actions on user accounts. TAG-Nav is a navigation-memory module for existing backbones rather than a new execution capability: it improves where the agent goes, not what novel actions it can perform. DestBench is intentionally scoped to page reaching and excludes the irreversible final actions most relevant to abuse scenarios such as purchasing, sending, deleting, or changing sensitive settings. All backbone models are used under their respective open-weight or API licenses. We plan to release code, exploration scripts, and the evaluation protocol under terms limiting use to authorized accounts and test environments; app graphs and trajectories will be shared only where the underlying screenshots can be released.

Compute and environmental impact. Offline exploration and online evaluation run on commodity Android emulators; open-weight backbones run on a local GPU host, while Qwen models and GPT-4o are queried through the DashScope and OpenAI APIs. We do not train a new GUI backbone in this work, so compute usage is modest relative to model-training contributions, and the offline exploration cost amortizes across all later navigation tasks for the same app version. At inference time, TAG-Nav adds one retrieval call per task to the backbone's own calls; graph-construction counters are reported in Appendix D.8.

Data minimization and annotator labor. DestBench task descriptions and final-screenshot targets were annotated by the authors. Strict-judge calibration used the same pool to label 530 (task, final-screenshot) pairs, with labelers shown only the two screens and the user instruction.

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*.

Yuxiang Chai, Shunye Tang, Han Xiao, Weifeng Lin, Hanhao Li, Jiayu Zhang, Liang Liu, Pengxiang Zhao, Guangyi Liu, Guozhi Wang, Shuai Ren, Rongduo Han, Haining Zhang, Siyuan Huang, and Hongsheng Li. 2026. A3: Android agent arena for mobile GUI agents with essential-state procedural evaluation. In *Findings of the Association for Computational Linguistics: ACL 2026*, pages 3774–3789, San Diego, California, United States. Association for Computational Linguistics.

Jingxuan Chen, Derek Yuen, Bin Xie, Yuhao Yang, Gongwei Chen, Zhihao Wu, Li Yixing, Xurui Zhou, Weiwen Liu, Shuai Wang, Kaiwen Zhou, Rui Shao, Liqiang Nie, Yasheng Wang, Jianye Hao, Jun Wang, and Kun Shao. 2025a. SPA-Bench: A comprehensive benchmark for SmartPhone agent evaluation. In *Proceedings of the International Conference on Learning Representations*.

Weizhi Chen, Ziwei Wang, Leyang Yang, Sheng Zhou, Xiaoxuan Tang, Jiajun Bu, Yong Li, and Wei Jiang. 2025b. PG-Agent: An agent powered by page graph. In *Proceedings of the 33rd ACM International Conference on Multimedia*, pages 6878–6887, New York, NY, USA. Association for Computing Machinery.

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

Shihan Deng, Weikai Xu, Hongda Sun, Wei Liu, Tao Tan, Jianfeng Liu, Ang Li, Jian Luan, Bin Wang, Rui Yan, and Shuo Shang. 2024. Mobile-Bench: An evaluation benchmark for LLM-based mobile agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 8813–8831, Bangkok, Thailand. Association for Computational Linguistics.

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 *Proceedings of the International Conference on Learning Representations*.

Ziyi Guan, Jason Chun Lok Li, Zhijian Hou, Pingping Zhang, Donglai Xu, Yuzhi Zhao, Mengyang Wu, Jinpeng Chen, Thanh-Toan Nguyen, Pengfei Xian, Wenao Ma, Shengchao Qin, Graziano Chesi, and Ngai Wong. 2025. KG-RAG: Enhancing GUI agent decision-making via knowledge graph-driven retrieval-augmented generation. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 5396–5405, Suzhou, China. Association for Computational Linguistics.

Wenyi Hong, Weihan Wang, Qingsong Lv, Jiazheng Xu, Wenmeng Yu, Junhui Ji, Yan Wang, Zihan Wang, Yuxiao Dong, Ming Ding, and Jie Tang. 2024. CogAgent: A visual language model for GUI agents. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14281–14290.

Aaron Hurst, Adam Lerer, Adam P. Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, Aleksander Mądry, Alex Baker-Whitcomb, Alex Beutel, Alex Borzunov, Alex Carney, Alex Chow, Alex Kirillov, Alex Nichol, Alex Paino, and 398 others. 2024. GPT-4o system card. *arXiv preprint arXiv:2410.21276*.

Youngmin Im, Byeongung Jo, Jaeyoung Wi, Seungwoo Baek, Tae Hoon Min, Joo Hyung Lee, Sangeun Oh, Insik Shin, and Sunjae Lee. 2026. MobiBench: Multi-branch, modular benchmark for mobile GUI agents. *arXiv preprint arXiv:2512.12634v3*.

Quyu Kong, Xu Zhang, Zhenyu Yang, Nolan Gao, Chen Liu, Panrong Tong, Chenglin Cai, Hanzhang Zhou, Jianan Zhang, Liangyu Chen, Zhidan Liu, Steven Hoi, and Yue Wang. 2026. MobileWorld: Benchmarking autonomous mobile agents in agent-user interactive and MCP-augmented environments. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 6142–6167, San Diego, California, United States. Association for Computational Linguistics.

Wei Li, William Bishop, Alice Li, Chris Rawles, Folawiyo Campbell-Ajala, Divya Tyamagundlu, and Oriana Riva. 2024. On the effects of data scale on UI control agents. In *Advances in Neural Information Processing Systems*, volume 37, pages 92130–92154. Curran Associates, Inc.

Yanda Li, Chi Zhang, Wenjia Jiang, Wanqi Yang, Bin Fu, Pei Cheng, Xin Chen, Ling Chen, and Yunchao Wei. 2025. AppAgent v2: Advanced agent for flexible mobile interactions. *arXiv preprint arXiv:2408.11824*.

Yuanchun Li, Ziyue Yang, Yao Guo, and Xiangqun Chen. 2017. DroidBot: a lightweight UI-guided test input generator for Android. In *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*, pages 23–26.

Quanfeng Lu, Wenqi Shao, Zitao Liu, Lingxiao Du, Fanqing Meng, Boxuan Li, Botong Chen, Siyuan Huang, Kaipeng Zhang, and Ping Luo. 2025. GUIOdyssey: A comprehensive dataset for cross-app GUI navigation on mobile devices. In *Proceedings of the*

IEEE/CVF International Conference on Computer Vision, pages 22404–22414.

Yadong Lu, Jianwei Yang, Yelong Shen, and Ahmed Awadallah. 2024. OmniParser for pure vision based GUI agent. *arXiv preprint arXiv:2408.00203*.

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*.

Qwen Team. 2026a. Qwen3.5: Towards native multimodal agents. Qwen blog post, `https://qwen.ai/blog?id=qwen3.5`. Accessed: 2026-09-30.

Qwen Team. 2026b. Qwen3.6-Plus: Towards real world agents. Qwen blog post, `https://qwen.ai/blog?id=qwen3.6`. Accessed: 2026-09-30.

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

Christopher Rawles, Sarah Clinckemaillie, Yifan Chang, Jonathan Waltz, Gabrielle Lau, Marybeth Fair, Alice Li, William E. Bishop, Wei Li, Folawiyo Campbell-Ajala, Daniel Kenji Toyama, Robert James Berry, Divya Tyamagundlu, Timothy P. Lillicrap, and Oriana Riva. 2025. AndroidWorld: A dynamic benchmarking environment for autonomous agents. In *Proceedings of the International Conference on Learning Representations*.

Christopher Rawles, Alice Li, Daniel Rodriguez, Oriana Riva, and Timothy Lillicrap. 2023. Android in the Wild: A large-scale dataset for Android device control. In *Advances in Neural Information Processing Systems*, volume 36, pages 59708–59728. Curran Associates, Inc.

Yibo Shi, Jungang Li, Linghao Zhang, Zihao Dongfang, Biao Wu, Sicheng Tao, Yibo Yan, Chenxi Qin, Weiting Liu, Zhixin Lin, Hanqian Li, Yu Huang, Song Dai, Yonghua Hei, Yue Ding, Xiang Li, Shikang Wang, Chengdong Xu, Jingqi Liu, and 9 others. 2026. AndroTMem: From interaction trajectories to anchored memory in long-horizon GUI agents. *arXiv preprint arXiv:2603.18429*.

Yuchen Sun, Shanhui Zhao, Tao Yu, Hao Wen, Samith Va, Mengwei Xu, Yuanchun Li, and Chongyang Zhang. 2025. GUI-Xplore: Empowering generalizable GUI agents with one exploration. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 19477–19486.

Haoming Wang, Haoyang Zou, Huatong Song, Jiazhan Feng, Junjie Fang, Junting Lu, Longxiang Liu, Qinyu Luo, Shihao Liang, Shijue Huang, Wanjun Zhong, Yining Ye, Yujia Qin, Yuwen Xiong, Yuxin Song, Zhiyong Wu, Aoyan Li, Bo Li, Chen Dun, and 93 others. 2025a. UI-TARS-2 technical report: Advancing GUI agent with multi-turn reinforcement learning. *arXiv preprint arXiv:2509.02544*.

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

Zhenhailong Wang, Haiyang Xu, Junyang Wang, Xi Zhang, Ming Yan, Ji Zhang, Fei Huang, and Heng Ji. 2025b. Mobile-Agent-E: Self-evolving mobile assistant for complex tasks. In *NeurIPS 2025 Workshop on Scaling Environments for Agents*.

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, New York, NY, USA. Association for Computing Machinery.

Bin Xie, Rui Shao, Gongwei Chen, Kaiwen Zhou, Yinchuan Li, Jie Liu, Min Zhang, and Liqiang Nie. 2025. GUI-explorer: Autonomous exploration and mining of transition-aware knowledge for GUI agent. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 5650–5667, Vienna, Austria. Association for Computational Linguistics.

Weikai Xu, Zhizheng Jiang, Yuxuan Liu, Pengzhi Gao, Wei Liu, Jian Luan, Yuanchun Li, Yunxin Liu, Bin Wang, and Bo An. 2025a. Mobile-Bench-v2: A more realistic and comprehensive benchmark for VLM-based mobile agents. *arXiv preprint arXiv:2505.11891*.

Yifan Xu, Xiao Liu, Xueqiao Sun, Siyi Cheng, Hao Yu, Hanyu Lai, Shudan Zhang, Dan Zhang, Jie Tang, and Yuxiao Dong. 2025b. AndroidLab: Training and systematic benchmarking of Android autonomous agents. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2144–2166, Vienna, Austria. Association for Computational Linguistics.

Mingxian Yu, Siqi Luo, and Xu Chen. 2026. GraphPilot: GUI task automation with one-step LLM reasoning powered by knowledge graph. *Journal of Intelligent Computing and Networking*, 2(1):13–21.

Chi Zhang, Zhao Yang, Jiaxuan Liu, Yanda Li, Yucheng Han, Xin Chen, Zebiao Huang, Bin Fu, and Gang Yu. 2025a. AppAgent: Multimodal agents as smartphone

users. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems*, pages 1–20, New York, NY, USA. Association for Computing Machinery.

Yanzhao Zhang, Mingxin Li, Dingkun Long, Xin Zhang, Huan Lin, Baosong Yang, Pengjun Xie, An Yang, Dayiheng Liu, Junyang Lin, Fei Huang, and Jingren Zhou. 2025b. Qwen3 Embedding: Advancing text embedding and reranking through foundation models. *arXiv preprint arXiv:2506.05176*.

Zhong Zhang, Yaxi Lu, Yikun Fu, Yupeng Huo, Shenzhi Yang, Yesai Wu, Han Si, Xin Cong, Haotian Chen, Yankai Lin, Jie Xie, Wei Zhou, Wang Xu, Yuanheng Zhang, Zhou Su, Zhongwu Zhai, Xiaoming Liu, Yudong Mei, Jianming Xu, and 6 others. 2025c. AgentCPM-GUI: Building mobile-use agents with reinforcement fine-tuning. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 155–180, Suzhou, China. Association for Computational Linguistics.

Hanzhang Zhou, Xu Zhang, Panrong Tong, Jianan Zhang, Liangyu Chen, Quyu Kong, Chenglin Cai, Chen Liu, Yue Wang, Jingren Zhou, and Steven Hoi. 2025. MAI-UI technical report: Real-world centric foundation GUI agents. *arXiv preprint arXiv:2512.22047*.

Sibo Zhu, Wenyi Wu, Kun Zhou, Stephen Wang, and Biwei Huang. 2026. Hybrid self-evolving structured memory for computer-use agents. In *Findings of the Association for Computational Linguistics: ACL 2026*, pages 11287–11304, San Diego, California, United States. Association for Computational Linguistics.

Appendix

Table of Contents

	Section	Page
A	**DestBench Construction**	15
	A.1 App selection rationale and full app list	15
	A.2 Task-collection pipeline	15
	A.3 Destination annotation protocol	15
	A.4 Representative tasks	15
	A.5 GPT-4o judge calibration	15
	A.6 DestBench app counts	15
B	**TAG-Nav Implementation Details**	15
	B.1 Hypernode matching	15
	B.2 Local-BFS walk policy	15
	B.3 Memory record schema	16
	B.4 Backbone configurations	16
	B.5 Protocol details	16
C	**Prompts**	16
	C.1 Metadata extraction prompt (page summary, direct intent, route distillation)	16
	C.2 Target selector prompt	17
	C.3 Destination-memory injection template	18
	C.4 GPT-4o judge prompt	18
D	**Additional Quantitative Results**	18
	D.1 Per-app, per-backbone, per-benchmark breakdown	18
	D.2 Per-app uncertainty	19
	D.3 Graph-arm memory provenance	19
	D.4 Per-app graph-quality breakdown	20
	D.5 Route cleanliness	20
	D.6 Memory format and retrieval-depth ablation	20
	D.7 Cross-family and prior-memory outcomes	20
	D.8 Cost analysis	20
	D.9 Record-content ablation	20
	D.10 Informed no-memory baseline	20
E	**Format-Adapted Reimplementation Protocol**	20
	E.1 Per-system adaptation rules	20
F	**Qualitative Examples**	20
	F.1 Failure trajectories per class	20
	F.2 Success trajectories	20
	F.3 Raw destination-record examples	20
G	**Detailed Related-Work Positioning**	20
H	**Limitations, Robustness, and Ethics**	20
	H.1 Dynamic-screen handling	20
	H.2 Cross-app-version robustness	21
	H.3 Multi-path and ambiguous-target handling	22
	H.4 Privacy and sensitive-screen handling	22
	H.5 Use of AI in paper preparation	23
	H.6 Models used in the TAG-Nav pipeline	23
	H.7 Dynamic-memory variants	23

A DestBench Construction

This section documents the construction of DestBench, from app selection through annotation through automated judging.

A.1 App selection rationale and full app list

We selected 20 Android apps spanning nine product categories, with a core cohort of 6 apps used for the controlled graph-condition study (no-memory, random, Auto-BFS, clean manual). Apps were chosen to span (i) shallow vs deep navigation depth, (ii) static vs dynamic content, (iii) presence vs absence of account-gated screens, and (iv) presence of modals, drawers, and bottom-sheet UI patterns. Table 9 lists all apps with their category and core-cohort membership.

App	Category	Core 6
Amazon	Shopping	•
Calendar	Productivity	•
Clock	Utility	•
Google Maps	Travel	•
Settings	Utility	•
YouTube	Entertainment	•
Airbnb	Travel	
Chrome	Productivity	
eBay	Shopping	
Instagram	Social Media	
LinkedIn	Social Media	
Spotify	Entertainment	
Starbucks	Food & Drink	
Target	Shopping	
TikTok	Social Media	
Uber	Travel	
Uber Eats	Food & Drink	
Walmart	Shopping	
WhatsApp	Communication	
Yelp	Local Services	

Table 9: Full DestBench app inventory. 20 Android apps spanning Shopping, Productivity, Utility, Travel, Entertainment, Social Media, Food & Drink, Communication, and Local Services categories. The 6-app core cohort is the subset used for the controlled graph-condition study (no-memory, random, automated Local-BFS, clean manual).

A.2 Task-collection pipeline

Tasks are derived from real user-trajectory recordings collected in our benchmark substrate. Each recording is segmented at the pre-action destination boundary (the last screen before a task-specific irreversible operation, defined in §4) and labeled with a natural-language instruction by paraphrasing the original goal. Instructions are filtered to remove user-identifying content and to disambiguate destinations that admit multiple visually-distinct correct pages.

A.3 Destination annotation protocol

Each task instance is annotated with a target screenshot x_{target} that visually identifies the pre-action destination, plus a dynamic-state tolerance mask that lists which UI regions (e.g., timestamps, scroll positions, advertisement slots) are ignored by the judge. The primary curator annotated the full core cohort; a second author audited a random 20% sample of the annotated targets, and the small number of disagreements (predominantly cases where multiple visually-distinct pages plausibly satisfied the instruction) were resolved jointly against the original trajectory recording. We report this rather than a full second-pass inter-annotator agreement because the dominant source of annotation noise in our pipeline is target ambiguity, not labeler disagreement on unambiguous cases, and the strict-judge calibration in App. A.5 provides a downstream check on residual labeling noise. The primary metric requires the reference destination; a near miss is a failure.

A.4 Representative tasks

Table 10 gives six representative tasks across six apps, showing the instruction, target page, and the stopping boundary that excludes downstream execution from scoring.

A.5 GPT-4o judge calibration

The judge prompt (Appendix C.4) is calibrated against a single-curator gold set of 530 (task, final-screenshot) pairs. Table 11 reports binary agreement, false-pass/false-fail breakdown, and ordinal κ. The residual 13 disagreements are primarily false-pass, and concentrate on near-miss cases (sibling pages, stopping one page early). The ordinal Cohen's κ of 0.95 indicates substantial agreement beyond chance.

Metric	Value
Total calibration tasks	530
Binary agreement	517 / 530 (97.5%)
False pass	10
False fail	3
Cohen's κ (binary)	0.95

Table 11: GPT-4o judge calibration against human annotators. Calibration set: 530 (task, agent-final-screenshot) pairs spanning all 20 apps. Binary agreement represents matching judge and human results. False pass represents a reported success from the judge, and failure from the human annotator, while false fail represents a reported failure from the judge, and success from the human annotator. Binary success label uses scores 4–5 = success.

A.6 DestBench app counts

Table 2 (rendered inline in §4) shows per-app task counts in the core cohort. The 6-app core cohort spans Google Maps, Amazon, YouTube, Settings, Clock, and Calendar, totaling 84 tasks.

B TAG-Nav Implementation Details

B.1 Hypernode matching

Hypernode matching compares a new screen against each stored member screen of a hypernode on two signals: visual similarity, as the cosine similarity of CLIP ViT-B/32 screenshot embeddings (Radford et al., 2021), and layout similarity of the Android XML view hierarchy, using a normalized tree-edit distance and `text-embedding-v4` embeddings of the page layout. A screen joins the hypernode when both signals are within thresholds tuned during graph collection.

B.2 Local-BFS walk policy

Exploration runs up to 1,000 walks from the app root, each with at most five forward steps. With probability 0.7, a walk expands locally: it tries untested actions on the current screen and verifies each return, for up to 15 successful undo-and-check cycles; otherwise it explores randomly. Actions already tried on the current screen or its matched siblings are skipped, and the rest are ordered by type (image elements first, then scroll, swipe, and card actions, then the rest), with random tie-breaking. Table 20 reports the resulting walk and action counts per app.

App	User instruction q	Target page (destination)	Stopping boundary
Amazon	"View past orders."	"Your Orders" history page	Before opening any order detail
Calendar	"Change the default reminder time for events to 15 minutes."	"Default notification time" settings row	Before tapping the time picker
TikTok	"Open the privacy settings for who can comment on my videos."	"Who can comment" configuration page	Before selecting a permission
Chrome	"Clear browsing history from the last hour."	"Clear browsing data" time-range dialog	Before tapping "Clear data"
Google Maps	"Switch the map style to satellite."	Map-style selector with satellite option	Before confirming the style change
WhatsApp	"Turn off read receipts."	"Read receipts" toggle row	Before flipping the toggle

Table 10: Representative DestBench task examples. Each task specifies a natural-language instruction q, a pre-action destination screen the agent must reach, and a safe stopping boundary that excludes the irreversible final operation. Success is judged by visual equivalence between the agent's final screenshot and the destination annotation; the stopping-boundary action is never executed in scoring.

B.3 Memory record schema

Each destination record $r_{h,n}$ is a tuple $(h, n, d, I, W, H, z^d, z^I)$ where h is the hypernode id, n is the member-node id, d is a natural-language page summary, I is a list of direct user-intents the page satisfies, W is a deduplicated waypoint sequence from the app root, H is a list of per-transition disambiguation hints, and z^d, z^I are DashScope `text-embedding-v4` embeddings (Zhang et al., 2025b) of d and the concatenation of I. Records are stored as JSONL with one record per line for fast top-k retrieval at inference.

B.4 Backbone configurations

Table 12 lists the shared prompting and decoding settings used for both backbones.

Setting	UI-TARS-1.5-7B	Qwen-VL-Max
Image resolution	native (1080p portrait)	native
Max input tokens	8192	API-controlled
Max output tokens	512	API-controlled
Temperature	0.0	0.0
Top-p	1.0	1.0
Action space	mobile-ops	mobile-ops
History window (steps)	3	3
Step budget T	15	15

Table 12: Backbone configurations. Same prompting and decoding hyperparameters across both backbones; only the model weights and tokenizer differ. `mobile-ops` is the shared action vocabulary (`click`, `type`, `scroll`, `back`, `home`, `wait`, `finished`).

B.5 Protocol details

Device configuration (Android emulator on Android 16, API level 36.1, ABI arm64-v8a), action space, retrieval top-k, step budgets, and seed/temperature settings used in §5. Each app runs on a freshly reset emulator image with a dedicated benchmark account; login state and account-private data are persisted across tasks within the same app but never shared across apps.

C Prompts

All prompts are reproduced verbatim. We use them with the model versions reported in Appendix H.6.

C.1 Metadata extraction prompt (page summary, direct intent, route distillation)

The page summary, direct-intent extraction, and route distillation are performed by a single VLM call that extracts all three fields jointly. Given a sequence of annotated screenshots (with green rectangles marking tap targets and arrows marking scrolls) plus optional action descriptions, the VLM returns a JSON object with `visual_page_summary`, `intent_from_current_screen`, and `navigation_memory` (containing `relevant_waypoint_sequence` and `transition_hints`).

System prompt

```
You extract compact retrieval metadata and navigation
      memory for the final Android UI screen in a visual
      trajectory.

Return only valid JSON. No markdown. No explanation.

You are given:
1. a sequence of annotated screenshots
2. optional action descriptions that help explain the
      visual annotations

The screenshots are ordered from earlier to later. The
      last screenshot is the current/final screen. Only
      the last screenshot should be treated as the target
      screen for retrieval.

The earlier screenshots show how the user arrived at the
      final screen. They may contain visualized actions:
- scroll or swipe actions are shown with arrows
- other actions are shown with green rectangles around the
       clicked target

Most important rule for retrieval metadata: focus on the
      active interaction layer of the final screenshot.

Active interaction layer:
- If a modal, dialog, popup, bottom sheet, drawer, menu,
      permission prompt, overlay, snackbar action panel,
      or foreground panel is open, treat that foreground
      layer as the current screen.
- In that case, ignore the dimmed, blurred, partially
      covered, or background content behind it.
- Do not extract intents from the background screen when a
       foreground layer is active.
- If a bottom sheet or modal covers only part of the
      screen, still treat the sheet/modal as the active
      screen and ignore unrelated background content.
- If an overflow menu or floating menu is open, describe
      the menu's purpose and visible menu options, not the
       page behind it.
- If a permission dialog or system dialog is open,
      describe the dialog's intent, not the app screen
```

```
      behind it.
- If there is no foreground overlay, use the full final
      screenshot.

Important for retrieval metadata:
- The action annotations on earlier screenshots are
      context for the path.
- If the final screenshot contains visible actions,
      controls, tabs, menu items, or destinations that
      would lead somewhere else, do not treat those as the
       user's intent.
- Do not list sibling pages or destinations visible on the
       final screen as intents.
- Do not infer that the user wants to go to another page
      just because that option is visible.
- The goal is to describe what the active final screen/
      layer itself directly satisfies if it is the
      intended destination.

Extract these fields:

visual_page_summary:
- one short phrase describing the stable purpose of the
      active final screen or foreground layer
- base this mainly on the active layer in the final
      screenshot
- use earlier screenshots only to disambiguate what the
      final active layer is
- do not describe the path or previous screens
- if a modal/dialog/menu/sheet is active, summarize that
      foreground component, not the background page

intent_from_current_screen:
- user intents directly satisfied by the active final
      screen or foreground layer itself
- include information the user can inspect on the active
      layer
- include actions the user can complete on the active
      layer without needing to navigate to another screen
- do not include intents requiring leaving the active
      layer
- do not include sibling tabs, bottom navigation
      destinations, menu entries from the background, or
      links to other pages
- do not include low-level UI mechanics
- if an overlay is open, include only intents supported by
       the overlay, dialog, menu, sheet, or foreground
      panel

navigation_memory:
- a compact reusable memory describing how to reach the
      CURRENT/final node from the trajectory
- use earlier screenshots and action descriptions to infer
       the clean semantic route to the final screen
- the CURRENT node remains the source of truth
- remove loops, retries, oscillations, backtracking, and
      irrelevant detours
- prefer the shortest semantically valid forward route to
      the CURRENT node
- if early path segments vary or are noisy, prefer the
      shared sub-route closest to the CURRENT node
- keep everything strictly in-app
- never imply leaving the app, going to the launcher, or
      using the OS home screen
- if a waypoint could be confused with OS home, rewrite it
       as an in-app destination such as "home feed tab (in-
      app)", "main feed", or "app home screen"
- waypoints must be semantic page/state labels, not action
       phrases
- transition hints must describe semantic guidance, not
      low-level gesture mechanics
- avoid coordinates, bounding boxes, brittle visual
      details, and literal event-log replay
- prefer reusable semantic labels over usernames, product
      names, exact post topics, timestamps, counts, or one-
      off visible values unless essential
- do not mention internal reasoning, clusters, node ids,
      graph-analysis language, retrieval, ranking,
      screenshots, annotations, or trajectories

Output style rules:
- use lowercase for visual_page_summary and
      intent_from_current_screen
- navigation_memory may use title-case semantic waypoint
      labels if clearer
- keep only semantically distinct intents
- prefer specific user goals over generic descriptions
- ignore os/status bar data, ads, timestamps, temporary
      feed items, and promotional content
- avoid low-level UI words in intents: tap, click, press,
      icon, button, rectangle, arrow, swipe, scroll, tab,
      menu
- return at most 8 intents
- return at most 6 waypoints
- return at most 6 transition hints
```

User prompt

```
Extract retrieval metadata and navigation memory for the
      final/current screen.

Action descriptions, in trajectory order:
{action_descriptions}

Remember:
- screenshots are provided in order
- the last screenshot is the current/final screen
- earlier screenshots are only path context
- green rectangles and arrows show actions taken in the
      trajectory
- focus retrieval metadata only on the active interaction
      layer in the final screenshot
- if a modal, dialog, popup, bottom sheet, drawer, menu,
      permission prompt, or foreground overlay is open,
      extract metadata only from that foreground layer
- ignore dimmed, blurred, covered, or background content
      behind an active foreground layer
- do not treat actions available on the final screen as
      the user's intent unless the active final screen/
      layer itself directly satisfies that intent
- do not include destinations reachable from the final
      screen or from the background screen
- for navigation_memory, infer the cleanest semantic in-
      app route that reaches the current/final screen
- remove loops, retries, oscillations, backtracking, and
      irrelevant detours
- waypoints must be semantic page/state labels, not action
       phrases
- transition hints must be semantic guidance, not low-
      level gesture instructions

Return strict JSON only with exactly this schema:
{"visual_page_summary": "", "intent_from_current_screen":
      [],
 "navigation_memory": {"relevant_waypoint_sequence": [],
 "transition_hints": []}}
```

C.2 Target selector prompt

Given the user query and a list of candidate nodes, this prompt asks the VLM to rank the candidate nodes and select the single best target. Each candidate node has a `visual_page_summary` and `intent_from_current_screen`.

System prompt

```
You are selecting the best target node in an Android UI
      graph.

You receive:
1. a user query
2. a list of candidate UI nodes
3. each candidate has: index, visual_page_summary,
      intent_from_current_screen, optional retrieval
      scores

Your job: Choose the candidate whose CURRENT screen best
      satisfies the user query as the final target node.

Important ranking rules:
- Prefer the actual target screen, not a loading screen
      before it.
- Penalize loading screens, splash screens, blank screens,
       and temporary transition screens unless the user
      explicitly asks for them.
- Penalize modals, dialogs, bottom sheets, overlays,
      drawers, and overflow menus unless the user
```

```
      explicitly asks for that foreground component.
- Do not choose a page only because it has an option that
      leads to the target.
- Do not choose a settings page merely because it mentions
       the requested feature. Example: "timer sound
      settings" is not the same as "timer section".
- For page/navigation queries such as "navigate to the
      help page", "open timer", or "go to settings":
      prioritize visual_page_summary.
- For task/action queries such as "turn off alarm", "
      change timer sound", or "send feedback": use
      intent_from_current_screen more strongly.
- If the query asks for a general help page, prefer a help
       resources/support page over a specific help article,
       unless the query asks for that article.
- If the query asks for a specific help topic, prefer the
      matching help article.
- If a candidate is a loading screen for a page and
      another candidate is the actual loaded page/content,
       choose the actual loaded page/content.
- Choose only from the provided candidate indices.

Candidate labels:
- target: the current screen is the best final target.
- acceptable: relevant and usable, but not the best.
- transient_related: loading/temporary state related to
      the target.
- waypoint: can lead to the target but is not itself the
      target.
- related_but_wrong: related terms appear, but it is not
      the target.
- wrong: irrelevant.

Return only valid JSON. No markdown. No explanation
      outside JSON.

Output schema:
{"best_index": 0, "best_label": "target", "confidence":
      0.0,
 "reason": "short reason", "ranked_candidates": [{"index":
       0,
 "label": "target|acceptable|transient_related|waypoint|
      related_but_wrong|wrong",
 "score": 0.0, "reason": "short reason"}]}
```

User prompt

```
User query: {user_query}

Application description: {app_description}

Candidate UI nodes: {compact_candidates}

Select the best target node index.

Remember:
- The target is the node whose current screen itself
      satisfies the query.
- Do not choose loading screens if an actual content page
      is available.
- Do not choose menus, modals, or bottom sheets unless the
       query asks for them.
- Do not choose a settings page just because it mentions
      the requested feature.
- For "help page", prefer the general help/support/
      resources page over a specific article unless the
      query names that article.
```

C.3 Destination-memory injection template

At the beginning of task navigation, this text block is prepended to the backbone actor's system prompt with the memory plan to complete the current task. The injected memory contains the user instruction, the retrieved waypoint sequence, and the per-transition hints. The block is fixed across the trajectory in the open-loop canonical configuration; dynamic variants (Appendix H.7) update it during execution.

```
[Navigation Memory]
Final goal: {prompt}

Relevant waypoint sequence:
{waypoint_1} -> {waypoint_2} -> ... -> {waypoint_n}

Transition hints:
- {hint_1}
- {hint_2}
- ...
```

C.4 GPT-4o judge prompt

A GPT-4o call is used to compare the agent's final screenshot against the ground-truth target screenshot and user instruction to return a 1–5 ordinal score plus a one-sentence justification. Scores 4–5 are considered a success.

```
You are evaluating whether a mobile-app navigation agent
      reached the intended target page or target control
      for a user task.

Task: {prompt}

Image 1 is the AGENT's final screen.
Image 2 is the GROUND-TRUTH target screen.

Score from 1 to 5:
5 = exact target page/control reached.
4 = correct functional page/section reached; only minor
      visual, scroll, modal, or dynamic-content
      differences.
3 = partially correct: same broad area/app section, but
      wrong specific leaf, tab, field, or subpage.
2 = wrong section of the same app, or agent is clearly not
       at the requested destination.
1 = wrong app, launcher/home, stuck, or unrelated screen.

Be strict for score 4-5. A sibling settings page, nearby
      list item, or same broad menu but wrong specific
      target is score 3, not success.

Respond with ONLY a JSON object:
{"score": 1-5, "justification": "<brief explanation>"}
```

D Additional Quantitative Results

D.1 Per-app, per-backbone, per-benchmark breakdown

Table 13 (DestBench) and Table 5 (DroidTask-Nav and SPA-Bench-Nav) give the per-app results behind Table 3; cells report successes over tasks for both the no-memory and +TAG-Nav arms.

App	No memory		+ TAG-Nav	
	UI-TARS-1.5-7B	Qwen-VL-Max	UI-TARS-1.5-7B	Qwen-VL-Max
Amazon	9/17	8/17	10/17	11/17
Calendar	5/10	7/10	4/10	7/10
Clock	1/9	4/9	3/9	7/9
Google Maps	12/24	12/24	15/24	17/24
Settings	8/9	7/9	7/9	7/9
YouTube	9/15	12/15	10/15	11/15
Total	44/84	50/84	49/84	60/84
SR (%)	52.4	59.5	58.3	71.4

Table 13: DestBench per-app drilldown (6 apps, 84 tasks). Human-annotated strict SR.

App	Graph condition	Success	No memory	Δ pp	95% CI pp	Exact p
Amazon	Random	11/17	9/17	+11.8	[+0.0, +29.4]	0.500
Calendar	Random	7/10	5/10	+20.0	[-30.0, +60.0]	0.688
Clock	Random	4/9	1/9	+33.3	[+0.0, +66.7]	0.250
Google Maps	Random	16/24	12/24	+16.7	[-8.3, +41.7]	0.344
Settings	Random	7/9	8/9	-11.1	[-33.3, +0.0]	1.000
YouTube	Random	9/15	9/15	+0.0	[-33.3, +33.3]	1.000
Amazon	Auto-BFS	10/17	9/17	+5.9	[+0.0, +17.6]	1.000
Calendar	Auto-BFS	4/10	5/10	-10.0	[-30.0, +0.0]	1.000
Clock	Auto-BFS	3/9	1/9	+22.2	[+0.0, +55.6]	0.500
Google Maps	Auto-BFS	15/24	12/24	+12.5	[-16.7, +41.7]	0.581
Settings	Auto-BFS	7/9	8/9	-11.1	[-33.3, +0.0]	1.000
YouTube	Auto-BFS	10/15	9/15	+6.7	[-13.3, +26.7]	1.000
Amazon	Manual	12/17	9/17	+17.6	[+0.0, +35.3]	0.250
Calendar	Manual	6/10	5/10	+10.0	[-20.0, +40.0]	1.000
Clock	Manual	3/9	1/9	+22.2	[+0.0, +55.6]	0.500
Google Maps	Manual	20/24	12/24	+33.3	[+12.5, +54.2]	0.021
Settings	Manual	8/9	8/9	+0.0	[+0.0, +0.0]	1.000
YouTube	Manual	10/15	9/15	+6.7	[+0.0, +20.0]	1.000

Table 14: UI-TARS-1.5-7B per-app comparisons against no memory on DestBench (human strict labels; intervals and tests as in Appendix D.2).

App	Graph condition	Success	No memory	Δ pp	95% CI pp	Exact p
Amazon	Random	9/17	8/17	+5.9	[-17.6, +29.4]	1.000
Calendar	Random	8/10	7/10	+10.0	[-20.0, +40.0]	1.000
Clock	Random	7/9	4/9	+33.3	[+0.0, +66.7]	0.250
Google Maps	Random	12/24	12/24	+0.0	[-25.0, +25.0]	1.000
Settings	Random	6/9	7/9	-11.1	[-44.4, +22.2]	1.000
YouTube	Random	12/15	12/15	+0.0	[-26.7, +26.7]	1.000
Amazon	Auto-BFS	11/17	8/17	+17.6	[+0.0, +35.3]	0.250
Calendar	Auto-BFS	7/10	7/10	+0.0	[+0.0, +0.0]	1.000
Clock	Auto-BFS	7/9	4/9	+33.3	[+0.0, +66.7]	0.250
Google Maps	Auto-BFS	17/24	12/24	+20.8	[-8.3, +50.0]	0.267
Settings	Auto-BFS	7/9	7/9	+0.0	[-33.3, +33.3]	1.000
YouTube	Auto-BFS	11/15	12/15	-6.7	[-33.3, +20.0]	1.000
Amazon	Manual	8/17	8/17	+0.0	[-29.4, +29.4]	1.000
Calendar	Manual	7/10	7/10	+0.0	[-30.0, +30.0]	1.000
Clock	Manual	6/9	4/9	+22.2	[+0.0, +55.6]	0.500
Google Maps	Manual	16/24	12/24	+16.7	[-8.3, +41.7]	0.344
Settings	Manual	7/9	7/9	+0.0	[-44.4, +44.4]	1.000
YouTube	Manual	12/15	12/15	+0.0	[-26.7, +26.7]	1.000

Table 15: Qwen-VL-Max per-app comparisons against no memory on DestBench (human strict labels; intervals and tests as in Appendix D.2).

Benchmark	Backbone	Success	No memory	Δ pp	95% CI pp	Exact p
DroidTask-Nav	UI-TARS-1.5-7B	107/150	95/150	+8.0	[+0.0, +16.0]	0.073
DroidTask-Nav	Qwen-VL-Max	106/150	99/150	+4.7	[-2.0, +12.0]	0.265
SPA-Bench-Nav	UI-TARS-1.5-7B	23/30	20/30	+10.0	[-10.0, +30.0]	0.508
SPA-Bench-Nav	Qwen-VL-Max	28/30	23/30	+16.7	[+0.0, +33.3]	0.125

Table 16: Converted-benchmark comparisons against no memory (human strict labels; intervals and tests as in Appendix D.2).

D.2 Per-app uncertainty

Tables 14 and 15 compare each graph condition with no memory per app on the 84 DestBench tasks, and Table 16 does the same for the converted benchmarks. All outcomes are human strict labels. Intervals are 95% paired bootstrap intervals over tasks (10,000 resamples), and p values come from exact two-sided McNemar tests; neither is adjusted for multiple comparisons, here or in Tables 3 and 4.

D.3 Graph-arm memory provenance

In some random- and manual-arm app cells, the harness supplied the same app's memory from a different graph condition. Table 17 repeats the comparisons of Table 4 without those apps; every gain remains positive.

Arm (excluded apps: memory received)	n	No mem.	Arm	Δ pp [95% CI]	p
UI-TARS-1.5-7B random (YouTube, Settings: Auto-BFS)	60	27	38	+18.3 [5.0, 31.7]	0.027
UI-TARS-1.5-7B manual (Clock: random-graph targets)	75	43	56	+17.3 [8.0, 28.0]	0.002
Qwen-VL-Max random (Settings: Auto-BFS)	75	43	48	+6.7 [−6.7, 20.0]	0.424
Qwen-VL-Max manual (Clock: random-graph targets)	75	46	50	+5.3 [−8.0, 18.7]	0.557

Table 17: Graph conditions without the affected apps. Each row names the excluded apps and the memory they received. Human strict success; intervals and tests as in Appendix D.2.

D.4 Per-app graph-quality breakdown

Table 18 is the per-app counterpart to Table 6. Per-app patterns make the aggregate finding concrete: deep-menu apps benefit most from any memory (the destination is several hops below the entry screen), while shallow apps like Settings and Calendar benefit least because their residual difficulty is execution rather than navigation.

D.5 Route cleanliness

Table 19 reports how properties of the injected route correlate with success in the 313 GPT-4o-judged runs of Section 5.3. Routes from manual graphs are shorter than Auto-BFS routes (2.9 vs. 6.0 waypoints on average).

D.6 Memory format and retrieval-depth ablation

Ablations supporting the canonical configuration: waypoint vs step-numbered memory crossed with top-k retrieval depth. Step-numbered observations trigger spurious `wait()` loops when the agent treats a state observation as an event; top-$k > 1$ multi-route injection pollutes context with routes that do not match the live UI. The canonical configuration is semantic waypoints with top-1 retrieval.

D.7 Cross-family and prior-memory outcomes

The AgentCPM-GUI rows in Table 7 are paired runs on the 84 DestBench tasks, with later app versions than the UI-TARS-1.5-7B and Qwen-VL-Max rows and labels blind to memory condition; memory raises success from 28/84 to 33/84 (exact McNemar $p = 0.23$). Some of these episodes began from in-app state left by the previous task, such as Clock's last tab or a Settings sub-page. The MAI-UI rows come from a separate 79-task six-app evaluation, 72 of whose tasks overlap with DestBench; three baseline tasks were not run and are counted as failures, and one baseline task lacks a label, so the two rates use different denominators and are not a paired comparison. The PG-Agent adaptation reaches 36/84 strict successes (37/84 under a second human labeling), and GUI-Explorer reaches 47/84 with a GPT-4o actor and its released knowledge base.

D.8 Cost analysis

Table 20 reports the one-time exploration effort for each core app. A walk is one exploration episode from the app root (Appendix B.2), and a forward action is an action that moves the explorer to a new or known screen; resets and undo steps are not counted. The explorer records both counts in each graph.

D.9 Record-content ablation

On 27 Instagram and eBay tasks with graphs and apps at matched versions, we vary which fields of the destination record are injected. In this cohort, transition hints alone gave the highest success (Table 21); Appendix H.2 evaluates the same tasks after app updates.

D.10 Informed no-memory baseline

To test whether structured records add value beyond general app knowledge, we compare them with a fixed, hand-written manual for each core app that gives general guidance but no task-specific destination or route (Table 22). The manual and every record variant improve on no memory by four to nine tasks, and none of the differences is significant: app-specific text captures part of the benefit of destination memory, and this diagnostic does not separate it from the structured records.

E Format-Adapted Reimplementation Protocol

E.1 Per-system adaptation rules

Table 23 lists how each prior system's memory abstraction was reconstructed on our hypernode graph for Table 7, and what was deliberately not reproduced (custom actors, validators, planners).

F Qualitative Examples

F.1 Failure trajectories per class

Figures 4–6 show one canonical trajectory per failure class from Table 24. **Class A** (wrong retrieval, 46%): the embedding-based retrieval matched the query to the wrong graph node, and the agent faithfully followed incorrect waypoints to a semantically-related but wrong destination. **Class B** (near miss, 31%): the agent reached the correct area of the app but stopped at a parent page, sibling leaf, or adjacent sub-item rather than the exact target. **Class C** (execution failure, 23%): the memory pointed in the right direction but the agent could not execute the navigation within the step budget, typically looping or scrolling without locating the specific UI element.

F.2 Success trajectories

Two representative successes illustrate how the destination block guides the actor. On Calendar, an unseen target buried four taps under a settings drawer is reached by following the waypoint sequence verbatim; the actor never opens the side-menu it would have explored without memory. On Google Maps, a multi-route destination (reachable through search or through the saved-places tab) is solved by the shorter cleaned route stored in the record; the per-transition hints disambiguate a tab whose icon collides with a sibling control.

F.3 Raw destination-record examples

Each record stores a page summary, a list of direct intents, a waypoint sequence from the app root, and per-transition hints. A representative record for the YouTube target "open the watch-history list" contains the summary "screen listing previously watched videos with per-item remove buttons," the intent "view watch history," the waypoint sequence `Home` $\rightarrow$ `Library` $\rightarrow$ `History`, and the hint "Library is the bottom-bar icon labeled You, not the side-drawer entry of the same name." Records are stored as JSONL and never modified after exploration.

G Detailed Related-Work Positioning

Table 25 expands the categorical positioning in Table 1, using concise fields so the comparison stays readable in the appendix flow.

H Limitations, Robustness, and Ethics

H.1 Dynamic-screen handling

DestBench screens contain dynamic content: advertisements, recommendation feeds, trending lists, and time-varying notifications. The metadata-extraction prompt (Appendix C.1) is written to focus on the persistent active layer, and the judge prompt (Appendix C.4) is told to tolerate cosmetic differences. Persistent dynamic state (e.g., login-gated content, A/B layouts) is a known failure mode and is documented per-app in our records.

App	No memory	Random	Auto-BFS	Manual
Amazon	9/17	11/17	10/17	12/17
Calendar	5/10	7/10	4/10	6/10
Clock	1/9	4/9	3/9	3/9
Google Maps	12/24	16/24	15/24	20/24
Settings	8/9	7/9	7/9	8/9
YouTube	9/15	9/15	10/15	10/15
Total	44/84	54/84	49/84	59/84
Success Rate	52.4%	64.3%	58.3%	70.2%

Table 18: Per-app graph-quality breakdown on DestBench using UI-TARS-1.5-7B. Cells are `successes / attempts` under the strict human-scoring rule; all cells are human-labeled (Appendix D.2).

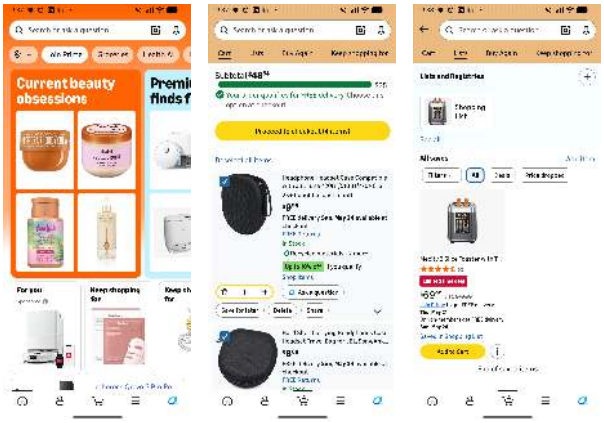


Figure 4: Class A Failure (Wrong retrieval) Task: "Visit the user's saved cards" (Amazon). Memory: `Shopping Cart` → `Lists Tab` → `Lists and Registries`. The retrieval system incorrectly matched the task to the Lists node. The agent then followed the memory faithfully: (1) Started at Amazon home page, (2) Navigated to Shopping Cart page and tapped the Lists tab, (3) Arrived at the Lists and Registries page showing a Shopping List. The embedding confused "saved cards" from the task with "saved items/lists."

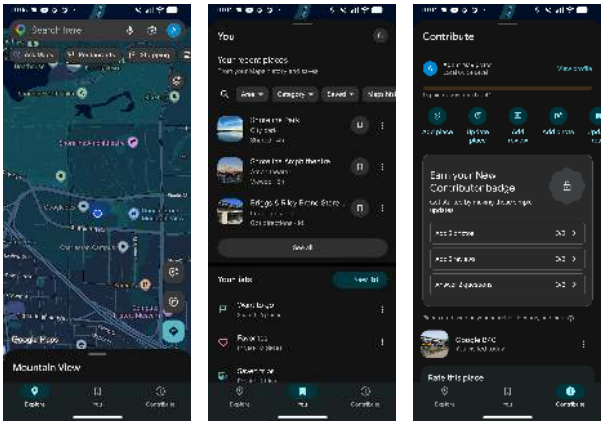


Figure 5: Class B Failure (Near miss) Task: "View the user's contribution profile" (Google Maps). Memory: `Google Maps Home` → `Contribute hub` → `Profile details`. The agent followed the memory: (1) Started on the Maps home page, (2) Tapped the Contribute tab, (3) Arrived at the Contribute page showing the "View profile" link. It then declared success without clicking it. The retrieval selected the Contribute hub node, which is the parent node to the correct Profile details node.

Injected-route measure	n	Point-biserial r	p
Route length (waypoints)	313	−0.16	0.006
Transition-hint count	313	−0.19	0.001
Loop count	313	−0.03	0.652
Backtrack count	313	+0.04	0.467
Detour count	313	+0.02	0.783

Table 19: Route-cleanliness associations. GPT-4o-judged runs of Section 5.3. Route length and hint count correlate negatively with success; this observational analysis does not establish causality.

H.2 Cross-app-version robustness

Graphs are tied to a specific app version on a specific device configuration. In a limited 27-task probe on Instagram (about five months between versions) and eBay (about ten weeks), full-record memory matched no memory at both versions (Table 26). This probe does not establish general cross-version robustness. Apps with sufficiently large navigation-affecting changes still require the graph to be revalidated or rebuilt before its records can be trusted.

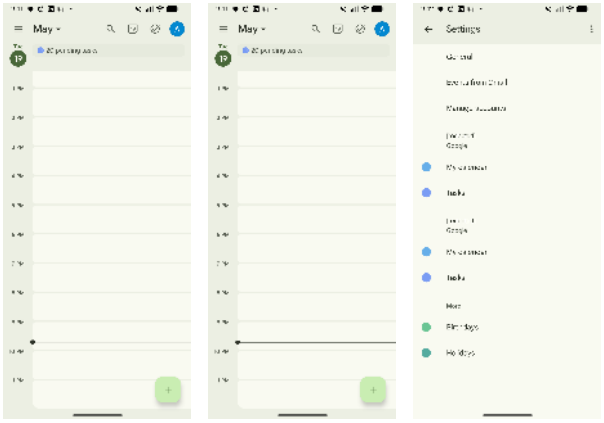


Figure 6: Class C Failure (Execution failure) Task: "Toggle Use device time zone" (Calendar). Memory: `Calendar Day View` → `Side Navigation Menu` → `Settings Page` (correct). The agent followed the memory: (1) Started on the Calendar day view, (5) Remained on the same Calendar view after 4 wasted steps, and (15) Finally reached the Settings page but exhausted the step budget (T=15) before locating the specific toggle. The memory was correct but the agent could not efficiently navigate the multi-level settings hierarchy. Account identifiers in the final screenshot are redacted.

Core app	Walks	Forward actions
Maps	869	3,166
Amazon	804	2,440
YouTube	638	2,174
Clock	994	3,990
Settings	148	510
Calendar	209	655
Total	3,662	12,935

Table 20: Exploration effort per core app. Walks and forward actions recorded during graph construction (Appendix D.8).

Record condition	Instagram	eBay	Total
Transition hints	4/11	13/16	17/27
Full record	2/11	12/16	14/27
No memory	4/11	10/16	14/27
Route only	2/11	11/16	13/27
Goal only	2/11	10/16	12/27

Table 21: Version-matched record-content ablation. Human-judged Instagram/eBay tasks at matched graph versions. The small sample is suggestive.

Condition	Success	Δ	p
No memory	35/80	–	–
Goal-only record	39/80	+4	0.388
Hand-written app manual	41/80	+6	0.109
Full record	39/80	+4	0.541
Route-only record	42/80	+7	0.167
Transition-hints record	44/80	+9	0.108

Table 22: Informed no-memory baseline. UI-TARS-1.5-7B, human-judged strict success on 80 core-app tasks. The app manual gives fixed app-level guidance without task-specific destinations or routes. p: exact two-sided McNemar against no memory.

H.3 Multi-path and ambiguous-target handling

When a destination is reachable by multiple distinct routes (e.g., side-menu vs bottom-tab), the record stores all routes that the explorer found and the top-1 retrieval selects the shortest cleaned route. When an instruction admits multiple visually-distinct targets, the selector prompt (Appendix C.2) breaks ties using the five preference rules; if all preferences are tied, the lower-depth target wins.

System	Backbone	Protocol adaptation
GUI-Explorer	GPT-4o	Used their published interaction-triple KB, retrieval, and guideline-generation pipeline on our emulator/app versions; exploration was not rerun.
PG-Agent	Qwen2.5-VL-72B	Built a page graph from DestBench task episodes with their construction pipeline, retrieved page guidelines by screen embedding, and replaced the multi-agent planner with a single Qwen2.5-VL-72B navigator.

Table 23: Evaluation protocol for prior systems on DestBench. Each row keeps the system's memory abstraction but uses our tasks and human-annotated strict-SR judge.

Class	N	%	Addressed by
A: wrong retrieval	22	46	embeddings, coverage
B: near miss	15	31	target selection
C: execution failure	11	23	step budget, replanning

Table 24: Failure taxonomy. Labeled open-loop DestBench failures.

System	Stored memory	Retrieved unit	Coupling
AutoDroid / AppAgent	UI graph + element notes	Element hint / note	Own LLM
KG-RAG	UTG distilled into KG store	Intent–trajectory pair	Own agent loop
GUI-explorer	Interaction-triple store	Current-screen cue	Own MLLM
GUI-Xplore	Video transition graph	App-level prior	Own VLM
PG-Agent / GraphPilot	Page / app knowledge graph	Guideline or action rule	Planner + validator
TAG-Nav	**Nav. graph + dest. records**	**Target page + compact route**	**Unmodified actor**

Table 25: Detailed positioning of TAG-Nav. Prior systems retrieve hints, trajectories, guidelines, or action rules and are coupled to their own actors; TAG-Nav retrieves a structured destination record for an unchanged stepwise actor.

H.4 Privacy and sensitive-screen handling

We use dedicated benchmark accounts for each app, with no real user data. Screens that would contain personal data on a real account (e.g., contacts, message bodies, payment instruments) are blank or contain synthetic placeholders. Any public release of screenshots requires screening for identifying

App	Condition	Matched	Updated
Instagram	No memory	4/11	1/11
	Full record	2/11	2/11
	Transition hints	4/11	4/11
eBay	No memory	10/16	10/16
	Full record	12/16	9/16
	Transition hints	13/16	11/16
Total	No memory	14/27	11/27
	Full record	14/27	11/27
	Transition hints	17/27	15/27

Table 26: Limited stale-graph probe. Human-judged paired tasks using the original graph at matched and updated app versions. Instagram: 416.0.0.47.66 to 437.0.0.33.78; eBay: 6.242.0.2 to 6.251.0.1. Full-record memory matched no memory in aggregate at both versions, with differing per-app outcomes; this small study does not establish general cross-version robustness.

content.

H.5 Use of AI in paper preparation

LLM-based assistants were used for literature and manuscript review, drafting, LaTeX editing, and analysis scripts. The authors are responsible for verifying all statements, evidence, and calculations and for approving the final manuscript. The assistants are not authors.

H.6 Models used in the TAG-Nav pipeline

All hosted models were called through Alibaba Cloud's DashScope API, except the GPT-4o judge, which used the OpenAI API. Exploration runs UI-TARS-1.5-7B locally for action grounding, uses `qwen3.6-plus` (Qwen Team, 2026b) to extract interactive elements, and uses `qwen3.6-flash` or UI-TARS-1.5-7B for loading-screen checks. Hypernode matching uses CLIP ViT-B/32 screenshot embeddings and `text-embedding-v4` page-layout embeddings (Appendix B.1). Offline post-processing (page summary, direct intents, waypoint route, and transition hints) makes one `qwen3.6-plus` call per root-to-state path. Record fields and user instructions are embedded with `text-embedding-v4`, a Qwen3-Embedding model (Zhang et al., 2025b), and the target selector is `qwen3.5-flash` (Qwen Team, 2026a). The navigation actors are UI-TARS-1.5-7B and Qwen-VL-Max (`qwen-vl-max-latest`); the closed-loop variant (Appendix H.7) also describes the live screen with Qwen-VL-Max. GPT-4o (Hurst et al., 2024) is the evaluation judge. No proprietary data leaves the local environment except through these API calls.

H.7 Dynamic-memory variants

The closed-loop variant localizes the current screen against the graph at every step and refreshes progress annotations; the plan-edit variant rewrites the memory block when the actor signals confusion. Both are controlled ablations of the canonical open-loop TAG-Nav configuration evaluated in Section 5.