GCFSIM: LEARNING GOAL-CONDITIONED FUNCTIONAL SIMILARITY FOR MOBILE GUI NAVIGATION

Anonymous authors
Paper under double-blind review

ABSTRACT

GUI agents trained on large corpora of mobile-application trajectories can solve a wide range of user-requested tasks. A key challenge, however, is adapting such agents to a particular application without increasing inference latency. A promising approach is to explore the application offline, store application-specific knowledge, and retrieve relevant experience during navigation. This raises a fundamental question: *which previously observed screen should the agent retrieve when deciding what to do next?* We show that neither visual similarity nor goal progress alone is sufficient for this purpose. Two screens may look very different yet support equivalent useful actions for a particular task, while two states at the same distance from the goal may require different next actions.

We formulate application-memory retrieval as *goal-conditioned graph localization*. Given the current screenshot and user goal, the objective is to retrieve a state from the explored application graph that provides useful guidance for the agent's next decision. For each goal, graph states are grouped according to the optimal actions they support and the goal-directed continuations reached through those actions. We use these functional relationships as supervision to learn **GCFSim**, a lightweight goal-conditioned visual–language representation $z_\theta(S, g)$. Although graph structure is used to construct training supervision, inference requires only the current screenshot and user goal.

We evaluate GCFSim on a new cross-platform navigation corpus spanning 49 Android and HarmonyOS applications, with 7,903 observed UI states, 39,154 transitions, and 36,351 automatically generated navigation intents. The code and dataset are available anonymously for review[*] and will be publicly released upon acceptance. Using the same application memories, functional supervision retrieves states that provide more useful navigation guidance than visual similarity, shortest-path distance supervision, and goal-conditioned value or Q representations. These results show that a useful retrieved state need not resemble the current screen or represent the same amount of progress toward the goal. What matters is whether the two states play the same functional role in completing the user's task.

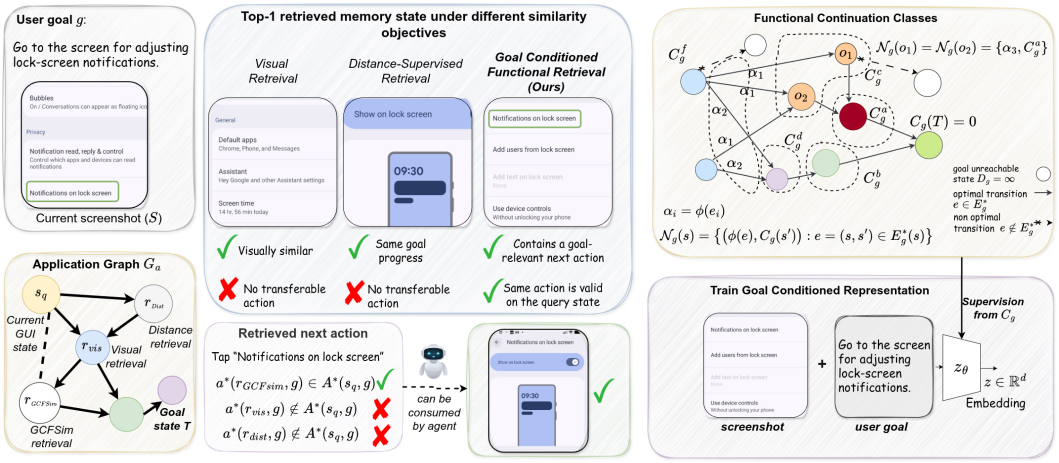
1 INTRODUCTION

Vision–language models have enabled increasingly capable GUI agents that solve user tasks in mobile applications through screenshots and low-level actions (Hong et al., 2024; Wang et al., 2024; Lin et al., 2025; Gou et al., 2025). Their progress has been supported by large-scale interaction datasets and benchmarks such as AndroidInTheWild and GUIOdyssey (Rawles et al., 2023; Lu et al., 2025), with more recent systems scaling end-to-end GUI policies through larger interaction corpora and deliberate action reasoning (Qin et al., 2025; Zhou et al., 2025). Despite this progress, a general-purpose GUI agent may still lack detailed knowledge of a particular application, including its navigation structure and application-specific functionality.

One way to provide such knowledge without fine-tuning the agent is to separate *exploration* from *inference*. An application can first be explored offline, and the acquired navigation knowledge can then be stored and reused for future tasks (Zhang et al., 2025; Wen et al., 2024; Sun et al., 2025;

[*] Anonymous code repository: `https://anonymous.4open.science/r/gcfsim-F1DD/`. Anonymous dataset: `https://figshare.com/s/cc690180b971777812a4`.

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



(a) **Retrieval by similarity objective** (b) **Graph supervision**

Figure 1: **Goal-conditioned functional retrieval for GUI navigation.** **(a)** Visual and distance-supervised retrieval can return states whose stored actions do not transfer to the current state s_q; GCFSim instead retrieves a functionally compatible state with a valid next action. **(b)** The explored graph G_a induces goal-conditioned continuation classes C_g, which supervise the learned screenshot–goal representation.

Xie et al., 2025). Recent approaches increasingly organize this knowledge as an application graph, where nodes represent observed UI states and edges represent interactions between them (Chen et al., 2025; Guan et al., 2025; Chai et al., 2026).

The challenge is how to use this graph efficiently during inference. An explored application may contain hundreds of states and many more transitions, making it impractical to provide the entire graph to the GUI agent at every step. Doing so increases the amount of information the agent must process and can substantially increase inference latency. Instead, the current screen should first be *localized* within the stored application knowledge so that only useful navigation guidance is retrieved. This leads to the central question of this work: *given the current screenshot and user goal, which previously seen screen should the agent retrieve to decide what to do next?*

Existing approaches commonly rely on visual or semantic similarity. Screenshot-based retrieval is lightweight and fast; UI-KOBE, for example, first retrieves candidate graph states using screenshot similarity before semantic verification (Chai et al., 2026). However, visual similarity does not necessarily reflect navigation usefulness. Two screens can look completely different yet support equivalent useful actions for a particular user task, while visually similar screens may require different actions. Semantic localization can provide richer information by describing the current page and matching it against stored descriptions, but doing so requires an additional vision–language model inference, which is less attractive for latency-sensitive GUI navigation.

Our key observation is that localization should not depend on the current screen alone: the agent also knows the *user goal*. The goal changes which states are useful. Two states that are interchangeable for one goal may require different decisions for another. Goal progress alone is also insufficient: two states can be equally many steps from the desired target while requiring different next actions. Rather than asking which stored state looks most similar to the current screen, we therefore ask which state is most *functionally similar for the current goal*. Figure 1 illustrates this distinction. For the goal of adjusting lock-screen notifications, visual and graph-distance retrieval return states that do not provide a transferable next action. GCFSim instead retrieves a visually different state that supports the appropriate goal-directed decision.

We formalize this problem as *goal-conditioned graph localization*. Given the current screenshot and user goal, the objective is to retrieve a state in the application graph that provides useful guidance for what the agent should do next. We use the explored graph as *privileged supervision*: for each goal, states are grouped according to the optimal actions they support and, recursively, the goal-directed continuations reached by those actions. The resulting *functional continuation classes* capture when two states can serve as interchangeable sources of navigation guidance for the current goal.

We distill these graph-derived relationships into **GCFSim**, a lightweight goal-conditioned visual–language representation $z_\theta(S, g)$. Graph topology, goal distances, transition annotations, and continuation classes are used only during training. At inference time, GCFSim requires only the current screenshot and natural-language goal, allowing stored application states to be ranked through lightweight embedding similarity without graph traversal or an additional VLM call for page understanding.

Our formulation is related to goal-conditioned state abstraction and bisimulation (Schaul et al., 2015; Zhang et al., 2021; Hansen-Estruch et al., 2022). Unlike general state abstractions that preserve broader action–transition behavior, our relation is specialized for application memory retrieval and considers the optimal transitions relevant to the current goal. Detailed connections to these formulations are discussed in Sec. 2.

We construct a cross-platform mobile navigation corpus covering 49 Android and HarmonyOS applications, with 7,903 observed UI states, 39,154 observed transitions, and 36,351 automatically generated navigation intents. Across applications, GCFSim improves navigation-compatible retrieval over visual similarity, shortest-path distance supervision, goal-conditioned value/Q representations, graph-based alternatives, and visual–semantic reranking. Controlled experiments further show that the advantage of functional supervision persists when training pairs, batch schedules, architecture, loss formulation, optimization, and training budget are held fixed. Finally, GCFSim-retrieved memories improve first-action selection across multiple frozen GUI agents, showing that improved retrieval translates into downstream navigation utility.

Our contributions are:

- We formulate *goal-conditioned graph localization*, where retrieval seeks a previously observed state that can provide interchangeable navigation guidance for the current screenshot and user goal.
- We introduce **GCFSim**, a graph-supervised metric-learning framework that recursively captures the optimal actions available at a state and the goal-directed continuations reached by those actions, and distills this structure into a lightweight screenshot–goal representation requiring no graph traversal or additional VLM call at inference time.
- We construct a 49-application Android/HarmonyOS navigation corpus and demonstrate improvements over visual, goal-distance, value/Q, graph-based, and semantic-reranking alternatives, together with improved first-action selection across multiple frozen GUI agents.

2 Related Work

2.1 GUI Agents and Application-Specific Memory

General-purpose GUI agents have advanced through large interaction datasets, stronger visual grounding, and screenshot-based policies. AndroidInTheWild and GUIOdyssey provide large-scale mobile interaction data (Rawles et al., 2023; Lu et al., 2025), while systems such as CogAgent, Mobile-Agent-v2, ShowUI, UGround, UI-TARS, and MAI-UI improve GUI perception, grounding, action prediction, and end-to-end navigation (Hong et al., 2024; Wang et al., 2024; Lin et al., 2025; Gou et al., 2025; Qin et al., 2025; Zhou et al., 2025). Our work is complementary to improving the underlying policy: we study how knowledge acquired from prior interaction with a target application should be retrieved and reused.

Several systems explicitly separate application exploration from task-time execution. AppAgent and AutoDroid acquire reusable application knowledge through exploration or application analysis (Zhang et al., 2025; Wen et al., 2024), while later work extends this paradigm through autonomous exploration (Sun et al., 2025; Xie et al., 2025) and structured application memory. PG-Agent, KG-RAG, UI-KOBE, GraphPilot, and Executable Agentic Memory use application graphs or related structures to support retrieval, planning, or action selection (Chen et al., 2025; Guan et al., 2025; Chai et al., 2026; Yu et al., 2026; Qin et al., 2026). Other work studies more general structured or task-oriented GUI memory (Zhu et al., 2026; Liu et al., 2026). These methods motivate persistent application memory; our focus is the orthogonal retrieval question: *which stored state should provide guidance for the current screen and goal?*

2.2 Goal-Conditioned Retrieval and State Similarity

GUI screen representations have long been studied for interface search and interaction modeling. Rico supports data-driven interface retrieval (Deka et al., 2017), Screen2Vec learns semantic screen representations from visual, textual, and interaction context (Li et al., 2021), and more recent work develops visual–language or structure-aware GUI representations (Cheng et al., 2024; Gou et al., 2025; Wei et al., 2025; Ayli et al., 2025).

Application memory systems likewise retrieve states using visual, semantic, or structural similarity. UI-KOBE combines screenshot retrieval with semantic verification (Chai et al., 2026), while PG-Agent and KG-RAG combine retrieved knowledge with graph search or language-model reasoning (Chen et al., 2025; Guan et al., 2025). In contrast, we learn the retrieval relation itself: two states should be close when one can substitute for the other as a source of goal-directed navigation guidance. This relation is inherently goal dependent and need not coincide with visual or semantic resemblance.

Our formulation is also related to goal-conditioned representation learning and state abstraction. Universal Value Function Approximators condition state utility on a goal (Schaul et al., 2015); bisimulation-based methods learn representations that preserve behaviorally relevant reward and transition structure (Zhang et al., 2021), including goal-conditioned equivalence (Hansen-Estruch et al., 2022); and contrastive goal-conditioned reinforcement learning connects representation similarity with goal-conditioned value functions (Eysenbach et al., 2022). Rather than learning a value, Q-function, or policy from rewards, we use an explored application graph as privileged supervision and directly optimize a nearest-neighbor retrieval geometry for functional interchangeability. We compare against goal-conditioned value and Q representations under the same retrieval protocol.

3 Method

We learn a goal conditioned representation in which UI states are similar when they provide functionally interchangeable navigation guidance for a user goal, rather than merely when they look alike. During training, an explored application graph provides *privileged supervision*; at inference time, the learned representation requires only a screenshot and a natural language goal. We represent each application as a directed graph $G_a = (V_a, E_a)$, where nodes are observed UI states and edges are executed interactions (Appendix A).

3.1 Goal Conditioned Functional Equivalence

Goal directed graph structure. For a goal g, let $T_g^{\mathrm{sem}} \subseteq V_a$ denote the observed states recognized as satisfying that goal (Appendix B). The remaining shortest path distance is $D_g(s) = \min_{t \in T_g^{\mathrm{sem}}} d_{G_a}(s,t)$; states with $D_g(s) = \infty$ are excluded.

Among the outgoing transitions of a reachable state, we retain

$$E_g^*(s) = \{(s,s') \in E_a : D_g(s') = D_g(s) - 1\},$$

which are the graph optimal transitions toward g. Distance therefore captures how much progress remains, but not necessarily which decision should be taken next: two states at the same distance can expose different useful actions.

Recursive continuation classes. Let $\mathcal{A}$ denote a finite vocabulary of coarse functional action types, and let $\phi : E_a \to \mathcal{A}$ map each observed transition to its abstract type, such as *open*, *select*, *input*, or *scroll*. Details are given in Appendix C.1.

We recursively characterize what useful navigation remains from each state. All target states receive the terminal class $C_g(s) = 0$ for $s \in T_g^{\mathrm{sem}}$. For every other reachable state, define

$$\mathcal{N}_g(s) = \left\{(\phi(e), C_g(s')) : e = (s,s') \in E_g^*(s)\right\}, \qquad C_g(s) = \Phi_g(\mathcal{N}_g(s)), \tag{1}$$

where Φ_g assigns the same class identifier exactly to states with identical continuation signatures. We use set semantics, so repeated transitions with the same abstract type and successor class do not alter the signature. Since every edge in E_g^* decreases D_g by one, the classes can be computed recursively outward from the targets.

Intuitively, two states receive the same class when they support the same abstract optimal decisions and those decisions lead to equivalent goal directed continuations. We define functional equivalence by

$$s_i \sim_g s_j \iff C_g(s_i) = C_g(s_j). \tag{2}$$

This relation is stricter than equal goal distance: $C_g(s_i) = C_g(s_j)$ implies $D_g(s_i) = D_g(s_j)$, but the converse does not generally hold. Thus, goal distance tells us *how far* a state is from the target, whereas C_g additionally captures *which useful decisions remain and where they lead*.

Proposition 1 (Necessity of goal conditioning)**.** *There exist states s_i, s_j and goals g_1, g_2 such that $s_i \sim_{g_1} s_j$ but $s_i \not\sim_{g_2} s_j$. Hence, a representation that does not condition on the goal cannot in general preserve functional equivalence for all goals.*

Concrete examples of this goal-dependent change in functional equivalence are shown in Appendix K.3, Fig. 8.

The continuation classes compress the application graph by replacing several goal equivalent UI states with one abstract state. Two states can be merged when they support the same abstract optimal actions and those actions lead to the same abstract futures. For a fixed ϕ, C_g gives the most compressed state representation that still preserves this goal directed navigation structure.

Proposition 2 (Maximal plan preserving compression)**.** *Fix $\phi : E_a \to \mathcal{A}$ and consider the optimal goal directed subgraph $G_g^* = (V_g, E_g^*)$. Among state abstractions that preserve the abstract optimal transition types defined by ϕ and recursively preserve the equivalence of their successor states, the continuation classes C_g give the coarsest partition of V_g. States from different C_g classes therefore cannot be merged without losing optimal plan structure under ϕ.*

Proofs are provided in Appendix D.

3.2 Distilling Functional Equivalence

We distill the graph derived functional relation into a representation computed directly from a screenshot and goal, as illustrated in Figure 1b. With frozen image and text encoders, $x_S = E_I(S)$ and $x_g = E_T(g)$, we learn $z_\theta(S, g) = F_\theta([x_S, x_g, x_S \odot x_g])$, where F_θ is a lightweight projection. Conditioning on g allows the same screenshot to occupy different regions of the embedding space for different goals.

Training. States in the same C_g class form positive pairs, while states in different classes form negatives. Sampling emphasizes same-distance states, visually similar but functionally incompatible states, and *goal flip* cases whose relation changes across goals. We optimize $\mathcal{L} = \mathcal{L}_{\text{BCE}} + \lambda_f \mathcal{L}_{\text{flip}} + \lambda_s \mathcal{L}_{\text{sup}} + \lambda_n \mathcal{L}_{\text{NCE}} + \lambda_h \mathcal{L}_{\text{hard}}$, including online hard negative mining. Pair construction, loss definitions, coefficients, and optimization details are provided in Appendix E.

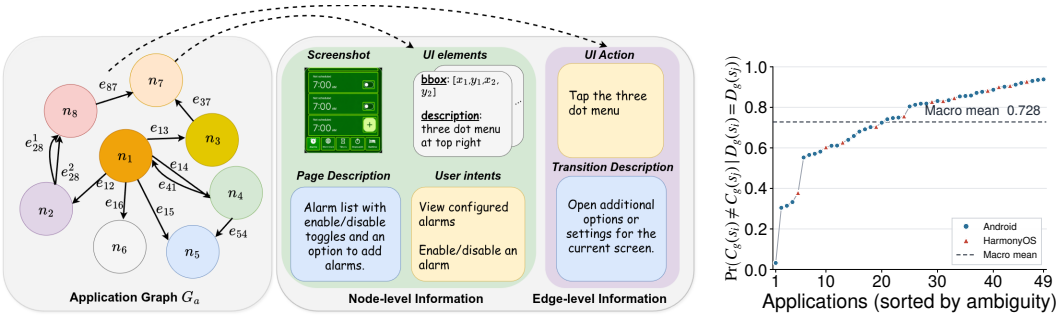
Retrieval. At inference time, the query (S_q, g) and memory states from the same application are ranked by cosine similarity in the learned embedding space. Graph topology, goal distances, continuation classes, node identifiers, and transition annotations are not required to compute retrieval scores at inference.

4 Experiments

We evaluate GCFSim along four dimensions: retrieval quality against visual, textual, distance-based, value/Q, and graph-based alternatives; downstream utility for frozen GUI agents; the specific contribution of functional continuation supervision; and robustness to transition abstraction and graph noise. We further analyze goal dependence and residual retrieval failures.

4.1 Experimental Setup

Application corpus. We evaluate on **49 mobile applications** spanning diverse interfaces and interaction patterns. Each application is explored to construct a graph of observed UI states and transitions (Fig. 2a); construction and annotation details are provided in Appendix A. A manual audit of



(a) **Application graph and stored information**

(b) **Same-distance functional ambiguity**

Figure 2: **Application graphs and distance ambiguity.** **(a)** Nodes store UI states and semantic annotations; edges store executed interactions and transition descriptions. **(b)** Fraction of equal-distance state pairs with $C_g(s_i) \neq C_g(s_j)$, sorted by application; markers denote platform. The macro average is 0.728 over 1,030,979 pairs.

300 semantic annotations finds validity rates of 99.0% for page descriptions, 91.9% for goal target associations, and 96.0% for transition descriptions; exact destination targets are valid in 95.0% of cases (Appendix F). Corpus statistics are reported in Table 3.

Supervision and splits. Semantic target expansion uses $\tau_g = 0.95$, and functional classes follow Sec. 3.1 with set semantics. GCFSim is trained separately for each application: each application has its own checkpoint trained only on that application's training goals, with no parameter sharing across applications. For each application, we use 14 training goals, 5 validation goals, and 5 test goals, giving 686 training, 245 validation, and 245 test goals across the 49 applications. The test goals yield 2,403 evaluation queries in total. The same fixed splits are reused for all learned methods, which are evaluated over three training seeds.

Retrieval protocol. We use *leave one visual* evaluation to avoid trivial screenshot matching. For each query s_q, we remove the query state and screenshots with frozen visual similarity above $\tau_{\text{dup}} = 0.92$, then retrieve from the remaining states of the same application. All methods use the same candidate memory. At inference, GCFSim requires only the screenshot and goal; graph information is used only for training supervision.

Evaluation metrics. Since retrieved memory guides the agent's next decision, our primary metric is NEXTACTION@1. Action-level evaluation uses shortest-path distance to the original exact goal targets T_g, denoted D_g^{exact} (Appendix G). Let

$$\mathcal{A}^*_{\text{exact}}(s_q, g) = \left\{ a(s_q, s') : D_g^{\text{exact}}(s') = D_g^{\text{exact}}(s_q) - 1 \right\}.$$

For the top retrieved state $r^\star$,

$$\text{NEXTACTION@1} = \mathbb{1}\left[\hat{a}(s_q \mid r^\star, g) \in \mathcal{A}^*_{\text{exact}}(s_q, g)\right].$$

Any exact-target optimal continuation is accepted. We additionally report CONFLICT NA, OPTIMALACTIONSEMSIM@1, EQUIV@1, and GOALFLIP; full definitions are provided in Appendix G. Results are macro averaged across the 49 applications. Learned methods report mean±standard deviation over three seeds, and paired differences use 95% application-level bootstrap confidence intervals with 10,000 samples.

4.2 MAIN RETRIEVAL RESULTS

Distance ambiguity. Equal goal distance does not imply compatible navigation guidance: 72.8% of equal-distance state pairs belong to different functional continuation classes (Fig. 2b). Thus, equal goal progress can still require different next decisions. Details are provided in Appendix H.2.

We compare GCFSim with visual and text retrieval, Goal Distance, GC Value, GC Abstract Q, VisualGNN, visual–distance fusion, and a 7B reranking baseline. Goal Distance captures remaining

Table 1: **Retrieval performance and controlled supervision ablation.** **(a)** Comparison with retrieval baselines on our application corpus. **(b)** Effect of changing only the supervision relation while keeping training pairs, batch schedules, architecture, loss formulation, optimization, training budget, and retrieval protocol fixed. Results use application macro averaging; learned methods report mean±standard deviation across three seeds.

(a) **Main retrieval results.**

Method	NA@1 ↑	Conflict NA ↑	OptimalAction SemSim@1 ↑	Equiv@1 ↑
Visual SigLIP	.342	.000	.582	.255
TextDense GoalOnly	.103	.078	.480	.040
Goal Distance	.463±.010	.338±.004	.609±.006	.413±.015
Visual + Distance	.469±.007	.333±.003	.613±.005	.417±.012
GC Value	.362±.008	.233±.003	.565±.003	.176±.002
GC Abstract Q	.423±.007	.317±.009	.600±.003	.254±.002
VisualGNN	.486±.004	.386±.004	.617±.001	.600±.014
Visual SigLIP Top 5 + Qwen 7B	.339	.126	.600	.280
GCFSim (ours)	**.517±.005**	**.401±.008**	**.637±.004**	**.670±.005**

(b) **Strict control of the supervision relation.** All variants use identical training pairs, batch schedules, architecture, loss formulation, optimization, and training budget; only the supervision relation and its induced labels change.

Supervision relation	NA@1 ↑	Conflict NA ↑	GoalFlip ↑	Equiv@1 ↑
Visual similarity	.333±.007	.142±.006	.345±.006	.251±.008
Goal distance D_g	.446±.005	.330±.014	.440±.008	.398±.008
Abstract Q $\mathcal{A}^*_\phi$	.409±.003	.285±.007	.520±.003	.376±.001
Functional continuation C_g	**.486±.004**	**.361±.001**	**.585±.006**	**.569±.002**

progress, GC Value learns a goal-progress representation, and GC Abstract Q captures immediately optimal abstract actions; GCFSim additionally models the continuations reached by those actions. All methods use the same queries and candidate memories. Implementation details are provided in Appendix H.3. Table 1a shows that GCFSim achieves the highest NEXTACTION@1 at .517±.005, compared with .486±.004 for VisualGNN and .463±.010 for Goal Distance. Visual–distance fusion reaches only .469±.007, leaving a .048 gap to GCFSim (95% CI $[+.034, +.062]$). GCFSim also performs best on visual conflict queries and OPTIMALACTIONSEMSIM@1, showing that functional continuation captures decision-relevant structure beyond appearance, goal progress, and immediate optimal actions. Representative qualitative examples of these retrieval differences are shown in Appendix K, Fig. 6.

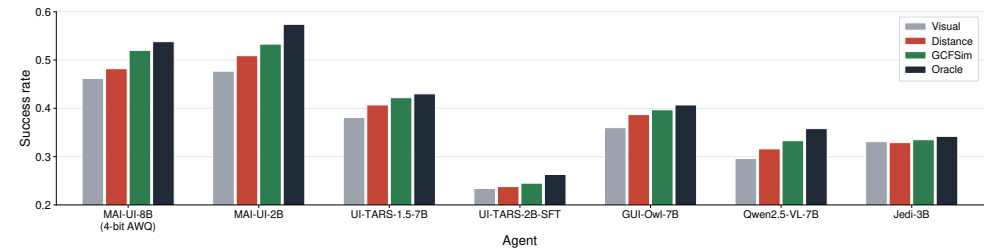
Retrieval efficiency. GCFSim requires 29.6 ms per query, comparable to other lightweight learned retrievers and substantially faster than the 7B reranking baseline (420 ms). Full latency and parameter comparisons are in Appendix H.5.

4.3 DOWNSTREAM UTILITY FOR FROZEN GUI AGENTS

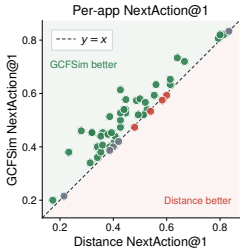
We next ask whether better retrieval leads to better decisions by frozen GUI agents. For each query, the retrieved state provides up to three candidate goal directed transitions as additional context in a fixed prompt. The agent then predicts its first action on the current query screen. Agent weights, decoding settings, and prompts remain fixed across conditions; only the retrieval source changes. We evaluate seven agents from different model families and scales.

We score each prediction using FIRSTACTIONSUCCESS: the predicted action is grounded to an outgoing edge of the query state and counted as successful if its resulting successor is exact-target graph optimal for the current goal. Thus, the metric tests whether the agent chooses a valid shortest path next action for the current state and goal. The *Success Rate* reported in Fig. 3a is the application-macro average of FIRSTACTIONSUCCESS. Full grounding and evaluation details are provided in Appendix I.

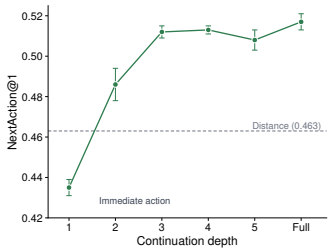
Figure 3a shows that GCFSim provides the strongest non oracle memory for all seven agents. For MAI-UI-8B, Success Rate increases from .370 without memory to .482 with Goal Distance retrieval and .520 with GCFSim. GCFSim also outperforms Goal Distance for every other evaluated agent,



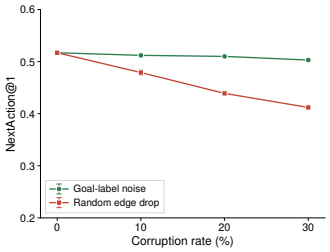
(a) **Frozen agent guidance**



(b) **Per application retrieval**



(c) **Continuation depth**



(d) **Graph robustness**

Figure 3: **Downstream utility and analysis of functional retrieval. (a)** GCFSim memory gives the highest non oracle Success Rate for all seven frozen agents. **(b)** GCFSim improves NEXTACTION@1 over Goal Distance for most applications. **(c)** Adding continuation structure beyond immediate optimal actions substantially improves retrieval, with most gains reached by depth 3. **(d)** GCFSim is robust to goal annotation noise, while missing graph edges cause larger degradation.

showing that the benefit of functional retrieval transfers across different agent architectures and model scales.

Functional retrieval is especially useful when goal distance is ambiguous. Goal Distance is competitive overall, but states with the same remaining distance can still require different next decisions. We therefore isolate queries for which candidate memory states at the same goal distance belong to multiple functional continuation classes, while at least one candidate functionally compatible with the query remains available. This subset is defined entirely from the application graph and candidate memory, independent of which state GCFSim retrieves.

On these queries, GCFSim outperforms Goal Distance for all seven frozen agents (Appendix I.4). The largest gains occur for MAI-UI-8B (.591 vs. .526, +6.5 points) and MAI-UI-2B (.596 vs. .556, +4.0 points). The larger gains on this subset support the interpretation that functional continuation is particularly useful when goal distance cannot distinguish states that require different next decisions.

4.4 WHAT DRIVES FUNCTIONAL RETRIEVAL?

Supervision relation. We isolate the effect of the supervision target using a strict shared-pair control. All variants receive identical state-pair instances and serialized batch schedules and use the same screenshot–goal encoder, metric architecture, loss formulation, optimization, and training budget; only the supervision relation changes.

In addition to visual similarity and goal distance, we consider an *Abstract-Q* relation based on the set of abstract graph-optimal next actions, $\mathcal{A}^*_\phi(s, g) = \{\phi(e) : e \in E^*_g(s)\}$. Two states are Abstract-Q equivalent when $\mathcal{A}^*_\phi(s_i, g) = \mathcal{A}^*_\phi(s_j, g)$. This captures immediate optimal-action compatibility but ignores the continuation structure reached after those actions; it therefore corresponds to the depth-1 version of our continuation abstraction.

Table 1b shows that functional continuation provides the strongest retrieval under this control. Visual supervision reaches .333±.007 NEXTACTION@1, while goal-distance supervision improves to .446±.005. Abstract-Q supervision reaches .409±.003, whereas full recursive functional continua-

tion achieves $.486 \pm .004$, improving over goal distance by $+.040$ and over Abstract-Q by $+.077$. Functional supervision also achieves the highest CONFLICT NA (.361), GOALFLIP (.585), and EQUIV@1 (.569). Thus, its advantage persists when training examples and optimization are controlled: goal distance captures coarse progress and Abstract-Q captures immediate decision compatibility, whereas recursive continuation additionally distinguishes the goal-directed futures reached by those decisions. Qualitative examples of this distinction are provided in Appendix K.2, Fig. 7. Further details of the shared pair control and supervision relations are provided in Appendix J.1.

Is recursive continuation necessary? We next ask whether the benefit of functional supervision can be explained by matching only the immediately optimal actions. We truncate the continuation teacher at depth h. At $h = 1$, two states are equivalent when they expose the same set of abstract optimal next actions. Larger values of h additionally capture the states reached by those actions and their subsequent continuations.

Figure 3c shows a clear benefit from this recursive structure. NEXTACTION@1 increases from .436 at $h = 1$ to .486 at $h = 2$ and .511 at $h = 3$, approaching .517 with full recursion. Full recursion improves over $h = 1$ by $+.080$ (95% CI $[+.061, +.099]$), whereas the difference between $h = 3$ and full recursion is only $+.005$ (95% CI $[-.007, +.017]$) (Table 13). Thus, matching immediate optimal actions alone does not explain the full benefit of functional supervision. Most of the additional information is captured within the first few continuation levels, while the full construction avoids choosing a fixed horizon. Full depth results are provided in Appendix J.2.

Sensitivity to the transition abstraction ϕ. We next test whether GCFSim depends on a carefully chosen transition taxonomy. Collapsing all transitions into a single class reduces NEXTACTION@1 from .517 to .465, whereas a semantic abstraction with only $K = 2$ classes reaches .513. Performance remains relatively stable across other abstraction granularities, indicating that distinguishing alternative continuations is important, while the precise taxonomy is less critical. Full results are provided in Appendix C.1.

Robustness to imperfect exploration graphs. We perturb only the graph used to construct training supervision while keeping test queries, candidate memories, and evaluation ground truth fixed. Figure 3d shows that GCFSim is robust to moderate goal annotation noise: with 20% corruption, NEXTACTION@1 remains .510, retaining 98.7% of clean performance. Missing transitions have a larger effect; removing 20% of training edges reduces performance to .439, retaining 84.9%. This indicates that incomplete exploration is more consequential than noisy semantic annotations. Full results are provided in Appendix C.2.

4.5 WHERE DO THE REMAINING RETRIEVAL ERRORS COME FROM?

We finally examine whether the remaining retrieval errors arise from insufficient application memory or imperfect ranking. Transferable optimal guidance exists somewhere in the available memory for .904 of queries, while NAORACLE@3 reaches .675, compared with GCFSim's NEXTACTION@1 of .517. Moreover, functional ranking errors account for 70.6% of the remaining top 1 failures. Thus, the main remaining bottleneck is ranking useful states near the top of the candidate memory rather than insufficient memory coverage. Full top K analysis and failure decomposition are provided in Appendix L.

5 CONCLUSION AND LIMITATIONS

We introduced GCFSim, a goal-conditioned approach to retrieving useful application memory for GUI navigation. Using graph-derived functional continuation rather than visual similarity or goal progress, GCFSim improves retrieval across 49 mobile applications and consistently improves first-action selection for frozen GUI agents.

GCFSim assumes prior exploration of the target application, so performance depends on graph coverage and may degrade under application drift. Its functional teacher also depends on automatically generated semantic goal targets and the transition abstraction ϕ, which can introduce noise or omit fine-grained distinctions. Our downstream evaluation focuses on the immediate navigation decision supported by retrieved memory rather than complete task execution.

Ethics Statement

Our corpus is constructed through automated exploration of mobile applications on researcher-controlled devices. Because observed screens may contain third-party or potentially identifying information, we apply an anonymization pipeline before public release that removes or obscures personal identifiers from both screenshots and associated graph metadata. This includes detected faces, names, usernames, contact information, credentials, and other potentially identifying content. Details of the anonymization procedure are provided in Appendix A. The released corpus is intended for research on GUI navigation and application-memory retrieval, not for identifying or profiling individuals.

Reproducibility Statement

We provide an anonymous implementation and dataset for review, with public release planned upon acceptance. The appendix documents application exploration and graph construction, semantic goal construction, training objectives and hyperparameters, evaluation metrics, baseline implementations, and downstream agent evaluation. We additionally provide complete proofs, per-application corpus statistics, annotation-validation results, and robustness and ablation experiments. Together with the released code and data, these materials specify the components required to reproduce the reported experiments.

AI Use Disclosure

AI-based tools were used during manuscript preparation to aid and polish writing, including improving wording, clarity, organization, and presentation. They were also used for retrieval and discovery purposes, including identifying and locating potentially relevant related work. In addition, AI tools were used to assist in drafting and refining proofs of the theoretical propositions. All cited literature was checked by the authors, and all proof steps, mathematical arguments, and resulting claims were independently reviewed and verified by the authors. AI tools were not used to determine the scientific conclusions of the work. All AI-assisted suggestions and generated content were reviewed and validated by the authors.

References

Maroun Ayli, Youssef Bakouny, Tushar Sharma, Nader Jalloul, Hani Seifeddine, and Rima Kilany. UISearch: Graph-based embeddings for multimodal enterprise ui screenshots retrieval. *arXiv preprint arXiv:2511.19380*, 2025.

Yuxiang Chai, Han Xiao, Xinyu Fu, Jinpeng Chen, Rui Liu, and Hongsheng Li. UI-KOBE: Knowledge-oriented behavior exploration for lightweight graph-guided gui agents. *arXiv preprint arXiv:2605.29534*, 2026.

Weizhi Chen, Ziwei Wang, Leyang Yang, Sheng Zhou, Xiaoxuan Tang, Jiajun Bu, Yong Li, and Wei Jiang. PG-Agent: An agent powered by page graph. In *Proceedings of the 33rd ACM International Conference on Multimedia*, pp. 6878–6887. Association for Computing Machinery, 2025. doi: 10.1145/3746027.3755189.

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

Biplab Deka, Zifeng Huang, Chad Franzen, Joshua Hibschman, Daniel Afergan, Yang Li, Jeffrey Nichols, and Ranjitha Kumar. Rico: A mobile app dataset for building data-driven design applications. In *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology*, UIST '17, pp. 845–854, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450349819. doi: 10.1145/3126594.3126651. URL https://doi.org/10.1145/3126594.3126651.

Benjamin Eysenbach, Tianjun Zhang, Sergey Levine, and Ruslan Salakhutdinov. Contrastive learning as goal-conditioned reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 35, pp. 35603–35620, 2022.

Boyu Gou, Ruohan Wang, Boyuan Zheng, Yanan Xie, Cheng Chang, Yiheng Shu, Huan Sun, and Yu Su. Navigating the digital world as humans do: Universal visual grounding for GUI agents. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=kxnoqaisCT.

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. 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*, pp. 5396–5405. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.emnlp-main.274.

Philippe Hansen-Estruch, Amy Zhang, Ashvin Nair, Patrick Yin, and Sergey Levine. Bisimulation makes analogies in goal-conditioned reinforcement learning. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 8407–8426. PMLR, 2022.

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

Toby Jia-Jun Li, Lindsay Popowski, Tom M. Mitchell, and Brad A. Myers. Screen2Vec: Semantic embedding of gui screens and gui components. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, pp. 1–15. Association for Computing Machinery, 2021. doi: 10.1145/3411764.3445049.

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

Chen Liu, Ling Chen, Hanzhang Zhou, Xu Zhang, Quyu Kong, Panrong Tong, Wenhao Wang, Xin Yu, Steven Hoi, and Yue Wang. What memory do gui agents really need? from passive records to active task-driving states. *arXiv preprint arXiv:2606.31612*, 2026.

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

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, Xiaojun Xiao, Kai Cai, Chuang Li, Yaowei Zheng, Chaolin Jin, Chen Li, Xiao Zhou, Minchao Wang, Haoli Chen, Zhaojian Li, Haihua Yang, Haifeng Liu, Feng Lin, Tao Peng, Xin Liu, and Guang Shi. Ui-tars: Pioneering automated gui interaction with native agents. *arXiv preprint arXiv:2501.12326*, 2025.

Zerui Qin, Sheng Yue, Xingyuan Hua, Yongjian Fu, and Ju Ren. Executable agentic memory for gui agent. In *International Conference on Machine Learning*, 2026.

Christopher Rawles, Alice Li, Daniel Rodriguez, Oriana Riva, and Timothy Lillicrap. Android in the wild: A large-scale dataset for android device control. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

Tom Schaul, Daniel Horgan, Karol Gregor, and David Silver. Universal value function approximators. In *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pp. 1312–1320. PMLR, 2015.

Yuchen Sun, Shanhui Zhao, Tao Yu, Hao Wen, Samith Va, Mengwei Xu, Yuanchun Li, and Chongyang Zhang. GUI-Xplore: Empowering generalizable gui agents with one exploration. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 19477–19486, 2025. doi: 10.1109/CVPR52734.2025.01814.

Junyang Wang, Haiyang Xu, Haitao Jia, Xi Zhang, Ming Yan, Weizhou Shen, Ji Zhang, Fei Huang, and Jitao Sang. Mobile-agent-v2: Mobile device operation assistant with effective navigation via multi-agent collaboration. In *Advances in Neural Information Processing Systems*, volume 37, pp. 2686–2710, 2024. doi: 10.52202/079017-0088.

Jialiang Wei, Anne-Lise Courbis, Thomas Lambolais, Binbin Xu, Pierre Louis Bernard, Gérard Dray, and Walid Maalej. GUing: A mobile gui search engine using a vision-language model. *ACM Transactions on Software Engineering and Methodology*, 34(4):1–30, 2025. doi: 10.1145/3702993.

Hao Wen, Yuanchun Li, Guohong Liu, Shanhui Zhao, Tao Yu, Toby Jia-Jun Li, Shiqi Jiang, Yunhao Liu, Yaqin Zhang, and Yunxin Liu. Autodroid: Llm-powered task automation in android. In *Proceedings of the 30th Annual International Conference on Mobile Computing and Networking*, pp. 543–557. Association for Computing Machinery, 2024. doi: 10.1145/3636534.3649379.

Bin Xie, Rui Shao, Gongwei Chen, Kaiwen Zhou, Yinchuan Li, Jie Liu, Min Zhang, and Liqiang Nie. 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)*, pp. 5650–5667. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.acl-long.282.

Mingxian Yu, Siqi Luo, and Xu Chen. Graphpilot: Gui task automation with one-step llm reasoning powered by knowledge graph. *Journal of Intelligent Computing and Networking*, 2(1):13–21, 2026. doi: 10.64509/jicn.21.45.

Amy Zhang, Rowan McAllister, Roberto Calandra, Yarin Gal, and Sergey Levine. Learning invariant representations for reinforcement learning without reconstruction. In *International Conference on Learning Representations*, 2021.

Chi Zhang, Zhao Yang, Jiaxuan Liu, Yanda Li, Yucheng Han, Xin Chen, Zebiao Huang, Bin Fu, and Gang Yu. Appagent: Multimodal agents as smartphone users. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems*, pp. 70:1–70:20. Association for Computing Machinery, 2025. doi: 10.1145/3706598.3713600.

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

Sibo Zhu, Wenyi Wu, Kun Zhou, Stephen Wang, and Biwei Huang. Hybrid self-evolving structured memory for gui agents. *ArXiv*, abs/2603.10291, 2026. URL `https://api.semanticscholar.org/CorpusID:286457424`.

A Application Exploration and Graph Construction

We construct an application-specific directed interaction graph $G_a = (V_a, E_a)$ by automatically exploring each mobile application on physical Android or HarmonyOS devices. Nodes correspond to observed UI states, and directed edges correspond to interactions actually executed on the device. Importantly, we separate *graph observation* from *semantic post-processing*: screenshots, UI hierarchies, executed actions, and graph connectivity are collected during exploration, whereas the natural-language descriptions and user intents used by the functional teacher are generated afterward from the observed graph.

A.1 Automated Application Exploration

Exploration episodes. Each exploration episode begins by resetting the application to a stable canonical start state. The reset procedure force-stops and relaunches the configured application, optionally scrolls toward the top of the initial page, and invokes a GUI agent with an application-specific reset instruction when additional navigation is required. Exploration is restricted to the configured application package; an episode is terminated when the foreground application leaves that package.

We combine three complementary exploration strategies. **Random walk** executes available UI interactions for a bounded number of steps and primarily discovers new trajectories. **Local BFS** first reaches an anchor state and then explores multiple previously untried outgoing interactions, returning to the anchor between branches. **Model-guided BFS** prioritizes previously observed states that retain unexplored interactions, navigates back to a selected state, and applies the same local branching procedure. The exploration schedule progressively permits deeper trajectories as graph construction proceeds.

Interactive-element discovery. For a newly observed screenshot, a multimodal VLM extracts actionable elements using the screenshot as its primary input. The prompt first identifies the single topmost interactive surface and excludes controls belonging to inactive background layers when a dialog, sheet, popup, menu, drawer, or similar overlay is active. It then extracts stable application-level navigation controls, including tabs, menus, settings entries, and other actionable regions, together with their bounding boxes and short action descriptions.

The extraction suppresses dynamic or undesirable interaction targets such as advertisements, repeated content items, personal content, temporary values, destructive actions, and background controls beneath active overlays. Scroll and horizontal-swipe actions may additionally be synthesized from the spatial arrangement of detected controls. Each detected element is stored with its action type, interaction region, description, and visual features.

Before a previously stored tap-like action is executed on a revisited state, the live crop at the stored interaction region is compared with its discovery-time visual feature. Actions whose appearance has changed beyond the configured similarity threshold are not executed, reducing errors from dynamic UI content.

State identification. After each interaction, the resulting screenshot is matched against previously observed graph nodes through four progressively more semantic stages: (i) high-confidence visual similarity, (ii) joint visual and UI-structure similarity, (iii) consistency with the action-description sequence from a graph root, and (iv) similarity of the screen's navigational purpose.

For the final stage, a VLM extracts a compact screen identity consisting of the active top-level tab, active subtab or filter when applicable, and a one-sentence functional *page purpose*. The screenshot is treated as the primary evidence; recent action history and known interactive actions are used only to disambiguate visually ambiguous states. When an overlay is present, the prompt treats the foreground overlay rather than the obscured page as the active state.

Candidate nodes are first filtered by page-purpose embedding similarity. A second text-only VLM then conservatively determines whether the query and a candidate represent the same navigational purpose, using page purpose as the primary signal and action paths as supporting context. If no candidate passes the configured criteria, the screenshot becomes a new graph node.

Observed transitions. Whenever an executed interaction moves the application from state s to state s', we add a directed edge (s, s') containing the observed action type, interaction description, and region. Duplicate observations of the same source–target interaction are deduplicated. Thus, graph connectivity reflects device-observed transitions rather than VLM-predicted transitions.

Backtracking and exploration control. For BFS-style exploration, returning from a newly explored branch to its anchor uses two stages. First, a multimodal VLM compares the child and parent screenshots and predicts a single local inverse action, such as switching back to the previous navigation tab, reversing a scroll or swipe, collapsing an expanded menu, selecting the previous subtab, or using back navigation. The resulting state is verified visually and structurally against the intended parent. If local reversal fails, the application is reset and a known root-to-anchor graph path is replayed using the GUI agent.

For model-guided BFS, candidate states are ranked using their number and fraction of unexplored actions, graph depth, visit count, page description, and the semantic nature of remaining controls. The prompt favors states likely to expose new application structure while preferring shallower and less repeatedly visited states when exploration value is otherwise similar. These VLM calls affect graph coverage but do not directly generate the functional labels used to train GCFSim.

A.2 Semantic Post-processing

Once exploration is complete, we freeze the observed graph and run a separate semantic post-processing pipeline. No device actions or new graph transitions are introduced during this stage. The teacher-relevant pipeline consists of four semantic annotations: node descriptions, transition descriptions, path-level intents, and destination-level user intents.

Table 2 summarizes their provenance.

Table 2: **Semantic annotations used to construct goal-conditioned graph supervision.** Observed screenshots, actions, and topology are fixed before semantic post-processing.

Annotation	Inputs	Output
Page description	Node screenshot and exploration-time page summary	High-, medium-, and low-level description of the active UI surface
Transition description	Source/target descriptions, observed action metadata, interaction-region location, and optional action crops	Low-level visual grounding description and high-level functional action
Path intent	Alternating descriptions of states and observed transitions along a root-to-node shortest path	User-facing path goals and reliability score in $[0, 5]$
User intent	Destination-state description, selected path goals, and outgoing-transition descriptions	Natural-language navigation goals associated with the destination state

Node-level page descriptions. For each observed node, a multimodal VLM receives its screenshot and the weaker page summary produced during exploration. It generates three description levels: a high-level functional purpose, a medium-level description of major regions and page state, and a low-level description of stable visible content. The prompt restricts the description to the active foreground surface, avoids exhaustive element enumeration, and prohibits inference of navigation goals or outgoing transitions.

Abridged page-description prompt

Task. Describe only the visible *active UI surface*. If a modal, dialog, popup, bottom sheet, dropdown, picker, menu, or other overlay is open, treat that foreground surface as the page being described.

Output. Produce a high-level functional purpose, a medium-level description of the main regions and state, and a low-level description of stable visible labels and content.

Constraints. Do not infer navigation goals or outgoing transitions, and do not report coordinates, element identifiers, confidence values, or inactive background content.

Transition descriptions. For each directed source–target pair, all observed parallel interactions are processed together. The VLM receives the source and target page descriptions, observed edge metadata, interaction regions, and, when available, crops of the acted-on controls. Similar parallel actions may be grouped into a common action cluster.

Each cluster receives a low-level description for visual grounding and a high-level description of the generalized functional action. Bounding boxes are used only to infer coarse locations such as *top-right* or *bottom navigation*; raw coordinates are not included. Dynamic labels, personal names, prices, counts, and other transient attributes are explicitly discouraged.

Abridged transition-description prompt

Task. Describe each observed action between the source and target UI states.

Low level. Describe what control should be visually located, including its stable appearance, visible label or icon when applicable, component type, and rough screen location.

High level. Describe the generalized functional action performed by the interaction.

Constraints. Group visually and functionally equivalent parallel actions. Prefer stable descriptions over personal, product-specific, price-specific, or temporary content. Do not output raw coordinates or infer an unobserved transition.

Path-level navigation intents. To provide context for destination-state goal generation, we enumerate shortest paths from each graph root to each reachable node. For every path, the VLM receives the alternating sequence of node descriptions and observed transition descriptions and generates (i) concise user-facing goals supported by that path and (ii) a reliability score from 0 to 5.

The reliability prompt favors direct, stable, reusable routes and penalizes loops, unnecessary detours, repeated section switching, and interactions tied to dynamic or personalized content. Candidate paths are first ordered by this score. A second text-only VLM selects at most three paths per destination, favoring reliable paths that provide semantically or structurally diverse routes and avoiding near-duplicates that differ only through scrolling.

These path-level goals provide context for destination-intent generation; they are not treated as additional graph states or transitions.

Abridged path-intent prompt

Given one observed root-to-node navigation trajectory, generate concise user-facing goals describing why a user would follow the path and assign a reliability score from 0 to 5. Favor direct, stable, general paths aligned with the final state, and penalize loops, detours, or navigation that depends on personalized or temporary content.

A.3 Automatic User-Intent and Goal Generation

The goals used by the graph teacher are generated at the *destination-state* level. For every observed node v, a text-only VLM receives (i) its multi-level page description, (ii) goals associated with its selected root-to-v paths, and (iii) descriptions of its outgoing interactions.

The model produces concise commands that a user could naturally provide to a navigation agent, covering different levels of specificity from broad page purpose to specific capabilities available on

the current state. Crucially, the prompt excludes goals corresponding to outgoing transitions: if an interaction from v leads to another state, the capability of that successor should not be described as a goal satisfied by v. Dynamic, personalized, product-specific, price-specific, badge-specific, timestamp-specific, and other transient details are also excluded.

Abridged user-intent generation prompt

Task. Generate possible user intents for the *current screen only*. Write each intent as a natural command to a navigation agent, covering broad page-level, section-level, and specific current-screen capabilities.

Target constraint. Do not include goals corresponding to outgoing edges. If an action appears among the outgoing edges, assume that its destination goal belongs to another screen/state rather than the current one.

Content constraint. Exclude dynamic, personal, product-specific, price-specific, badge-specific, timestamp-specific, and other temporary details.

Each generated intent g is attached to the destination node v for which it was produced. Consequently, v defines an *exact target* of g in the graph teacher. The resulting goals are therefore automatically generated from observed UI states and their graph context; they are not independently human-authored task descriptions.

Given an intent g, we first define its exact target set T_g from its associated destination state(s). The semantic target expansion described in Appendix B may subsequently include other observed states whose associated goal descriptions indicate that they satisfy the same goal, yielding T_g^{sem}. The fixed observed graph and semantic target set T_g^{sem} then determine

$$D_g(s),\ E_g^*(s),\ C_g(s),$$

i.e., semantic goal distance, graph-optimal transitions, and functional continuation classes.

Accordingly, an "optimal" transition in our experiments means optimal *within the observed interaction graph*; it does not assert global optimality over application transitions that were never discovered during exploration.

Semantic post-processing uses structured JSON outputs. Node descriptions, transition descriptions, path intents, path selection, and user-intent generation use temperature 0 and seed 42; complete prompts and model configurations are provided with the released implementation.

A.4 Observed and Generated Information

The corpus preserves the distinction between physical observations and model-generated semantic metadata:

- **Observed on device:** screenshots, UI hierarchies, executed interactions, interaction regions, and graph connectivity.
- **Generated during exploration:** actionable-element descriptions, compact page-purpose information used for localization, local undo suggestions, and BFS exploration priorities.
- **Generated after exploration:** multi-level page descriptions, transition descriptions, path-level goals and reliability scores, and destination-level user intents.

Only the final group, together with the frozen observed topology, contributes semantic information to construction of the goal-conditioned functional teacher. Exploration-time VLM decisions influence graph coverage and localization but are not themselves continuation-class labels.

The post-processing pipeline additionally generates navigation plans and GUI-agent reasoning samples for separate downstream uses. These artifacts are *not* used to construct T_g^{sem}, D_g, E_g^*, or C_g, and are therefore not part of GCFSim's functional-supervision definition.

Figure 4 shows a representative explored application graph, illustrating the branching structure and diversity of UI states captured during offline exploration.

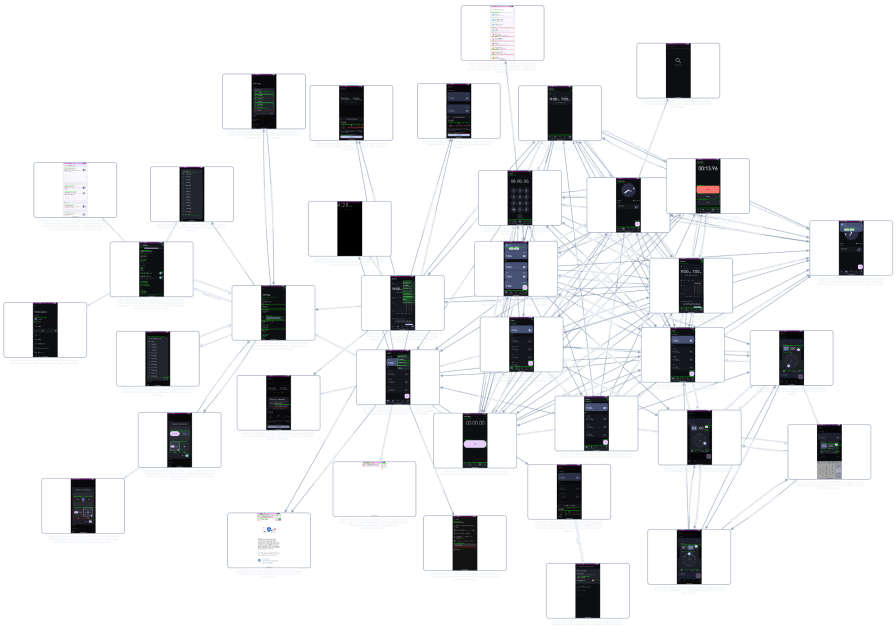


Figure 4: **Example explored application graph.** Representative interaction graph constructed for one application during offline exploration. Nodes correspond to observed UI states and display their associated screenshots, while directed edges correspond to interactions executed on the device. The dense branching and cross-links illustrate the scale and structural diversity of the application-specific memories used in our corpus.

A.5 Dataset Anonymization

Prior to public release, we apply an automated anonymization pipeline to the explored application graphs to remove potentially identifying information about third parties while preserving the visual and semantic structure required for navigation research. Anonymization is performed consistently across screenshots and their associated textual graph artifacts. For each observed screenshot, we detect faces using YuNet and extract visible text and corresponding image regions using EasyOCR. OCR text is screened using regular-expression rules for common personal identifiers, including names, usernames, email addresses, and phone numbers. A text VLM is additionally used to audit uncertain OCR fields and the low-level node and transition descriptions for potentially identifying content. Detected entities from these sources are consolidated into an application-level registry and assigned deterministic typed placeholders, such as `[PERSON_1]`, `[USERNAME_1]`, and `[EMAIL_1]`, so that repeated occurrences of the same entity are anonymized consistently across the graph. In the released screenshots, detected faces are blurred and text regions containing personal information are replaced with either type-preserving synthetic text, category labels (e.g., `NAME`, `EMAIL`, or `USERNAME`), or background-colored masks when a meaningful replacement is not appropriate. Registered entities are simultaneously replaced throughout the corresponding JSON artifacts, including node descriptions, transition descriptions, user intents, UI information, and other graph metadata. Learned embeddings are removed from the released artifacts, and credentials or API keys present in metadata are explicitly redacted. This procedure preserves observed graph connectivity and non-sensitive application content while preventing third-party identifying information from being propagated through either the visual or textual representation of a graph node.

A.6 Corpus Statistics

Table 3: **Per-application statistics of the 49-app navigation corpus.** UI states and transitions correspond to nodes and directed edges in the explored application graphs. Intents are unique automatically generated navigation goals associated with observed destination states.

Application	States	Trans.	Intents	Application	States	Trans.	Intents	Application	States	Trans.	Intents
AliExpress	214	846	1,042	Walmart	152	867	702	Google Play Store	133	790	588
Baidu (HarmonyOS)	348	1,680	1,710	Clock	37	255	150	Breaking News	17	33	71
Weibo (HarmonyOS)	335	1,343	1,674	Instagram	199	634	948	AutoScout24	59	462	257
Trip.com	310	987	1,589	OsmAnd	178	1,137	829	Books (HarmonyOS)	134	601	634
wikiHow	120	595	442	Yahoo Finance	99	685	470	Wallet (HarmonyOS)	95	313	406
AccuWeather	140	854	630	Temu	228	1,467	1,166	Bloomberg	137	1,116	627
LinkedIn	257	1,021	1,231	TikTok	317	1,469	1,486	Markor	62	377	278
Xiaohongshu (HarmonyOS)	249	1,281	1,201	Yelp	76	508	331	Chrome	149	635	705
YouTube	289	1,205	1,304	Spotify	119	802	547	Ctrip (HarmonyOS)	494	1,677	2,521
Alibaba (HarmonyOS)	199	935	961	Target	109	625	508	IMDb	98	491	423
SSENSE	107	1,008	319	Dongqiudi News (HarmonyOS)	339	1,728	1,271	Settings	139	434	628
Wish	141	987	646	DiDi (HarmonyOS)	157	836	711	Uber Eats	185	990	875
Amazon	218	1,078	1,042	Outlook	171	1,026	740	BBC Sport	126	807	531
AP News	115	638	498	Simple Calendar Pro	41	172	181	Game Center (HarmonyOS)	96	521	439
Huawei ToDo (HarmonyOS)	141	712	590	VLC (HarmonyOS)	46	363	182				
X	179	850	766	eBooks Reader	63	379	234	Home Workout	141	308	636
Babbel	44	231	174	Google Maps	101	395	457				
Total: 49 apps, 7,903 states, 39,154 transitions, 36,351 navigation intents											

B Semantic Goal Target Expansion

Each automatically generated user intent is initially associated with the observed destination state for which it was produced, yielding an exact target set T_g. However, semantically equivalent navigation goals may be generated with different surface forms for multiple states. Restricting goal completion to exact intent-string matches would therefore miss states that satisfy the same goal under a different description.

Let $\mathcal{G}(v)$ denote the set of generated goal descriptions associated with state v. We use a separate frozen semantic text encoder E_{sem} to compare goal descriptions and define

$$q(v, g) = \max_{g' \in \mathcal{G}(v)} \cos(E_{\text{sem}}(g), E_{\text{sem}}(g')) \,. \tag{3}$$

In our implementation, E_{sem} is the frozen `all-mpnet-base-v2` encoder used only for teacher construction; it is not an input to the learned GCFSim representation.

The exact target set is

$$T_g = \{v \in V_a : g \in \mathcal{G}(v)\} \,, \tag{4}$$

and we conservatively expand it as

$$T_g^{\text{sem}} = T_g \cup \{v \in V_a : q(v, g) \geq \tau_g\} \,. \tag{5}$$

Unless otherwise stated, $\tau_g = 0.95$.

This expansion is used only to identify alternative observed states whose associated goal descriptions strongly match g; semantic similarity between arbitrary UI states does *not* define functional equivalence. Once T_g^{sem} is fixed, goal distances, optimal graph transitions, and continuation classes are computed as in Sec. 3.1.

C Transition Abstraction Sensitivity and Graph Robustness

C.1 Transition Abstraction

The continuation construction associates each graph transition e with an abstract type $\phi(e)$. Our default mapping uses nine broad interaction intents: *scroll, dismiss, input, select, open, navigate, play, purchase,* and *share*, together with a residual *other* category.

We compare three abstraction families:

- **Manual:** the default nine interaction intents plus *other*;
- **Identity:** each transition description is retained as its own transition type;

- **Semantic:** transition descriptions are clustered automatically in a frozen semantic embedding space.

For semantic clustering, let $d(e)$ denote the transition description. Using the frozen MPNet semantic encoder introduced in Appendix B, we compute

$$h(e) = \frac{E_{\text{sem}}(d(e))}{\|E_{\text{sem}}(d(e))\|_2}, \qquad \phi_K(e) = \text{KMeans}_K(h(e)).$$

The clustering is independent of the user goal and uses no goal distances, target labels, continuation classes, or evaluation outcomes. For each K, we recompute the continuation classes and retrain GCFSim using the same data split, architecture, optimization procedure, and retrieval protocol.

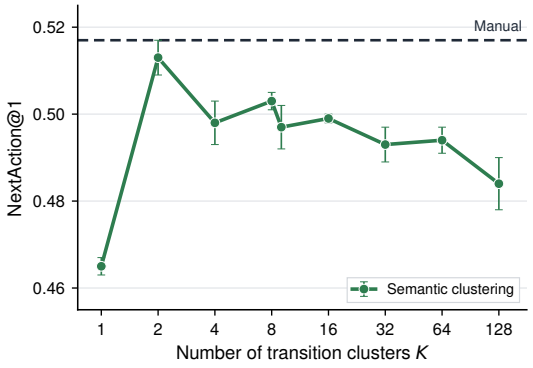


Figure 5: **Transition abstraction sensitivity.** Collapsing all transitions into one class reduces retrieval quality, while performance is relatively stable once multiple transition types are retained.

Table 4: **Sensitivity to transition abstraction ϕ.** Semantic variants cluster transition descriptions using a frozen semantic encoder. Results use application macro averaging and report mean±standard deviation across three seeds.

Abstraction	K	NA@1	Conflict NA	OptimalAction SemSim@1
Identity	N/A	.504±.006	.390±.009	.633±.003
Semantic	1	.465±.002	.346±.004	.614±.003
Semantic	2	.513±.004	.399±.002	.634±.001
Semantic	4	.498±.005	.386±.011	.639±.002
Semantic	8	.503±.002	.390±.008	.641±.002
Semantic	9	.497±.005	.388±.008	.636±.003
Semantic	16	.499±.001	.384±.005	.641±.002
Semantic	32	.493±.004	.384±.002	.642±.002
Semantic	64	.494±.003	.378±.002	.640±.001
Semantic	128	.484±.006	.368±.008	**.643±.003**
Manual	9 + other	**.517±.005**	**.401±.008**	.637±.004

As shown in Fig. 5, collapsing all transitions into a single class reduces NEXTACTION@1 from .517 to .465, indicating that preserving some distinction between alternative transitions is important. Once multiple transition types are retained, however, performance becomes much less sensitive to the abstraction granularity. Semantic clustering with only $K = 2$ classes reaches .513, within .004 of the default mapping, while the identity abstraction reaches .504. Across semantic abstractions from $K = 2$ to $K = 128$, there is no sharp optimum near the size of the manually defined vocabulary.

These results suggest that functional continuation benefits from distinguishing alternative transitions, but does not depend strongly on the precise semantics or granularity of the transition taxonomy.

C.2 Robustness to Imperfect Exploration Graphs

We perturb only the graph used to construct training supervision. Test queries, candidate memories, optimal action labels, and evaluation ground truth remain fixed to the clean graph.

We consider two forms of corruption:

- **Goal annotation noise:** a fraction ρ of the goal annotations used to construct the training teacher is corrupted;
- **Random edge removal:** a fraction ρ of graph transitions used for training is removed, simulating incomplete exploration.

For each corrupted graph, all affected graph quantities are recomputed before training. GCFSim and Goal Distance are trained using the same corruption level, while evaluation always uses the clean graph.

Table 5: **Robustness to imperfect graph supervision.** Evaluation uses the fixed clean test set. Retention is measured relative to clean GCFSim NextAction@1 (.517).

Teacher corruption	ρ	Goal Distance	GCFSim	Retention
Clean	0%	.463±.010	**.517±.005**	100.0%
Goal annotation noise	10%	.457±.003	**.512±.003**	99.1%
	20%	.454±.002	**.510±.001**	98.7%
	30%	.445±.003	**.503±.003**	97.4%
Random edge removal	10%	.436±.001	**.479±.005**	92.6%
	20%	.402±.001	**.439±.003**	84.9%
	30%	.390±.000	**.412±.002**	79.7%

Goal annotation noise has relatively little effect over the tested range. At 10%, 20%, and 30% corruption, GCFSim retains 99.1%, 98.7%, and 97.4% of its clean performance, respectively. Random edge removal has a larger effect because missing transitions alter reachability, shortest paths, and recursive continuation structure. After removing 10%, 20%, and 30% of the training edges, GCFSim retains 92.6%, 84.9%, and 79.7% of clean performance, respectively.

GCFSim remains above the corresponding Goal Distance baseline at every tested corruption level. The gap narrows as more edges are removed, indicating that functional continuation tolerates moderate annotation noise but benefits from sufficiently complete graph exploration.

D Proofs of Theoretical Results

D.1 Proof of Proposition 1

Proof. Consider two states s_i and s_j and two goal targets t_1 and t_2. Suppose that under goal g_1, whose target is t_1, both states have a graph-optimal transition of the same abstract type α directly to the target:

$$s_i \xrightarrow{\alpha} t_1, \qquad s_j \xrightarrow{\alpha} t_1.$$

Their continuation signatures are therefore identical,

$$\mathcal{N}_{g_1}(s_i) = \mathcal{N}_{g_1}(s_j) = \{(\alpha, 0)\},$$

and hence

$$s_i \sim_{g_1} s_j.$$

Now consider a second goal g_2 with target t_2. Suppose the corresponding optimal transitions are

$$s_i \xrightarrow{\alpha} t_2, \qquad s_j \xrightarrow{\beta} t_2, \qquad \alpha \neq \beta.$$

Then

$$\mathcal{N}_{g_2}(s_i) = \{(\alpha, 0)\}, \qquad \mathcal{N}_{g_2}(s_j) = \{(\beta, 0)\},$$

so

$$s_i \not\sim_{g_2} s_j.$$

Thus, whether the two states are functionally equivalent depends on the goal. A goal-independent representation therefore cannot in general preserve this equivalence relation for all goals. □

D.2 Proof of Proposition 2

Proof. Fix a goal g. Since every edge $e = (s, s') \in E_g^*$ satisfies

$$D_g(s') = D_g(s) - 1,$$

the optimal goal-directed subgraph G_g^* is acyclic when states are ordered by their distance to the target. Hence the recursive definition of C_g is well defined.

For any abstraction $\psi_g : V_g \to \mathcal{Z}$, define its abstract optimal-transition signature as

$$\Sigma_g^{\psi}(s) = \left\{ \big(\phi(e), \psi_g(s')\big) : e = (s, s') \in E_g^*(s) \right\}.$$

The plan-preservation condition can therefore be written as

$$\psi_g(s_i) = \psi_g(s_j) \Longrightarrow \Sigma_g^{\psi}(s_i) = \Sigma_g^{\psi}(s_j).$$

We first show that C_g is itself plan preserving. For non-terminal states,

$$\mathcal{N}_g(s) = \left\{ \big(\phi(e), C_g(s')\big) : e = (s, s') \in E_g^*(s) \right\}.$$

By construction, Φ_g assigns the same class identifier exactly to identical continuation signatures. Hence

$$C_g(s_i) = C_g(s_j) \Longleftrightarrow \mathcal{N}_g(s_i) = \mathcal{N}_g(s_j) \tag{6}$$

for non-terminal states, while all target states share terminal class 0. Thus states in the same C_g class expose identical abstract optimal transition types whose corresponding successors also belong to identical C_g classes. Therefore C_g is plan preserving.

It remains to show that every plan-preserving abstraction refines C_g. Let ψ_g be any plan-preserving abstraction. We prove by induction on

$$m = \max\{D_g(s_i), D_g(s_j)\}$$

that

$$\psi_g(s_i) = \psi_g(s_j) \Longrightarrow C_g(s_i) = C_g(s_j). \tag{7}$$

For $m = 0$, both states are target states and therefore $C_g(s_i) = C_g(s_j) = 0$.

Assume the claim holds for all pairs with maximum distance smaller than m, and consider a pair satisfying $\psi_g(s_i) = \psi_g(s_j)$ with $m > 0$. A target state has no remaining optimal transition, whereas every reachable non-terminal state has at least one edge in E_g^* along a shortest path. Plan preservation therefore cannot identify a target state with a non-terminal state.

For the non-terminal case, plan preservation gives

$$\Sigma_g^{\psi}(s_i) = \Sigma_g^{\psi}(s_j).$$

Hence every optimal transition $e_i = (s_i, s_i')$ has a corresponding transition $e_j = (s_j, s_j')$ such that

$$\phi(e_i) = \phi(e_j), \qquad \psi_g(s_i') = \psi_g(s_j'),$$

and conversely. Since both transitions are optimal,

$$D_g(s_i') = D_g(s_i) - 1, \qquad D_g(s_j') = D_g(s_j) - 1.$$

The successor pair therefore has maximum distance strictly smaller than m, so the induction hypothesis gives

$$C_g(s_i') = C_g(s_j').$$

Applying this argument to every corresponding optimal transition yields

$$\mathcal{N}_g(s_i) = \mathcal{N}_g(s_j),$$

and Eq. (6) gives $C_g(s_i) = C_g(s_j)$.

Thus every plan-preserving abstraction ψ_g refines the partition induced by C_g. Since C_g is itself plan preserving, it is the coarsest plan-preserving abstraction. □

E METRIC-LEARNING AND TRAINING DETAILS

E.1 TRAINING PAIR CONSTRUCTION

For each goal g, continuation classes induce binary pair supervision:

$$y_{ij}^g = \mathbb{1}[C_g(s_i) = C_g(s_j)]\,. \tag{8}$$

Pairs with $y_{ij}^g = 1$ are functionally equivalent positives, while pairs from different continuation classes are negatives.

To prevent training from being dominated by easy negatives, the fixed sampling mixture emphasizes several informative cases:

- **Same-distance negatives:**
$$D_g(s_i) = D_g(s_j), \qquad C_g(s_i) \neq C_g(s_j),$$
which separate states with identical remaining goal distance but different functional continuations.
- **Visual hard negatives:** visually similar state pairs under the frozen screenshot encoder satisfying $C_g(s_i) \neq C_g(s_j)$.
- **Goal-flip units:** state pairs whose functional relation changes with the goal, e.g.,
$$C_{g^+}(s_i) = C_{g^+}(s_j), \qquad C_{g^-}(s_i) \neq C_{g^-}(s_j).$$
- **Random negatives:** pairs from different continuation classes sampled without an additional hardness constraint.

For each goal, we sample approximately 400 training pairs using a fixed mixture of 30% positives, 25% same-distance negatives, 20% visual hard negatives, 15% goal-flip units, and 10% random negatives. These form the offline training set. During optimization, GCFSim additionally applies an online hard-negative term using difficult negatives identified from the current embedding space.

E.2 PAIRWISE FUNCTIONAL PREDICTION

In addition to optimizing the embedding geometry, we use a lightweight pairwise classification head. For

$$z_i = z_\theta(S_i, g), \qquad z_j = z_\theta(S_j, g),$$

the predicted probability of functional equivalence is

$$p_\theta(s_i, s_j, g) = \sigma\left(H_\theta\left[|z_i - z_j|, z_i \odot z_j\right]\right), \tag{9}$$

where H_θ is a trainable pairwise head and σ denotes the sigmoid.

E.3 TRAINING OBJECTIVE

The complete objective is

$$\mathcal{L} = \mathcal{L}_{\mathrm{BCE}} + \lambda_f \mathcal{L}_{\mathrm{flip}} + \lambda_s \mathcal{L}_{\mathrm{sup}} + \lambda_n \mathcal{L}_{\mathrm{NCE}} + \lambda_h \mathcal{L}_{\mathrm{hard}}. \tag{10}$$

The binary pair-classification term is

$$\mathcal{L}_{\mathrm{BCE}} = -y_{ij}^g \log p_\theta(s_i, s_j, g) - (1 - y_{ij}^g) \log\left(1 - p_\theta(s_i, s_j, g)\right). \tag{11}$$

The goal-flip term explicitly enforces goal dependence. For a pair equivalent under g^+ but non-equivalent under g^-, we use

$$\mathcal{L}_{\mathrm{flip}} = \max\left(0,\, m - \mathrm{sim}_{g^+}(s_i, s_j) + \mathrm{sim}_{g^-}(s_i, s_j)\right), \tag{12}$$

where

$$\mathrm{sim}_g(s_i, s_j) = \cos\left(z_\theta(S_i, g), z_\theta(S_j, g)\right).$$

The supervised contrastive term $\mathcal{L}_{\mathrm{sup}}$ brings states from the same continuation class together while separating different classes. The NT-Xent term $\mathcal{L}_{\mathrm{NCE}}$ provides additional in-batch contrastive regularization. Finally, $\mathcal{L}_{\mathrm{hard}}$ is an online hard-negative/hard-triplet term constructed from negatives that remain close in the current embedding space.

E.4 Optimization and Batch Construction

The image and text encoders remain frozen; only the goal-conditioned projection F_θ and auxiliary pairwise components are optimized. GCFSim uses AdamW with learning rate 10^{-3} and weight decay 10^{-4}. A single learning rate is applied to all trainable parameters, with no learning-rate schedule or gradual unfreezing.

Each epoch contains 25 mini-batches of size 128. A balanced sampler constructs each batch from

$$1 \text{ application} \times 4 \text{ goals} \times 32 \text{ pairs}.$$

Training runs for at most 20 epochs with early stopping on validation AUROC (patience 6).

The instantiated loss is

$$\mathcal{L} = 1.0\,\mathcal{L}_{\mathrm{BCE}} + 2.0\,\mathcal{L}_{\mathrm{flip}} + 1.0\,\mathcal{L}_{\mathrm{sup}} + 0.5\,\mathcal{L}_{\mathrm{NCE}} + 0.5\,\mathcal{L}_{\mathrm{hard}}. \tag{13}$$

The goal-flip and online hard-negative terms use margin 0.2, while the supervised contrastive and NT-Xent losses use temperature $\tau = 0.07$.

F Manual Audit of Semantic Annotations

Because the graph teacher depends on automatically generated semantic annotations, we manually audit components that can be judged independently of graph completeness. We sample 300 items across the application corpus: 100 page descriptions, 100 goal–target pairs, and 100 observed transition descriptions.

For page descriptions, the annotator judges whether the generated description accurately represents the visible purpose and content of the screen. For goal–target pairs, the annotator judges whether reaching the displayed state satisfies the stated user goal. For transition descriptions, the annotator judges whether the generated description accurately characterizes the observed before–action–after transition. The annotator is not shown graph-derived labels such as D_g, E_g^*, or C_g.

We deliberately do not ask the annotator to judge whether an action is globally optimal, since the explored graph may omit valid application transitions.

Table 6 shows high manual-validity rates across all three semantic components. One of the 100 sampled goal–target pairs was judged unclear and is therefore excluded from the corresponding validity denominator.

Table 6: **Manual audit of semantic annotations.** Human-valid denotes *Accurate* or *Partially accurate* for page and transition descriptions and *Yes* for goal–target satisfaction. Unclear judgments are excluded from the validity denominator. Confidence intervals are 95% application-bootstrap intervals over per-application rates.

Annotation	N	Human-valid (%)	95% CI
Page description	100	99.0	[97.0, 100.0]
Goal target	99	91.9	[87.0, 97.0]
Exact	60	95.0	–
Semantic-expanded	39	87.2	–
Transition description	100	96.0	[92.0, 99.0]

Exact destination targets are judged to satisfy their associated goal in 95.0% of cases, while states introduced only through semantic expansion are valid in 87.2% of cases. Thus, the automatically generated destination goals are highly consistent with their associated screens, while semantic expansion introduces a modest amount of additional label noise.

G Evaluation Metrics

Exact target evaluation. Functional continuation classes are constructed using the semantic target set T_g^{sem}, as described in Sec. 3.1. For action-level evaluation, however, we use the original target

set T_g associated with the generated goal. We define the exact target distance as

$$D_g^{\text{exact}}(s) = \min_{t \in T_g} d_{G_a}(s, t),$$

and the corresponding graph optimal outgoing edges as

$$E_g^{*,\text{exact}}(s) = \left\{(s, s') \in E_a : D_g^{\text{exact}}(s') = D_g^{\text{exact}}(s) - 1\right\}.$$

Thus, semantic target expansion is used to construct functional equivalence C_g, while action-level evaluation is tied to the original goal target.

Action transfer. For a retrieved state r, we first consider its exact-target optimal outgoing edges $E_g^{*,\text{exact}}(r)$. Let $\iota(e)$ denote the high-level action description associated with edge e, falling back to its available transition description, low-level action description, or action type when necessary.

For an edge $e = (r, s')$, the evaluator orders candidate optimal actions by

$$\left(D_g^{\text{exact}}(s'), \text{key}(e), \text{norm}(\iota(e))\right),$$

in ascending lexicographic order, where $\text{key}(e)$ is the stored edge key and $\text{norm}(\cdot)$ denotes normalized action text. The first edge under this ordering is denoted $e^\star = (r, s^\star)$, with action description $\iota^\star(r, g) = \iota(e^\star)$. Since all edges in $E_g^{*,\text{exact}}(r)$ reduce the exact target distance by one, remaining ties are resolved by edge key and normalized text.

We then transfer $\iota^\star(r, g)$ to the query state s_q by grounding it to an observed outgoing edge $(s_q, s') \in E_a$. Grounding is performed in the following order: (i) an exact match of normalized action text, or a match of action type and successor state; (ii) otherwise, the outgoing edge whose SigLIP action-text embedding has maximum cosine similarity to $\iota^\star(r, g)$, provided that the similarity is at least the validation-selected threshold $\tau_{\text{intent}} = 0.83$; and (iii) otherwise, an outgoing edge whose successor state is $s^\star$.

We denote the action associated with the resulting grounded edge by $\hat{a}(s_q \mid r, g)$. If no outgoing edge of s_q satisfies any of these grounding rules, $\hat{a}(s_q \mid r, g)$ is undefined.

Let $r^\star$ denote the top retrieved state for query s_q and goal g. The set of exact-target graph optimal next actions is

$$\mathcal{A}^*_{\text{exact}}(s_q, g) = \left\{a(s_q, s') : D_g^{\text{exact}}(s') = D_g^{\text{exact}}(s_q) - 1\right\}.$$

We define

$$\textsc{NextAction@1} = \mathbb{1}\left[\hat{a}(s_q \mid r^\star, g) \in \mathcal{A}^*_{\text{exact}}(s_q, g)\right].$$

If action transfer is undefined, NEXTACTION@1 is zero. When several optimal continuations exist, any corresponding action is considered correct.

For OPTIMALACTIONSEMSIM@1, let $u(e) = E_{\text{txt}}(h(e))$, where $h(e)$ is the semantic action description associated with edge e and E_{txt} is a frozen sentence encoder. The metric is

$$\textsc{OptimalActionSemSim@1} = \max_{\substack{e \in E_g^{*,\text{exact}}(s_q) \\ e' \in E_g^{*,\text{exact}}(r^\star)}} \cos(u(e), u(e')),$$

and measures the highest semantic similarity between exact-target optimal outgoing actions of the query and retrieved states.

Table 7 summarizes all retrieval metrics.

Table 7: **Retrieval evaluation metrics.** $r^\star$ denotes the top retrieved state, $E_g^{*,\text{exact}}(s)$ denotes the exact-target graph-optimal outgoing edges, and $\mathcal{A}^*_{\text{exact}}(s,g)$ denotes the corresponding optimal next actions.

Metric	Definition	Measures
NEXTACTION@1	$\mathbb{1}[\hat{a}(s_q \mid r^\star, g) \in \mathcal{A}^*_{\text{exact}}(s_q, g)]$	Whether the top retrieval provides an optimal next action for the query.
CONFLICT NA	NEXTACTION@1 on visual conflict queries	Whether retrieval recovers an optimal action when visual retrieval is misleading.
OPTIMALACTION SEMSIM@1	$\max_{\substack{e\in E_g^{*,\text{exact}}(s_q)\\ e'\in E_g^{*,\text{exact}}(r^\star)}} \cos(u(e), u(e'))$	Best semantic similarity between the optimal next actions of the query and retrieved state.
EQUIV@1	$\mathbb{1}[C_g(s_q) = C_g(r^\star)]$	Whether the retrieved state belongs to the same functional continuation class.
GOALFLIP	$\mathbb{1}[\text{sim}_\theta(s_i, s_j, g^+) > \text{sim}_\theta(s_i, s_j, g^-)]$	Whether similarity changes correctly when the user goal changes.

H RETRIEVAL AND BASELINE DETAILS

H.1 RETRIEVAL IMPLEMENTATION

At inference time, GCFSim requires only the current screenshot and natural-language user goal. For query screenshot S_q and goal g, we compute

$$z_q = z_\theta(S_q, g).$$

Each stored screenshot S_s from application a is represented under the same goal as $z_\theta(S_s, g)$. Retrieval is

$$\hat{s} = \arg\max_{s\in\mathcal{M}_a} \cos\left(z_\theta(S_q, g), z_\theta(S_s, g)\right), \tag{14}$$

where $\mathcal{M}_a$ denotes the candidate memory for application a.

During evaluation, $\mathcal{M}_a$ follows the leave-one-visual protocol from Sec. 4.1: the query node itself and near-duplicate screenshots are removed identically for all retrieval methods.

No graph traversal or graph-derived feature is required to compute the query representation or retrieval score. Graph topology, shortest-path distances, continuation classes, node identifiers, and semantic graph annotations are used to construct training supervision and offline evaluation labels, but are not inputs to GCFSim retrieval at inference.

H.2 DISTANCE AMBIGUITY ANALYSIS

For each application and goal, we consider pairs of reachable states (s_i, s_j) satisfying

$$D_g(s_i) = D_g(s_j).$$

Among these pairs, we measure the fraction that belong to different functional continuation classes:

$$\mathbb{1}[C_g(s_i) \neq C_g(s_j)].$$

The reported ambiguity rate is computed per application and then macro averaged across the 49 applications. In total, the analysis contains 1,030,979 equal distance state pairs. The resulting macro average is 72.8%.

H.3 RETRIEVAL BASELINES

All baselines use the same test queries and the same candidate memories under the leave one visual protocol.

Visual SigLIP. States are ranked by cosine similarity between frozen SigLIP screenshot embeddings. This baseline uses visual appearance alone and does not condition retrieval on the user goal.

TextDense GoalOnly. Each candidate state is represented by a stored structured node document containing its page description, associated user intents, and available outgoing high-level action descriptions from the explored application graph. The query contains only the user goal text, with no screenshot or graph structure. Candidate documents are ranked by dense textual similarity to the goal. This provides a goal-aware semantic retrieval baseline without using visual information or navigation topology at query time.

Goal Distance. We train a goal conditioned metric using remaining shortest path distance as the supervision signal. For a goal g, two reachable states are treated as equivalent when

$$s_i \sim_g^D s_j \iff D_g(s_i) = D_g(s_j).$$

Thus, states with the same remaining number of graph steps form positive pairs, while states at different distances form negative pairs. The model learns this relation from screenshots and goals; retrieval uses cosine similarity in the learned embedding space, and D_g itself is not available at inference. This baseline tests whether goal progress alone provides sufficient structure for retrieval.

Visual + Distance. We test whether combining visual appearance with goal progress can recover the benefit of functional supervision. Visual and Goal Distance scores are independently standardized using the mean and standard deviation computed on the validation set, then combined as

$$s_\alpha = (1-\alpha)\widetilde{s}_D + \alpha\widetilde{s}_V.$$

The mixing coefficient is selected using validation NEXTACTION@1, yielding $\alpha = 0.025$. The resulting fusion is therefore strongly dominated by Goal Distance. On the test set, it reaches $.469\pm.007$ NEXTACTION@1, compared with $.463\pm.010$ for Goal Distance and $.517\pm.005$ for GCFSim. The paired difference between GCFSim and the fusion baseline is $+.048$ with a 95% application bootstrap CI of $[+.034, +.062]$.

As an additional fusion control, reciprocal rank fusion with $k = 60$ reaches $.448\pm.010$. Visual and Goal Distance retrieval nevertheless make different errors: visual retrieval fails while Goal Distance succeeds on 21.4% of queries, whereas the reverse occurs on 10.2%. Their combination therefore provides some complementary information, but closes only a small part of the gap to GCFSim.

GC Value. GC Value uses the same graph derived goal progress signal D_g as the Goal Distance baseline, but with a different learning objective. Goal Distance uses equality of remaining distance as pairwise supervision and directly optimizes a retrieval embedding. GC Value instead learns to predict a scalar goal value from the screenshot and goal. Its hidden representation is then used for retrieval, although its training objective does not directly constrain the retrieval geometry. This baseline therefore tests whether a representation learned for goal progress prediction is itself sufficient for useful state retrieval. For each reachable state, we construct the normalized graph value

$$V_g(s) = -\frac{D_g(s)}{D_g^{\max}},$$

where $D_g^{\max}$ is the maximum finite goal distance used for normalization. Given screenshot S and goal g, the model produces a goal conditioned hidden representation $z_V(S, g)$ followed by a scalar prediction head,

$$\widehat{V}(S, g) = H_V(z_V(S, g)).$$

The model is trained by Huber regression,

$$\mathcal{L}_V = \mathrm{Huber}\Big(\widehat{V}(S, g), V_g(s)\Big).$$

Thus, states closer to the goal receive larger target values, while states farther from the goal receive smaller values.

At retrieval time, we discard the scalar prediction head and use the normalized 256 dimensional hidden representation $z_V(S, g)$ as the retrieval embedding. Memory states are ranked by cosine similarity under the query goal. Therefore, GC Value tests whether a representation that successfully encodes remaining goal progress also organizes states in a way that is useful for transferring navigation guidance.

GC Abstract Q. GC Abstract Q provides a stronger baseline that learns which abstract actions are useful for the current state and goal. Let $\mathcal{A}$ be the same abstract action vocabulary used by the graph teacher (Manual-9 intents plus `other`), and define the set of optimal abstract actions as

$$\mathcal{A}^*_\phi(s,g) = \left\{\phi(e) : e \in E^*_g(s)\right\}.$$

For each abstract action $\alpha \in \mathcal{A}$, we construct a graph-derived action-value target $Q_g(s,\alpha)$ from the observed goal-directed graph. Let D_g denote the reverse goal distance and define

$$D_g^{\max} = \max_{u:\, D_g(u)<\infty} D_g(u).$$

For an abstract action α, let

$$d_g(s,\alpha) = \min_{\substack{(s,s')\in E_a \\ \phi(s,s')=\alpha \\ D_g(s')<\infty}} D_g(s').$$

Whenever $d_g(s,\alpha)$ is defined, we set

$$Q_g(s,\alpha) = -\frac{1 + d_g(s,\alpha)}{D_g^{\max}+1}.$$

i.e., the best one-step progress achievable by any observed transition of abstract type α, normalized by $D_g^{\max}+1$. Let $m_\alpha(s,g)$ indicate whether such a finite-distance successor of type α is available; targets are supervised only where $m_\alpha(s,g)=1$.

We additionally define the optimality label

$$y_\alpha(s,g) = \mathbb{1}\left[\alpha \in \mathcal{A}^*_\phi(s,g)\right].$$

Given screenshot S and goal g, the model computes a goal conditioned hidden representation $z_Q(S,g)$. Separate prediction heads produce an abstract action value vector and optimality scores,

$$\widehat{\mathbf{Q}}(S,g) = H_Q(z_Q(S,g)), \qquad \widehat{\mathbf{p}}(S,g) = H_{\text{opt}}(z_Q(S,g)).$$

Training combines masked Q regression with optimal action classification:

$$\mathcal{L}_Q = \frac{\sum_{\alpha\in\mathcal{A}} m_\alpha \ \text{Huber}\Big(\widehat{Q}_\alpha, Q_g(s,\alpha)\Big)}{\sum_{\alpha\in\mathcal{A}} m_\alpha} + 0.25\ \text{BCE}(\widehat{\mathbf{p}}, \mathbf{y}). \tag{15}$$

At retrieval time, neither the predicted Q values nor the optimality scores are used directly. We discard the prediction heads and rank memory states by cosine similarity between the normalized 256 dimensional hidden representations $z_Q(S,g)$. GC Abstract Q therefore tests whether a representation trained to encode immediate optimal action information is sufficient for retrieval.

This prediction baseline is distinct from the controlled *Abstract Q relation* in Table 1b. GC Abstract Q is trained using its native graph derived Q and optimality prediction objectives. In the controlled experiment, the relation

$$s_i \sim_g^Q s_j \iff \mathcal{A}^*_\phi(s_i,g) = \mathcal{A}^*_\phi(s_j,g)$$

instead provides pair labels within the common metric learning recipe. The former evaluates a standalone Q based representation, while the latter isolates the effect of immediate optimal action supervision under controlled training.

VisualGNN. VisualGNN provides a graph structured baseline that explicitly incorporates the explored application topology into the learned state representations. Each node is initialized as

$$\mathbf{h}_v^{(0)} = F_0([\mathbf{x}_v; \mathbf{x}_g; \mathbf{x}_v \odot \mathbf{x}_g]),$$

where $\mathbf{x}_v$ is the frozen SigLIP screenshot embedding of state v, $\mathbf{x}_g$ is the frozen SigLIP text embedding of the user goal, and F_0 is a lightweight fusion MLP followed by ℓ_2 normalization.

The node representations are then updated using a two layer GraphSAGE encoder with mean aggregation over the bidirectional, untyped adjacency of the observed application graph. Edge types and semantic transition descriptions are not provided to the GNN, and the goal representation is reintroduced at each message passing layer.

VisualGNN uses the same functional pair supervision and objective family as GCFSim:

$$\mathcal{L} = \mathcal{L}_{\mathrm{BCE}} + 2\,\mathcal{L}_{\mathrm{flip}} + \mathcal{L}_{\mathrm{sup}} + \tfrac{1}{2}\mathcal{L}_{\mathrm{NCE}} + \tfrac{1}{2}\mathcal{L}_{\mathrm{hard}},$$

with pair labels derived from C_g. Thus, differences between VisualGNN and GCFSim are not due to replacing the functional supervision relation with a different teacher.

At retrieval time, the query is encoded directly from its screenshot and goal without inserting the query into the graph, while candidate representations are obtained through goal conditioned GraphSAGE message passing over the stored application graph. Candidates are ranked by cosine similarity to the query representation. VisualGNN therefore requires the explored graph at inference, whereas GCFSim distills the graph derived supervision into a screenshot and goal representation and requires no graph message passing at query time.

Visual SigLIP Top 5 + Qwen 7B. SigLIP first retrieves the five nearest visual states. A 7B language model then reranks these candidates using the user goal and candidate information. This tests whether a more expensive semantic reranking stage can compensate for retrieval based initially on visual similarity.

GCFSim. GCFSim ranks the same candidate memory by cosine similarity in the learned goal conditioned functional embedding space $z_\theta(S, g)$.

H.4 Goal Conditioned Baseline Training and Sanity Checks

GC Value and GC Abstract Q use the same frozen encoders as GCFSim and comparable lightweight goal conditioned fusion networks. Their trainable parameter counts are 1.74M, 1.81M, and 1.87M for GC Value, GC Abstract Q, and GCFSim, respectively. A separate frozen MPNet encoder is used only when constructing semantic graph supervision and is not an input to any of these learned retrieval models.

Table 8 summarizes their training recipes. GC Value and GC Abstract Q retain the native prediction objectives associated with the representations they are intended to test, whereas GCFSim is trained directly to shape a retrieval space.

Table 8: **Training recipes for the goal conditioned prediction baselines and GCFSim.** The models use the same frozen encoders and comparable trainable capacity, while retaining objectives appropriate to each representation. Hyperparameters for GC Value and GC Abstract Q are selected using validation data only.

Detail	GC Value	GC Abstract Q	GCFSim
Trainable parameters	1.74M	1.81M	1.87M
Target	$V(s, g)$	abstract Q and optimality	functional relation
Objective	Huber$(V, -D/\max D)$	masked Huber$(Q) + 0.25\,\mathrm{BCE}$	$1\,\mathrm{BCE} + 2\,\mathrm{flip} + 1\,\mathrm{SupCon} + 0.5\,\mathrm{NT\text{-}Xent} + 0.5\,\mathrm{hard}$
Offline hard negative sampling	No	No	Yes
Online hard negative term	No	No	Yes, every optimization step
SupCon / NT Xent	No	No	Yes, $\tau = 0.07$
Optimizer	AdamW	AdamW	AdamW
Learning rate	$3{\times}10^{-4}$	$3{\times}10^{-4}$	10^{-3}
Weight decay	10^{-4}	0	10^{-4}
Batch size	128	128	128 ($1{\times}4{\times}32$ pairs)
Training budget	≤ 40 epochs	≤ 40 epochs	≤ 20 epochs $\times$ 25 steps
Early stopping	patience 6, validation loss	patience 6, validation loss	patience 6, validation AUROC
Supervision budget	≤ 60 states per goal	≤ 60 states per goal	~ 400 pairs per goal

Both prediction baselines successfully learn their intended targets. GC Value obtains Pearson correlation .610, Spearman correlation .593, MAE .127, and $R^2 = .358$ on held out value targets; a global mean predictor has MAE .169. GC Abstract Q identifies an optimal abstract action with .809 top 1 accuracy and achieves .815 recall over the optimal action set. Its macro F1 is .407, and the predicted masked Q values have Spearman correlation .575 with their targets.

Validation tuning improves NEXTACTION@1 from .333 to .362 for GC Value and from .354 to .423 for GC Abstract Q. The corresponding paired improvements are $+.029$ with 95% application bootstrap CI $[+.016, +.042]$ and $+.070$ with CI $[+.054, +.085]$, respectively. GCFSim reaches .517; relative to the tuned GC Abstract Q baseline, the paired difference is $+.091$ with 95% CI $[+.075, +.109]$.

These results show that the retrieval gap cannot be attributed simply to the value or Q predictors failing to learn their intended targets. However, this is not an identical recipe comparison: GC Value and GC Abstract Q optimize their native prediction objectives, whereas GCFSim directly learns a retrieval geometry. We therefore isolate the effect of the supervision relation separately under a common metric learning recipe in Appendix J.1.

H.5 RETRIEVAL EFFICIENCY

Table 9 reports parameter count and online localization latency after model warm-up, measured on an NVIDIA RTX 5080. Application-memory representations that can be constructed offline are excluded from the timer, including frozen SigLIP image embeddings and TextDense node-document MPNet embeddings. The reported cost includes the runtime processing required for a new user goal, any goal-conditioned memory projection or graph message passing required by the method, and nearest-neighbor search over the leave-one-visual candidate memory. Parameter counts include frozen encoders required at inference.

Table 9: **Retrieval efficiency.** Online localization latency after model warm-up. Reusable candidate representations are precomputed offline. For TextDense GoalOnly, the reported latency includes MPNet encoding of a previously unseen goal.

Method	**Latency (ms)** $\downarrow$	**Params. (M)** $\downarrow$
Visual SigLIP	26.0	203
TextDense GoalOnly	3.0	109
Goal Distance	29.6	205
GC Value	29.6	205
GC Abstract Q	29.6	205
VisualGNN	30.6	205
Visual SigLIP Top 5 + Qwen 7B	420	8500
GCFSim (ours)	29.6	205

For TextDense GoalOnly, candidate node-document embeddings are constructed offline. Processing a previously unseen goal requires one MPNet forward pass, which takes approximately 3 ms, followed by a negligible cosine search over the cached document bank. During the evaluation workload, many queries share the same goal, so caching the goal embedding reduces subsequent localization calls for that goal to approximately 0.04 ms. We report the new-goal cost in Table 9 to avoid attributing this cache reuse to the underlying retrieval model.

For VisualGNN, the approximately 30 ms cost is dominated by SigLIP query encoding, while GraphSAGE adds only a lightweight per-query message-passing step. GCFSim requires no graph traversal or additional language-model call at inference and remains comparable in cost to the other lightweight learned retrieval methods, while the visual-plus-LLM reranking baseline is substantially more expensive.

I FROZEN AGENT GUIDANCE EVALUATION

I.1 EVALUATION PROTOCOL

For each query state q and user goal g, the retrieval method selects a memory state from the same application. Up to three goal directed transitions associated with the retrieved state are inserted into a fixed guidance prompt. The downstream agent then predicts its first action on the original query screen.

Agent parameters, decoding settings, query screenshots, user goals, and prompt templates are identical across retrieval conditions; only the retrieved memory changes. The agents are used as frozen policies and are not fine tuned on GCFSim outputs.

We compare five memory conditions: *No Memory*, *Visual*, *Distance*, *GCFSim*, and *Oracle*. No Memory provides no retrieved application guidance. Visual and Distance use the corresponding retrieval baselines, while GCFSim uses the learned functional representation. Oracle is included as a ground truth informed diagnostic of memory selection.

Oracle memory. Oracle uses the same leave-one-visual candidate memory as all other retrieval methods, but selects a memory state using graph information unavailable to the learned retrievers. For each query (q, g), candidates are considered using the following priority rule:

1. If one or more candidates belong to the same functional continuation class as the query,
$$C_g(r) = C_g(q),$$
Oracle selects the candidate with the lexicographically smallest node identifier. Here C_g is constructed using semantic goal targets.
2. Otherwise, each candidate is evaluated using the same exact-target action-transfer procedure as NEXTACTION@1. Candidates are retained when their transferred action is exact-target optimal for the query,
$$\hat{a}(q \mid r, g) \in \mathcal{A}^*_{\text{exact}}(q, g).$$
If several candidates satisfy this condition, Oracle selects the one with the lexicographically smallest node identifier.
3. If neither condition is satisfied, Oracle selects the lexicographically first candidate with finite positive D_g^{exact} and at least one outgoing edge in $E_g^{*,\text{exact}}$. If no such candidate exists, Oracle selects the lexicographically first candidate in the available memory.

The selected state's guidance is then inserted into exactly the same prompt used for the other memory conditions, and the frozen agent predicts its action with identical decoding settings. Oracle is therefore a ground truth informed diagnostic of memory selection quality rather than an oracle policy: the downstream agent must still interpret the retrieved guidance and ground a valid action on the query screen.

This Oracle condition is distinct from the .904 memory coverage rate in Appendix L. Memory coverage asks only whether some candidate in the full memory contains transferable optimal guidance. Oracle instead selects a single candidate according to the deterministic rule above, after which the frozen agent must still convert the retrieved guidance into a correct action.

Seven frozen GUI agents spanning different architectures and model scales are evaluated. MAI-UI-8B uses the deployed checkpoint with its LLM component quantized to 4 bit AWQ.

I.2 FIRST ACTION EVALUATION

For each query, we evaluate whether the agent's first predicted action is a valid graph optimal next action for the current state and goal. Let a denote the action predicted by the frozen agent on query state q. We ground a to an observed outgoing graph edge of q using UI bounding box matching.

Let
$$\mathcal{S}^*_{g,\text{exact}}(q) = \left\{ s' : (q, s') \in E_g^{*,\text{exact}}(q) \right\}$$
denote the exact-target graph-optimal successor states. If the grounded action corresponds to an edge (q, s_a), we define
$$\text{FIRSTACTIONSUCCESS}(q, g, a) = \mathbb{1}\left[s_a \in \mathcal{S}^*_{g,\text{exact}}(q) \right].$$

Thus, a prediction is successful when its grounded action reaches any optimal successor of the current query state. This also accepts distinct grounded actions that lead to the same optimal successor. When multiple optimal successors exist, any of them is considered correct.

This metric is distinct from functional equivalence C_g. Functional continuation classes are used to train GCFSim, whereas downstream agent scoring depends only on whether the predicted action makes immediate optimal progress according to D_g^{exact}. The evaluation therefore does not directly score agreement with GCFSim's semantic continuation-class labels.

For an application A with evaluation queries $\mathcal{Q}_A$, we report its success rate as

$$\text{SR}_A = \frac{1}{|\mathcal{Q}_A|} \sum_{(q,g)\in\mathcal{Q}_A} \textsc{FirstActionSuccess}(q, g, a_{q,g}),$$

where $a_{q,g}$ is the first action predicted by the agent. The reported *Success Rate* is the macro average of SR_A across applications. This is the quantity shown in Fig. 3a and Table 10.

I.3 Full Per Agent Results

Table 10: **Effect of retrieved application memory on frozen GUI agents.** Values are application macro Success Rate, obtained by averaging FIRSTACTIONSUCCESS within each application and then across applications. GCFSim provides the strongest non oracle memory for all seven agents.

Agent	No Memory	Visual	Distance	GCFSim	Oracle
MAI-UI-8B (4 bit AWQ LLM)	.370	.462	.482	**.520**	.538
MAI-UI-2B	.391	.477	.509	**.533**	.574
UI-TARS-1.5-7B	.327	.381	.407	**.422**	.430
UI-TARS-2B-SFT	.161	.234	.238	**.245**	.263
GUI-Owl-7B	.323	.360	.387	**.397**	.407
Qwen2.5-VL-7B	.235	.296	.316	**.333**	.358
Jedi-3B	.321	.331	.329	**.335**	.342

GCFSim provides the strongest non oracle memory for every evaluated agent, despite substantial differences in their absolute success rates. The gain is also consistent across model scales and agent families, suggesting that the retrieved functional guidance is useful independently of a particular downstream architecture.

I.4 Distance Ambiguous Queries

To isolate cases in which goal distance cannot uniquely identify useful navigation guidance, we define a *distance ambiguous* subset. A query (q, g) belongs to this subset when the remaining candidate memory contains states at the same goal distance that belong to different functional continuation classes, and at least one candidate with $C_g(s) = C_g(q)$ remains available. Thus, distance alone treats multiple functionally different states as equally progressed toward the goal, even though a functionally compatible retrieval is possible.

The subset is constructed from the application graph after applying the same candidate memory filtering used in the main evaluation. It does not depend on the state retrieved by GCFSim or any other method.

Table 11: **Frozen agent performance on distance ambiguous queries.** Values are application macro Success Rate on the distance ambiguous subset. Δ denotes the improvement of GCFSim over Goal Distance.

Agent	Goal Distance	GCFSim	Δ
MAI-UI-8B (4 bit AWQ LLM)	.526	**.591**	+.065
MAI-UI-2B	.556	**.596**	+.040
UI-TARS-1.5-7B	.434	**.459**	+.025
UI-TARS-2B-SFT	.260	**.266**	+.006
GUI-Owl-7B	.415	**.427**	+.012
Qwen2.5-VL-7B	.354	**.369**	+.015
Jedi-3B	.325	**.338**	+.013

GCFSim improves Success Rate over Goal Distance for all seven agents. The largest gains are observed for MAI-UI-8B ($+6.5$ points) and MAI-UI-2B ($+4.0$ points). These results provide a targeted test of the limitation of distance based retrieval: when states with identical goal progress correspond to different useful decisions, recursive functional information provides additional guidance.

J Additional Supervision Ablations

J.1 Controlled Supervision Relations

The controlled experiment in Table 1b isolates the effect of the relation used to supervise state similarity. All variants receive exactly the same state pair instances in the same serialized batches. They also use the same screenshot and goal encoder, metric architecture, loss formulation, optimization settings, training budget, and evaluation candidate memories. Only the supervision relation and the labels induced by that relation change.

For visual supervision, let

$$v(s) = \frac{E_I(S_s)}{\|E_I(S_s)\|_2}$$

denote the normalized frozen SigLIP embedding of the screenshot associated with state s. We define visual equivalence by

$$s_i \sim^V s_j \iff v(s_i)^\top v(s_j) \geq \tau_V,$$

where $\tau_V = 0.85$ is the fixed visual-similarity threshold used to construct the binary pair labels. This relation depends only on screenshot appearance and is independent of the user goal.

For goal distance, two states are considered equivalent when they have the same remaining distance to the goal:

$$s_i \sim_g^D s_j \iff D_g(s_i) = D_g(s_j).$$

For Abstract Q, equivalence is determined by the set of graph optimal abstract actions,

$$\mathcal{A}^*_\phi(s, g) = \{\phi(e) : e \in E^*_g(s)\},$$

such that

$$s_i \sim_g^Q s_j \iff \mathcal{A}^*_\phi(s_i, g) = \mathcal{A}^*_\phi(s_j, g).$$

This relation captures which abstract actions are immediately optimal, but does not distinguish the continuation reached after taking those actions.

Functional supervision instead uses the recursive continuation classes:

$$s_i \sim_g s_j \iff C_g(s_i) = C_g(s_j).$$

It therefore preserves both the abstract optimal actions available at a state and the functional structure of the states reached by those actions.

Because the underlying pair instances and batch order are fixed across all four variants, differences in retrieval performance cannot be explained by exposure to different training examples or optimization schedules. Relation-dependent labels may change which pairs contribute as positives, negatives, goal-flip constraints, or hard negatives, but the overall loss formulation remains unchanged.

J.2 Recursive Continuation Depth

To measure how much of the benefit comes from looking beyond the immediately optimal actions, we construct truncated continuation teachers with $h \in \{1, 2, 3, 4, 5\}$, together with the full recursive teacher. All depth variants use the same data split, transition abstraction ϕ, model architecture, optimization procedure, and retrieval protocol.

At depth $h = 1$, the continuation signature contains only the abstract optimal actions:

$$\mathcal{N}^{(1)}_g(s) = \{\phi(e) : e \in E^*_g(s)\}.$$

States with identical signatures receive the same class $C^{(1)}_g$. Thus, $h = 1$ captures immediate optimal action compatibility but does not consider where those actions lead.

For $h > 1$, we recursively include the continuation class of each optimal successor:

$$\mathcal{N}_g^{(h)}(s) = \left\{ \left(\phi(e), C_g^{(h-1)}(s')\right) : e = (s, s') \in E_g^*(s) \right\}, \qquad C_g^{(h)}(s) = \Phi_g^{(h)}\left(\mathcal{N}_g^{(h)}(s)\right).$$

Target states retain the terminal class at every depth. Increasing h therefore increases how much of the goal directed future structure is preserved by the teacher.

The $h = 1$ relation captures the same immediate optimal action information as the Abstract Q relation in Table 1b. The numerical results differ because Table 1b uses the strict shared pair control, whereas the depth experiment follows the standard continuation training construction. The former isolates the supervision relation; the latter measures how retrieval changes as progressively deeper continuation structure is added.

Table 12: **Effect of continuation depth.** Results use the 49-application corpus with application macro averaging and report mean±standard deviation across three seeds.

Teacher	NA@1	Conflict NA	OptimalAction SemSim@1
$h = 1$	.436±.005	.311±.007	.613±.003
$h = 2$	.486±.008	.371±.006	.631±.004
$h = 3$	.511±.003	.400±.004	.634±.001
$h = 4$	.512±.002	.403±.005	.636±.001
$h = 5$	.506±.005	.401±.007	.634±.002
Full	**.517±.005**	.401±.008	**.637±.004**

Immediate action supervision reaches .436 NEXTACTION@1. Adding one level of successor structure raises performance to .486, and $h = 3$ reaches .511, close to the full recursive teacher at .517. The largest gains therefore come from incorporating the first few levels of continuation structure rather than from matching immediate optimal actions alone.

Table 13: **Paired NEXTACTION@1 comparisons across continuation depths.** Confidence intervals are computed from 10,000 application-level bootstrap samples. W/T/L gives application-level wins, ties, and losses for the first method in each comparison. Paired differences are computed from unrounded per-application scores and therefore need not exactly equal differences between the rounded macro means in Table 12.

Comparison	Δ NA@1	95% CI	W/T/L
Full − $h = 1$	+.080	[+.061, +.099]	41/0/8
Full − $h = 2$	+.030	[+.018, +.041]	38/1/10
Full − $h = 3$	+.005	[−.007, +.017]	24/5/20
$h = 2 - h = 1$	+.050	[+.034, +.066]	35/4/10
$h = 3 - h = 1$	+.075	[+.056, +.095]	41/1/7

Paired Comparisons Across Continuation Depths Full recursion substantially improves over immediate action equivalence. However, $h = 3$ is already close to the full teacher, and the confidence interval for their difference includes zero. These results suggest that most of the useful structural information lies within the first few continuation levels. The full recursive definition remains useful because it captures the complete goal directed continuation structure without introducing a manually chosen horizon.

K ADDITIONAL QUALITATIVE EXAMPLES

K.1 RETRIEVAL EXAMPLES

Figure 6 compares visual retrieval, distance-supervised retrieval, and GCFSim. Visual or distance-based retrieval can return states with similar appearance or goal progress but non-transferable guid-

ance, whereas GCFSim retrieves functionally compatible states whose guidance yields an exact-target optimal next action, $\hat{a}(s_q \mid r, g) \in \mathcal{A}^*_{\text{exact}}(s_q, g)$.

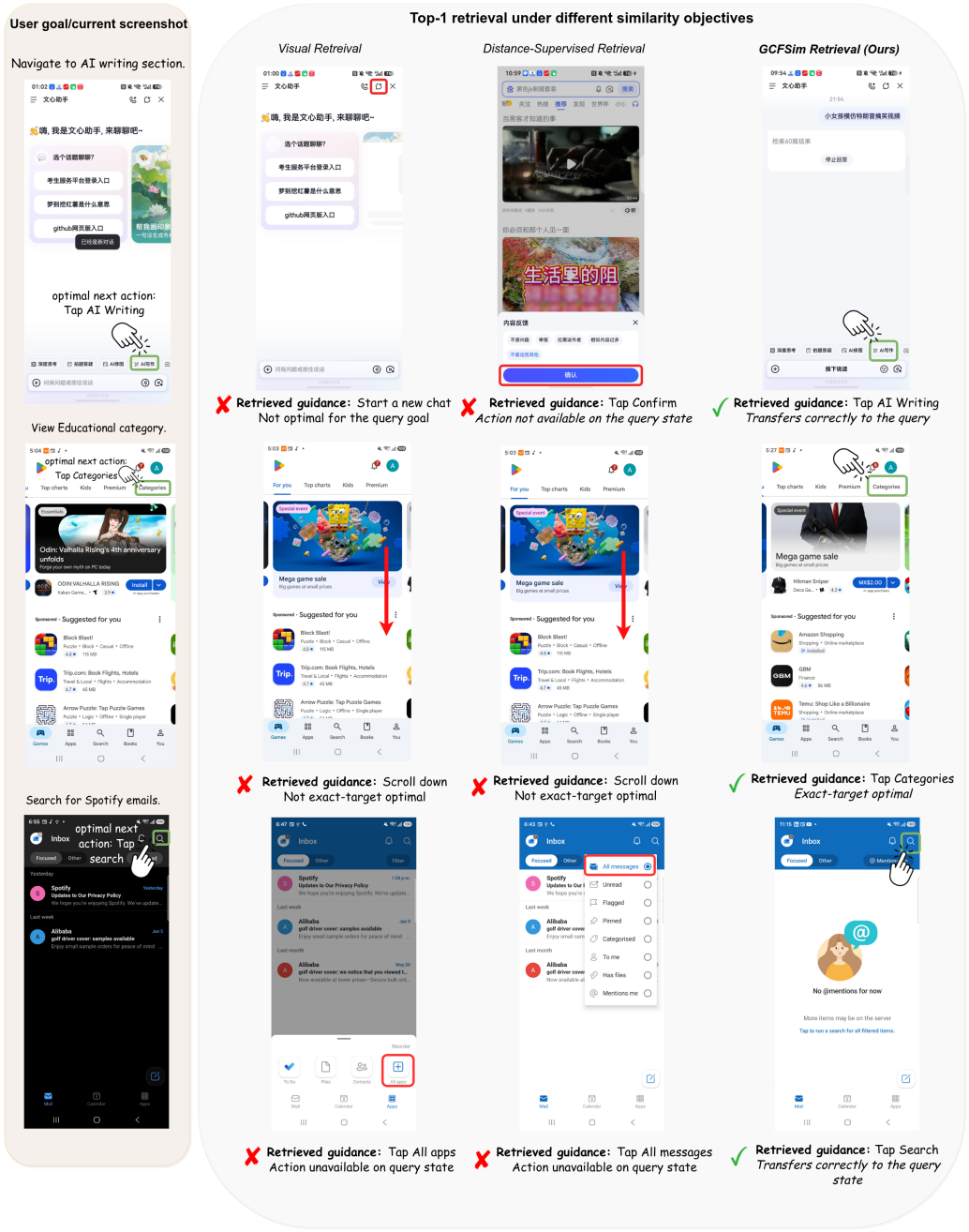


Figure 6: **Additional qualitative retrieval examples.** Visual and distance-supervised retrieval can return non-optimal or non-transferable guidance, while GCFSim retrieves functionally compatible states whose guidance transfers to the query.

K.2 Qualitative Comparison with Immediate-Action Supervision

Figure 7 illustrates why immediate optimal-action information alone is insufficient. GC Abstract Q can retrieve states with compatible immediate actions but incompatible continuations, whereas GCFSim retrieves guidance that transfers correctly to the query.

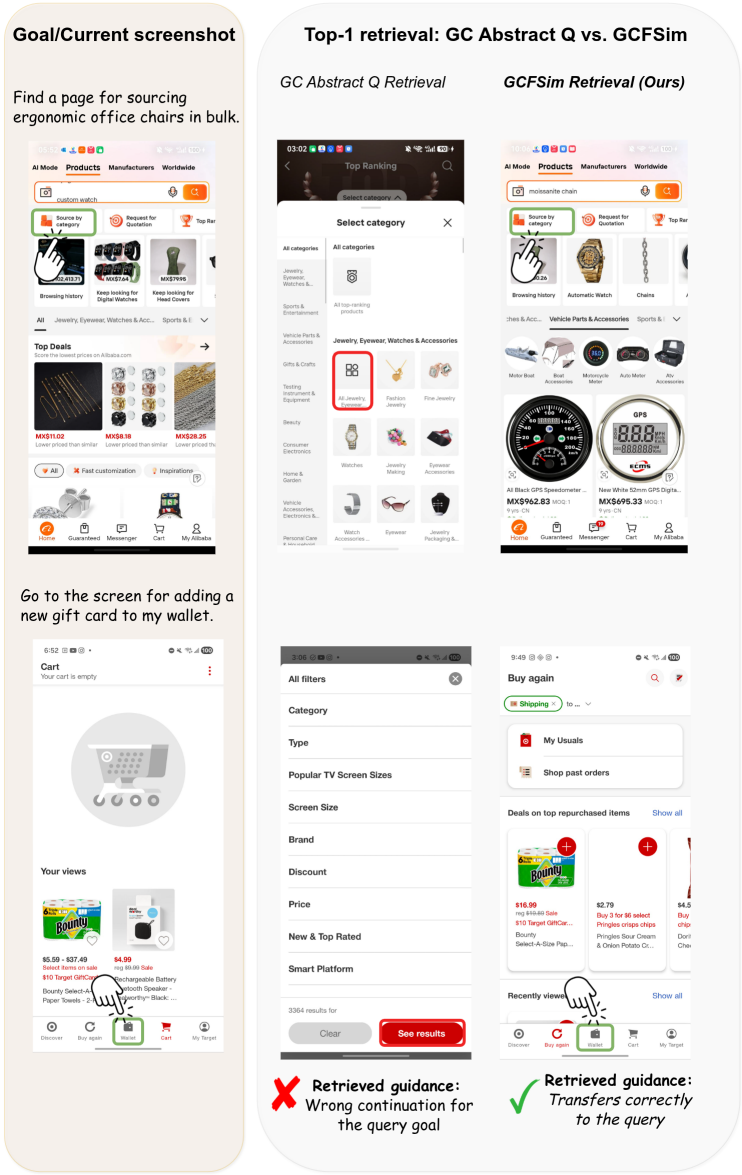


Figure 7: **GC Abstract Q vs. GCFSim.** GC Abstract Q can retrieve states whose immediate action structure appears compatible but whose continuation is incompatible with the query goal. GCFSim retrieves states whose guidance transfers correctly to the query.

K.3 Goal-Dependent Functional Equivalence

Functional similarity depends on the user goal. Figure 8 shows two cases in which the same pair of UI states is functionally equivalent for one goal but not for another. Under one goal, both states support the same useful next decision; under the other, they require different goal-directed decisions. This qualitatively illustrates the goal-flip phenomenon underlying Proposition 1.

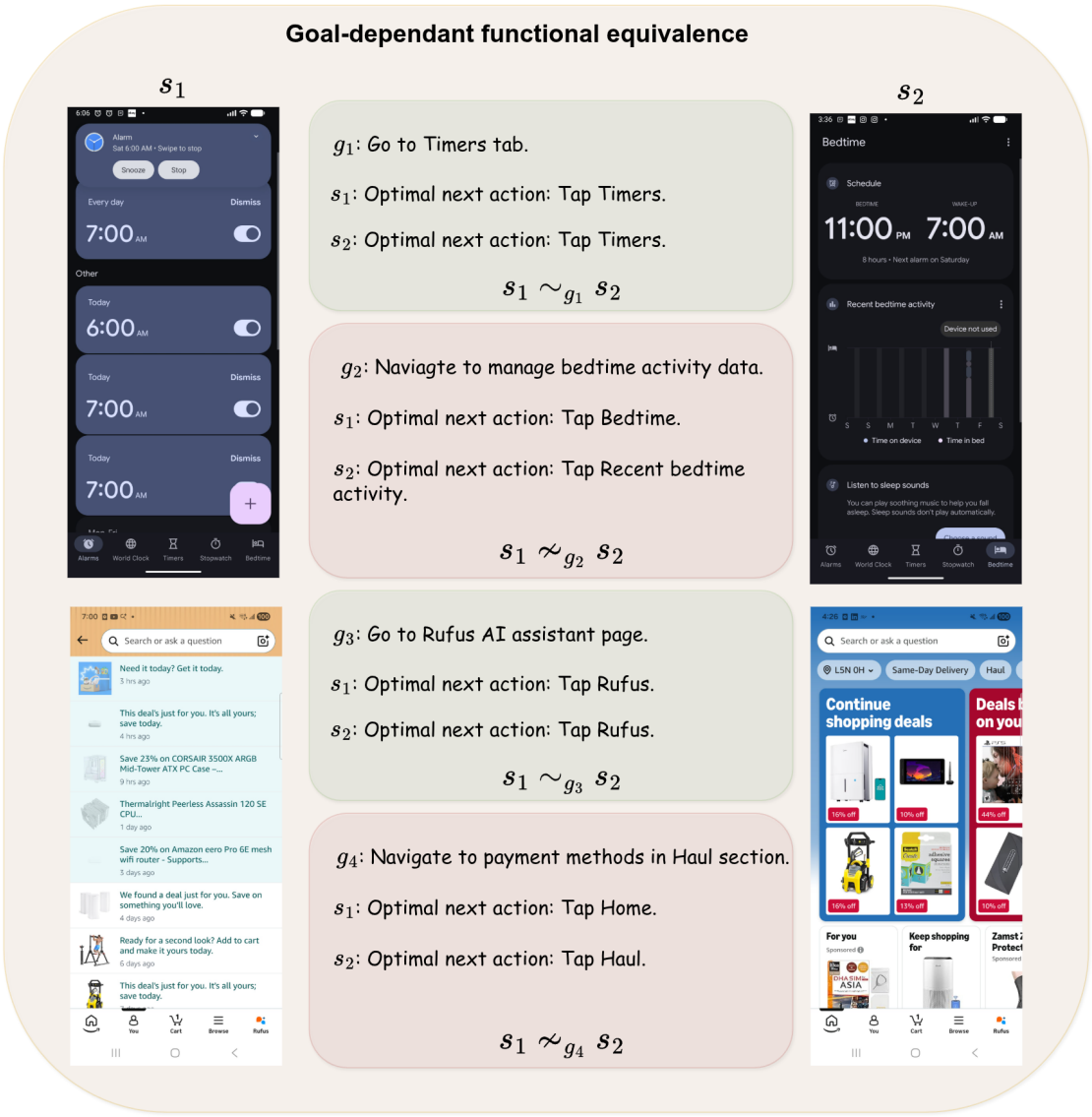


Figure 8: **Goal-dependent functional equivalence.** For the same pair of UI states, the functional relation can change with the user goal. Under one goal the states provide equivalent goal-directed guidance, while under another they require different useful decisions, illustrating why functional similarity must be conditioned on the goal.

L Retrieval Failure Analysis

L.1 Oracle Analysis Within the Top K

We first separate errors caused by insufficient memory coverage from errors caused by retrieval ranking. All analyses use the same candidate memory as the main evaluation after applying the leave one visual filtering protocol.

For a query state s_q and goal g, let $\mathrm{TopK}(s_q, g)$ denote the K highest ranked memory states. A retrieved state r provides useful guidance when its transferred action is exact-target optimal for the query, i.e.,

$$\hat{a}(s_q \mid r, g) \in \mathcal{A}^*_{\mathrm{exact}}(s_q, g).$$

We define

$$\text{NAORACLE@K}(s_q, g) = \mathbb{1}\left[\exists\, r \in \text{TopK}(s_q, g) \;:\; \hat{a}(s_q \mid r, g) \in \mathcal{A}^*_{\text{exact}}(s_q, g)\right].$$

Thus, NAORACLE@1 is identical to NEXTACTION@1. For $K > 1$, the metric asks whether at least one of the top K retrieved states contains transferable optimal guidance, assuming an oracle can select the useful state from that set. This separates the ordering of candidates from whether useful guidance is present in memory at all.

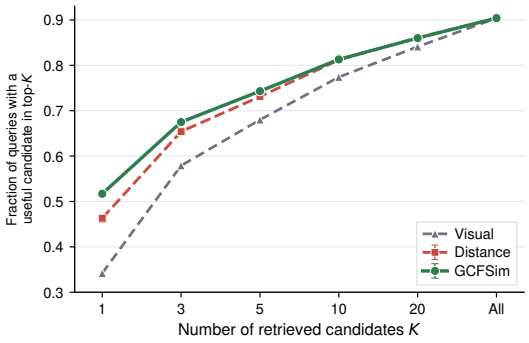


Figure 9: **Top K guidance coverage.** Fraction of queries for which at least one of the top K retrieved states provides transferable optimal guidance.

For GCFSim, NEXTACTION@1 is .517, while NAORACLE@3 reaches .675. Thus, an additional 15.8% of queries already contain useful guidance within the first three retrieved states even though it is not ranked first.

When the entire candidate memory is searched, transferable optimal guidance exists for .904 of queries. The memory coverage failure rate is therefore only .096, whereas the difference between full memory coverage and GCFSim top 1 retrieval is .387.

Figure 9 further shows that the advantage of GCFSim over Goal Distance is largest at small K and narrows as more candidates are considered. This indicates that functional retrieval primarily improves the ordering of useful states near the top of the ranking rather than changing whether useful guidance exists somewhere in memory.

L.2 TOP 1 FAILURE DECOMPOSITION

We next decompose GCFSim's top 1 outcomes to identify the source of the remaining errors. A *useful candidate* is any memory state whose transferred action belongs to $\mathcal{A}^*_{\text{exact}}(s_q, g)$.

Each query is assigned to one of four mutually exclusive outcomes:

- **Success:** the top retrieved state provides transferable optimal guidance.
- **Memory coverage failure:** no candidate in the available memory provides transferable optimal guidance.
- **Functional ranking failure:** useful guidance exists somewhere in memory, but the top retrieved state fails to transfer an optimal action and belongs to a different continuation class from the query.
- **Action transfer or grounding failure:** useful guidance exists somewhere in memory, and the top retrieved state belongs to the same continuation class as the query, but its concrete action does not transfer successfully.

Table 14: **GCFSim top 1 outcome decomposition.** Fractions use application macro averaging and sum to one.

Outcome	Fraction
Success	$.517 \pm .005$
Memory coverage failure	$.096$
Functional ranking failure	$.341 \pm .004$
Action transfer or grounding failure	$.046 \pm .002$

Functional ranking is the dominant residual failure mode. Since the total top-1 failure rate is $1 - .517 = .483$, functional ranking failures account for approximately $.341/.483 = 70.6\%$ of unsuccessful queries. By comparison, memory coverage failures account for 9.6% of all queries, while another 4.6% retrieve a state from the correct continuation class whose concrete action still fails to transfer.

Thus, for most remaining failures, useful guidance is already present in memory but is not ranked first. Insufficient memory coverage and concrete action transfer account for substantially smaller portions of the error, identifying improved discrimination near the top of the ranking as the main opportunity for further gains.