

---

## APPENDIX

This is the supplementary file for our submission titled *PrecogUI: Proactive GUI agents via Pre-cognitive Simulation and Experience Retrieval*. This material supplements the main paper with the following content:

- (A) **The use of large language models**
- (B) **Motivation of PrecogUI**
- (C) **Related work**
- (D) **PolyTouch: A Multi-Gesture and Macro Execution Layer**
- (E) **Additional Experiments**
  - (E.1) Implementation Detail
  - (E.2) Benchmarks
  - (E.3) Hyperparameter analysis of the exploration value.
  - (E.4) Hyperparameter analysis of parser thresholds.
- (G) **Prompts in Automated Pipeline**
  - (F.1) Output Format Structure Template
  - (F.2) Action Selection Template
  - (F.4) Role and Context Template
  - (F.3) Anomaly handling Template
  - (F.5) OS-Specific Hints
  - (F.6) General Instructions.
- (G) **Qualitative Analysis**
- (H) **Additional Discussions**

### A THE USE OF LARGE LANGUAGE MODELS

In this work, large language models (LLMs) are used exclusively for polishing the writing and checking grammar. They are not involved in research ideation, experimental design, data analysis, or the formulation of conclusions. The authors make all substantive intellectual contributions.

### B MOTIVATION OF PRECOGUI

Figure 1 reveals two critical patterns. First, the success rate (SR) declines sharply with an increasing number of injected disturbances. Reactive baselines plummet from nearly 100% SR to below 20% with zero to six injections, showing a performance gap of at least 10% by just two injections (left panel). This highlights the inherent brittleness of purely reactive policies under sustained interference. Second, disturbance timing significantly impacts performance (right panel). Shifting a single injection later in the trajectory yields greater SR losses across all baselines. For instance, UI-TARS exhibits an SR drop escalating from 3.0% (steps 0–5) to 16.4% ( $> 20$  steps). In contrast, PRECOGUI demonstrates consistent resilience, increasing only from 1.6% to 7.1%—approximately  $2.3\times$  less degradation than UI-TARS in the long-horizon tail—while maintaining higher nominal SR. These trends suggest that coupling experience priors with look-ahead simulation is crucial for mitigating late-stage error cascades.

#### B.1 ANALYSIS OF LONG-HORIZON EVALUATION

### C RELATED WORK

**Multimodal Large Language Models.** MLLMs (Li et al., 2023; Liu et al., 2024; Yue et al., 2024) have emerged as a central enabler for GUI automation, boosting both perceptual and reasoning capabilities of agents. By parsing complex screen structures and grounding natural language instructions

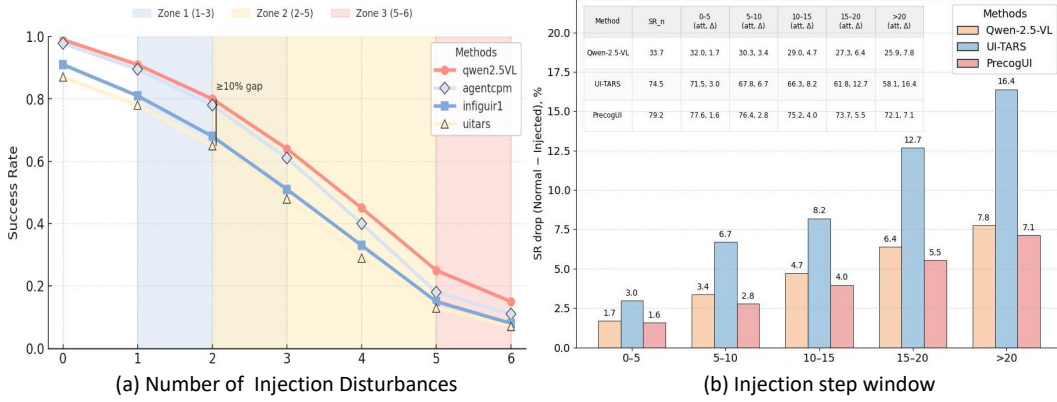


Figure 1: Impact of disturbance count and timing on policy success rates. The left panel shows SR degradation with increasing disturbance count. The right panel illustrates the greater sensitivity of reactive policies to later disturbance injections, in contrast to PRECOGUI’s robustness.

in UI elements, MLLMs serve as the perception backbone for many mainstream agents (Wang et al., 2024a; Yang et al., 2025). However, existing MLLMs (Wang et al., 2024b; Chen et al., 2024; Lai et al., 2024) are primarily pre-trained or fine-tuned on static, single-turn perception tasks such as visual question answering (Ma et al., 2024) or image captioning (Dai et al., 2023). Consequently, in dynamic UI scenarios, they tend to be stateless and myopic, producing immediate responses without sequential modeling or anticipatory reasoning.

**GUI agents.** Research on GUI agents (Gou et al., 2025b; Liu et al., 2025; Xu et al., 2025a) has also explored diverse strategies for policy learning and grounding. A common paradigm (Lu et al., 2025; Lin et al., 2025) is to fine-tune multimodal models, mapping instruction and screenshot inputs into sequential action predictions. For example, UGround (Gou et al., 2025a) trains a purely visual grounding model on millions of UI elements, enabling click and operation solely through visual localisation. Recent efforts (Gao et al., 2025; Zhang et al., 2025) have added structure and memory, with AutoDroid (Wen et al., 2024) handling anomalies by learning corrective scripts and MapAgent retrieving layout traces during planning. While effective on short, static benchmarks (Gao et al., 2024), these methods (Lei et al., 2025; Xu et al., 2025b) remain confined to a reactive framework, in which agents make decisions based solely on the current observation, leaving them vulnerable to unforeseen perturbations. An unexpected pop-up can easily hijack the agents attention, while even minor loading delays may be misinterpreted as failed actions.

## D POLYTOUCH: A MULTI-GESTURE AND MACRO EXECUTION LAYER

Real-world mobile applications often require multi-pointer and multi-step interactions, such as three- or four-finger system shortcuts, pinch/zoom and rotation in media and map viewers, or coordinated sequences in creative tools. Existing GUI agents generally assume single-touch atomic operations and one-shot execution, which makes them fragile when facing complex gestures, long interaction flows, or OS-level controls that demand precise synchronization. To address this gap, we introduce **PolyTouch**, an execution layer that extends the action space to multi-finger gestures and macro-level commands with explicit timing, guards, and rollback mechanisms.

PolyTouch supports a wide range of interaction patterns rarely considered in prior work: (i) Multi-finger chords for dialogs, split-screen, or editing shortcuts; (ii) Continuous gestures such as pinch, zoom, and rotation; (iii) Multi-step flows with explicit waiting, retries, and overlay dismissal; (iv) Recovery sequences (e.g., back, home, or targeted close) that must be executed atomically to exit unexpected states. These abstractions allow agents to operate robustly in long-horizon tasks where traditional atomic actions fail.

PolyTouch builds on Appiums W3C Actions API for deterministic multi-pointer synthesis and it falls back to ADB when accessibility channels are blocked. Its design centers on: (1) deterministic timing through tick-based scheduling; (2) unified coordinate formats (index, relative-in-box,

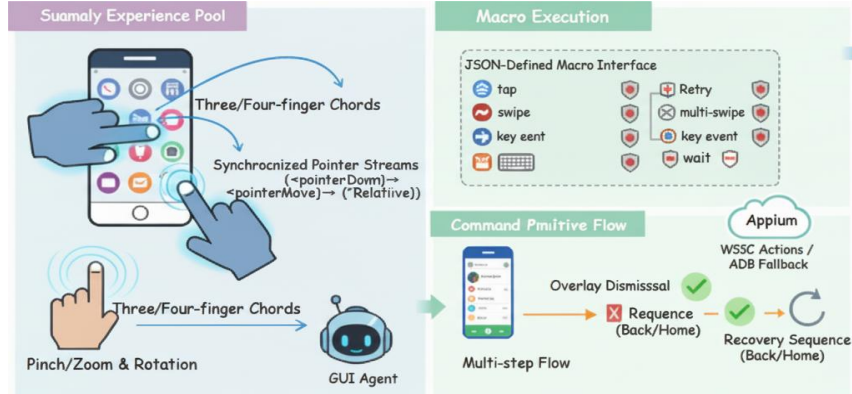


Figure 2: The illustration of PolyTouch, a multi-gesture and macro execution layer for GUI agents. It depicts multi-finger gestures and macro-level commands, highlighting their role in robust, long-horizon task execution.

absolute) with boundary-safe mapping; (3) a declarative macro interface that bundles taps, swipes, multi-swipes, key events, and waits into atomic, retryable units; (4) graceful degradation to equivalent ADB commands while preserving ordering and timing.

PolyTouch exposes two main capabilities: **(a) Multi-gesture execution.** Three- and four-finger gestures are represented as synchronized pointer streams (`pointerDown`  $\rightarrow$  `pointerMove`  $\rightarrow$  `pointerUp`), while pinch/zoom and rotation are parameterized around target boxes and derived from relative coordinates. **(b) Macro execution.** JSON-defined macros encapsulate an ordered list of primitives with explicit guard, retry, and rollback semantics, supporting flexible coordinate specifications.

PolyTouch integrates into the agent control loop by providing reliability-aware plans and structured execution reports (success flags, layout changes, anomaly tags). These outputs feed the Proactive Experience Pool to accumulate reusable patterns and guide the Pre-cognitive Execution Controller in anticipating failures and triggering recovery. In this way, PolyTouch transforms low-level taps into a closed-loop, macro-level control primitive that is both expressive and robust.

## E ADDITIONAL EXPERIMENTS

### E.1 IMPLEMENTATION DETAILS

**Hardware & Devices.** All experiments were conducted on a single training node with  $8 \times$  NVIDIA H20 (96 GB) GPUs. For on-device evaluation, we used a pool of mainstream Android phones covering Huawei/Honor, Xiaomi/Redmi, and OPPO/realme, spanning Android 10–14 and common resolutions (720p–1440p). Devices were connected over USB with ADB (USB debugging enabled) for reliable screenshot capture and input dispatch; Wi-Fi ADB was used only for long-duration soak tests.

**Data Collection & Real-World Tests.** We employ Appium 2.x (Android driver: `uiautomator2`) together with ADB to (i) scrape view hierarchies and screenshots, (ii) execute action sequences in real apps, and (iii) log pre/post frames, timing, and outcomes for replayable trajectories. For latency-critical fallback (e.g., when Appium is blocked by transient overlays), we issue low-level commands via `adb shell input (tap/swipe/keyevent)` and re-sync with Appium on the next stable frame. All experiments use fixed random seeds and identical capture settings across devices; screen coordinates are normalized to  $[0, 1]$  and mapped to device pixels at runtime.

### E.2 BENCHMARKS

**Grounding-Centric Benchmarks: ScreenSpot Series.** Accurate element localization is the foundation of GUI automation. ScreenSpot is a cross-platform grounding benchmark with over 1,200

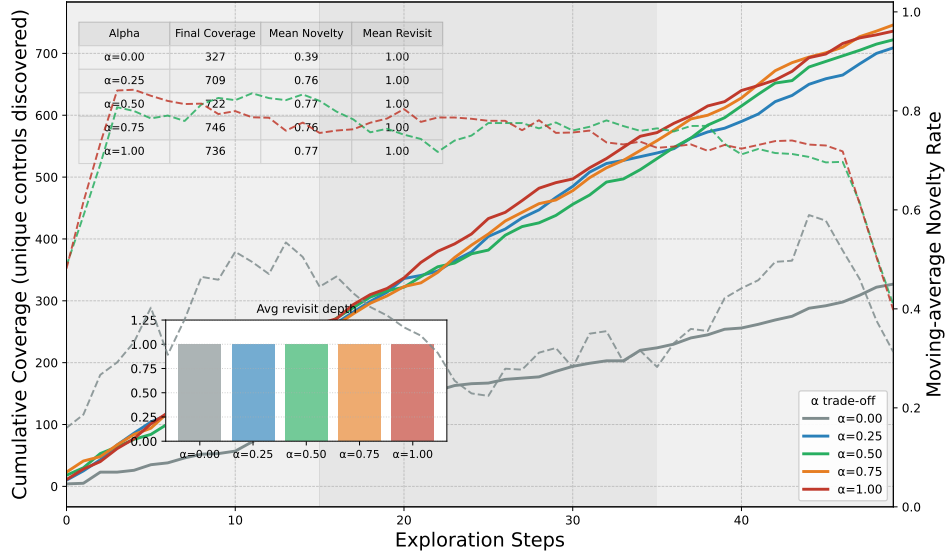


Figure 3: Effect of  $\alpha$  in the exploration value over a 50-step horizon. The main curves show cumulative coverage of unique controls for different  $\alpha$ ; dashed traces (right axis) depict the moving-average novelty, and the inset summarizes average revisit depth and final/mean statistics. Larger  $\alpha$  prioritizes discovery and accelerates coverage, while smaller  $\alpha$  favors rare-state revisits at the cost of slower expansion.

natural-language instructions spanning iOS, Android, macOS, Windows, and Web interfaces. Each instruction is paired with pixel-level bounding boxes and element-type labels (text, icon, or widget) and covers challenging scenarios such as icon-text composites and occluded controls.

**Navigation-Centric Benchmarks: AndroidControl & GUI Odyssey.** Once elements can be reliably located, agents must navigate within and across apps. AndroidControl (Li et al., 2024), the largest public mobile navigation corpus, contains 15,283 human demonstrations divided into low-difficulty single-app workflows (< 10 steps) and high-difficulty cross-app tasks with real-time interruptions (e.g., Select photo from Gallery Upload via Email). It evaluates agents comprehension of both high-level goals (Book a ride) and low-level operations (Tap Search). GUI Odyssey (Lu et al., 2024) extends this to long-horizon, cross-app navigation with 7,735 mission-based episodes across 201 apps and 1,400+ app combinations. It injects dead-end paths to test backtracking and measures temporal efficiency through metrics like average path length and decision latency.

**Disturbance-Aware Benchmark: InterfereBench.** InterfereBench covers 34 applications complex games, enterprise tools, and general apps with bilingual (Zh/En) UIs recorded on diverse phone models. It contains 1,160 long-horizon trajectories (1437 steps) and 27,124 screenshots; we captured 574 real abnormal screens and curated 217 synthetic disturbances (pop-ups, notifications, black screens, layout shifts). Each task is paired with a clean baseline and perturbed variant(s) to enable controlled normal vs perturbed comparisons. Annotations include high-level goals and step-level structures (action type, normalized coordinates, UI boxes, screen deltas, outcomes).

### E.3 HYPERPARAMETER ANALYSIS OF THE EXPLORATION VALUE.

We study the single coefficient  $\alpha$  that balances novel-control discovery against rare-state probing during exploration. As illustrated in Figure 4 On a 50-step horizon, larger settings (e.g.,  $\alpha \geq 0.5$ ) consistently deliver higher cumulative coverage and higher moving-average novelty, indicating faster expansion of the actionable UI space. Very small  $\alpha$  emphasizes repeatedly visiting under-explored screens; while this can stabilize early behavior, it sacrifices coverage and slows progress. We observe no significant increase in redundancy within 50 steps, suggesting that short-horizon exploration benefits most from prioritizing discovery. In practice,  $\alpha \in [0.5, 0.75]$  is a strong operating region that front-loads novel controls without noticeable revisit overhead. For longer horizons or highly volatile

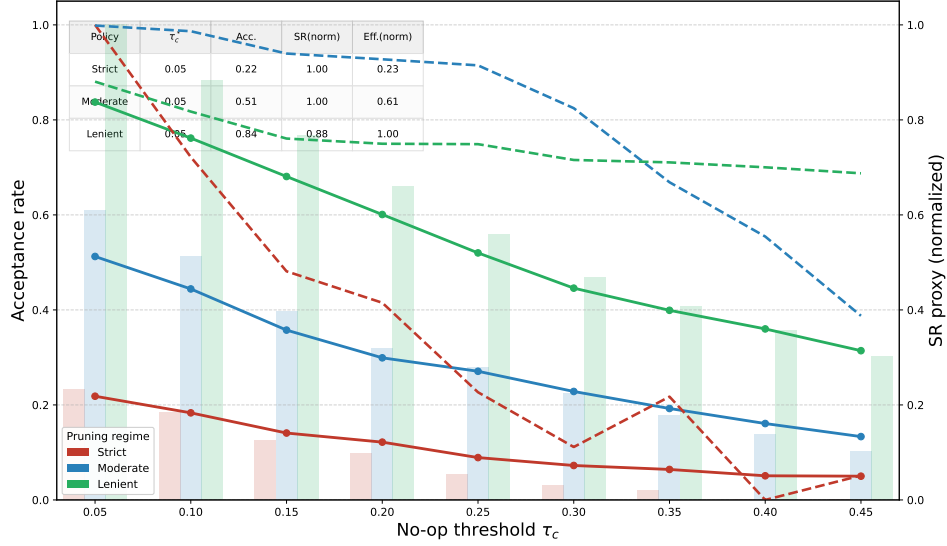


Figure 4: Parser-threshold analysis. The figure summarizes Stage-1 pruning versus  $\tau_c$ : acceptance monotonically decreases as  $\tau_c$  grows; the Moderate regime offers the best acceptance-success-efficiency trade-off near  $\tau_c \in [0.25, 0.35]$ .

apps, an adaptive schedule is preferable: start near  $\alpha \approx 0.5$  to stabilize initial navigation, then increase toward 0.75–1.0 as the uncovered-control ratio declines. Overall,  $\alpha$  provides an interpretable knob for exploration granularity; tuning (or scheduling) it materially impacts coverage speed and downstream success rates.

#### E.4 HYPERPARAMETER ANALYSIS OF PARSER THRESHOLDS.

We examine how the Stage-1 pruning thresholds (self-loop ratio and no-op ratio) interact with the no-op cutoff and impact downstream quality, as shown in Figure. 4. As the cutoff increases, the acceptance rate drops monotonically across all regimes (e.g., from  $\sim 0.60$ – $0.65$  at a low cutoff of 0.05 to  $\sim 0.20$  at 0.45), indicating that more micro-changes are filtered as no-ops. *Strict* pruning rapidly depresses acceptance (often  $< 0.25$  once the cutoff exceeds  $\approx 0.20$ ), and downstream quality declines as data volume becomes the bottleneck. *Lenient* pruning maintains high acceptance ( $> 0.55$  across most cutoffs) but retains many low-signal segments; the success proxy plateaus or degrades when the cutoff is high (e.g., normalized success  $\lesssim 0.55$  once the cutoff  $\geq 0.35$ ). By contrast, the *Moderate* regime achieves the best balance in a mid-range cutoff of **0.25–0.35**: acceptance stays around **0.35–0.50** while the normalized success proxy peaks around **0.75–0.85**, yielding the highest harmonic mean of acceptance and success.

## F PROMPTS IN AUTOMATED PIPELINE

### F.1 OUTPUT FORMAT STRUCTURE TEMPLATE

As illustrated in Figure 5, our `Deep Think & Decision` mechanism is governed by a mandated JSON schema that structures the agent’s output. This schema enforces a rigorous, multi-stage reasoning process through several key fields: `Historical_status` for visual verification of the previous action’s outcome, severing reliance on potentially noisy execution logs; `import_contents` for grounding the agent’s awareness in the current UI context; `think` for articulating a step-by-step causal rationale; `progress` and `next_goal` for explicit task decomposition and forward planning; and finally `action`, which specifies the precise, parameterized command for environmental actuation (e.g., via index-based coordinates). Crucially, the schema’s emphasis on populating fields like `Historical_status` based *solely* on visual evidence establishes a tight closed-loop verification system. This structured output thereby functions as a transparent and auditable interface between the agent’s cognitive deliberation and its concrete actions within the GUI environment.



### Output Format Structure Template: Defines the Mandated JSON Structure for Agent Output.

```
{
  "Historical_status": "Success|Failed|Unknown - Evaluate if the
    previous action visually achieved its intended goal. Base this ONLY on
    the screen image. Ignore the execution result status provided in the
    input.",
  "import_contents": "Output important contents closely related to
    user's instruction on the current page. If there is, please output the
    contents. If not, please output empty string \"\".",
  "think": "Provide a step-by-step thinking process. Analyze the
    current screen, relate it to the overall task and the visual outcome of the
    previous step ('Historical_status'). Decide the next best *single* action.
    Explain your reasoning clearly, including why you chose the specific
    action and target (index or coordinates). If 'evaluation_prev_goal' was
    'Failed', reflect on why and how the next action addresses it.",
  "progress": "Assess the overall progress. List the sub-tasks from the
    main 'task' or 'task_steps' that have been verifiably completed up to
    the current step. This judgment must be based on the *actual visual
    evidence* on the screen, not merely on the sequence of actions planned
    or executed, as actions can fail or produce unexpected results.",
  "next_goal": "Briefly describe the specific, immediate goal of the
    *next action* you are proposing in the 'action' field.",
  "action": {
    "action_name": { /* dictionary of parameters for the action */ } }
}
```

```
{
  "Historical_status": "Failed - The
    previous action (open settings) was
    blocked by a pop-up overlay.",
  "import_contents": "Popup dialog is
    visible with title '活动提醒'; two buttons:
    '立即前往' (index 15) and '稍后再说'
    (index 9). A close icon 'X' at top-right
    (index 5). Underlying Honor of Kings lobby
    is dimmed and not interactable.",
  "think": "A blocking popup covers the
    lobby. The overall task is to continue
    interacting with the lobby. The safest way
    to resume is to dismiss the popup without
    navigating elsewhere. The close icon at
    index 5 is visible and should remove the
    overlay without side effects. I will use the
    'click' action with this index.",
  "progress": ["1. Reached the main lobby
    (visible in background)."],
  "next_goal": "Dismiss the popup to
    restore interaction with the lobby.",
  "action": {"click": {"position": 5}}
}
```

Figure 5: Mandated JSON Schema for Agent Reasoning. The figure shows the output template (left) and an in-context example of handling a pop-up overlay (right).

### Action Position Selection: Standardize the three mutually exclusive ways to target UI and set a priority order.

When specifying the target for an action, choose EXACTLY ONE 'position' form:

- 1) **Highlight Index**: "position": <int>
  - Preferred when the target reliably maps to a single highlighted box.
- 2) **Relative-in-Box**: "position": [<int>, <float>, <float>]
  - Use when the target is INSIDE the highlighted box but needs a precise sub-point (e.g., a small icon inside a large button).
  - Floats are relative coordinates within that box in [0.0, 1.0]: (0,0)=top-left, (1,1)=bottom-right.
- 3) **Absolute Center Coordinates**: "position": [<int>, <int>]
  - Fallback when no reliable highlight exists, the index is unreadable, or the box is inaccurate/too large.
  - Coordinates are normalized pixels in [0,1000] for (x,y); values must not exceed bounds.

Priority: (1) > (2) > (3). If you use (2) or (3), briefly justify why in your reasoning.



Figure 6: Action command selection adopts a three-tiered, prioritized format (Index → Relative-in-Box → Absolute), with in-context examples: (1) semantic targeting via a unique index; (2) fine-grained targeting using relative coordinates within an indexed element; and (3) a robust fallback to normalized absolute coordinates.

## F.2 ACTION SELECTION TEMPLATE

To ensure robust action grounding, we define a hierarchical, three-tiered schema for specifying target coordinates, enforcing a graceful degradation from semantic to pixel-level references. The primary and most preferred format is (1) **Highlight Index**, which targets an element via a unique semantic identifier, providing high resilience to minor layout shifts. The secondary format, (2) **Relative-in-Box**, is used for sub-point targeting within an indexed element, thus combining a semantic anchor with fine-grained precision. The final fallback, (3) **Absolute Coordinates**, is used only when semantic indexing is infeasible, targeting a point in a normalized coordinate space. This strict priority order '(1) > (2) > (3)' ensures that the agent always defaults to the most robust targeting method available.

### Anomaly Handling: Cross-task discipline for reasoning, decomposition, verification, and termination.

#### A) Inputs:

- $L_t$ : current symbolic layout inferred from the current screen.
- $L_{t+1}$ : predicted next-step layout given ( $L_t$ , candidate action, goal).

#### B) Fast anomaly rules (apply to either the current screen or $\hat{L}_{t+1}$ when available):

##### 1) Pop-up/Overlay Anomaly:

- Signs: High-z modal panel covering main content; overlaps multiple interactive controls; typical dismiss affordances ("X/Close/Cancel/Not now/Later"), dimmed background.
- Mitigation: Click a safe dismiss (X/Close/Cancel/Later) → if none, try back once → short wait and re-check. Avoid "Go now/Claim/Start trial" unless explicitly required.

##### 2) Blank Screen

- Signs: Almost no interactive elements or very low saliency; prediction also "blank".
- Mitigation: Short wait → if persistent, back once or refresh per platform → optionally return to a known stable page (menu/home).

##### 3) Freeze / Ineffective Action

- Signs: Layout nearly unchanged and intended state not updated; animation halts without transition.
- Mitigation: Retry once with improved targeting (Relative-in-Box or safer index) → if still unchanged, back or wait then retry → if recurrent, re-plan (alternate path/control).

##### 4) Off-Goal / Misdirection

- Signs: Next screen diverges from goal (e.g., store/ads), goal elements vanish.
- Mitigation: Abort the risky path; dismiss/ back to restore context; choose a safer, on-goal alternative.

#### C) Post-Mitigation Re-check: After handling any anomaly, re-check: target page/controls are visible and no overlay remains; then continue the main task.

Figure 7: Mandated JSON Schema for Agent Reasoning. The figure shows the output template (left) and an in-context example of handling a pop-up overlay (right).

### F.3 ANOMALY HANDLING TEMPLATE

As shown in Figure 7, we frame anomaly handling as a concise, cross-task routine over prediction and verification. Given the current layout  $L_t$  and the forecast  $\hat{L}_{t+1}$ , the agent applies fast rules to classify and mitigate: (i) Pop-up/Overlay dismiss via safe affordances (X/Close/Cancel/Later); (ii) Blank Screen short wait, then Back/refresh or return to a stable hub; (iii) Freeze/Ineffective Actions single retry with safer targeting (Relative-in-Box or safer index), else Back/re-plan; (iv) Off-Goal/Misdirection abort the path and restore on-goal context. A compulsory post-mitigation re-check gates progress: continue only when target controls are visible and no overlay persists.

#### Role & Context: Define the agent's responsibilities and the I/O context (screenshots with highlighted regions, prior execution result, step number).

You are an expert GUI automation agent. Your job is to complete the user's task by interacting with PC/mobile GUIs using screenshots.

For each step, you receive:

- 1) The current screenshot with highlighted UI regions (each region has a top-left index).
- 2) The previous action's execution result (success/failed/unknown).
- 3) The current step number.

Always reason from on-screen visual evidence. Highlight boxes help locate elements; completion must be judged by actual page state changes (texts, titles, control states).

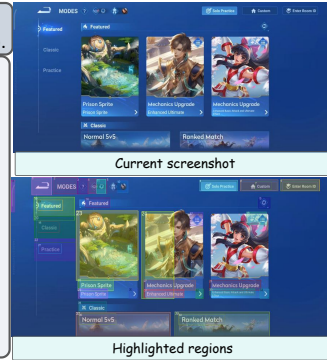


Figure 8: Role and context template. Specifies agent responsibilities and I/O context with indexed screenshots, prior execution results, and step numbers to guide evidence-based task completion.

### F.4 ROLE AND CONTEXT TEMPLATE

To structure the agent's operational context, we define a clear set of responsibilities and a standardized input format for each reasoning step. As illustrated in Figure 8, the agent is prompted with persona as an expert GUI automation agent. For each step, it receives a tripartite input: (1) the current screenshot augmented with indexed, highlighted bounding boxes over interactive elements; (2) feedback on the execution status (e.g., success or failure) of the prior action; and (3) the current temporal step index. Crucially, the agent is explicitly instructed to ground its reasoning *solely on visual evidence*, judging task progression based on observable changes in the UI state rather than

378  
379  
380  
381  
382  
383  
384  
385  
386  
387  
388  
389  
390  
391  
392  
393  
394  
395  
396  
397  
398  
399  
400  
401  
402  
403  
404  
405  
406  
407  
408  
409  
410  
411  
412  
413  
414  
415  
416  
417  
418  
419  
420  
421  
422  
423  
424  
425  
426  
427  
428  
429  
430  
431

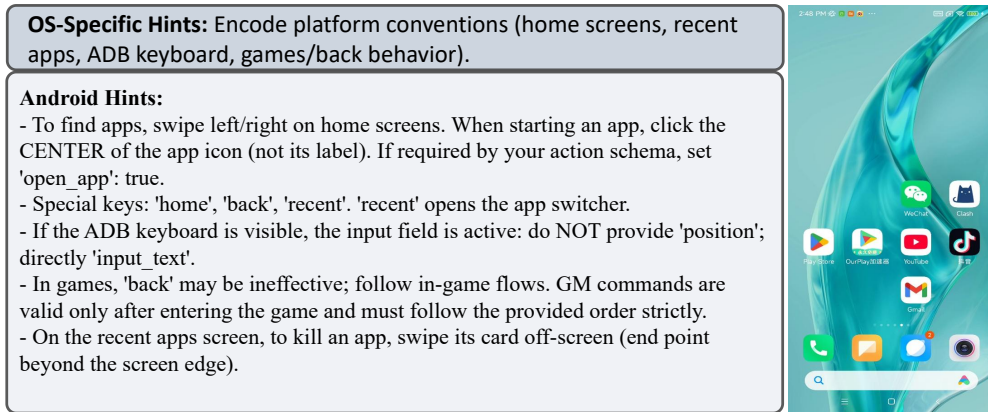


Figure 9: OS-specific action hints. Encodes Android conventions for app access, navigation keys, keyboard input, in-game flows, and app termination to ensure robust, context-aware execution.

uncritically accepting the programmatic execution status. This mandate establishes a tight, closed-loop visual verification process for all decision-making.

## F.5 OS-SPECIFIC HINTS

As shown in Figure 9, we encode platform conventions into structured hints that guide robust action execution on Android. These rules address common UI operations and context-sensitive behaviors: (i) app launching via centered icon clicks with optional `open_app` flag; (ii) special system keys such as `home`, `back`, and `recent` for navigation control; (iii) text input handling by directly invoking `input_text` when the ADB keyboard is active, avoiding redundant position specifications; (iv) game-specific flows where the `back` key may be ineffective, requiring strict adherence to in-game command order; and (v) app termination through swipe-off gestures in the recent-apps screen. Collectively, these hints ground agent actions in OS-level semantics, reducing execution ambiguity and improving cross-context stability.

## F.6 GENERAL INSTRUCTIONS.

As shown in Figure 10, this template encodes cross-task discipline for structured reasoning and verifiable execution. It emphasizes (i) step-by-step task decomposition into checkable sub-steps; (ii) precise targeting using highlighted regions or indices while avoiding ambiguous clicks; (iii) progress verification strictly by on-screen evidence such as titles, messages, or control states; (iv) controlled waiting to accommodate delays or animations; (v) fallback to anomaly-handling rules when overlays appear; and (vi) termination only after explicit visual confirmation of success. When targeting remains uncertain, the agent is required to re-locate or choose safer alternatives, ensuring robustness against cascading errors. Collectively, these rules establish a disciplined action loop where correctness validation precedes task advancement.

## G QUALITATIVE ANALYSIS

### G.1 APPS POP-UP HANDLING

As shown in Figure 11, we deploy a type-aware policy that closes in-app pop-ups while preserving task context. The controller first classifies the pop-up(i) announcement/notice panels, (ii) gift-package ads, (iii) event promotions, or (iv) confirmation/input dialogs and selects the safest dismissal affordance. Execution follows our hierarchical position schema: prioritize element indices for `X/Close/Cancel/Later`; degrade to Relative-in-Box when the target is a sub-control; and use normalized absolute coordinates only when indexing is unreliable. Each thumbnail shows the predicted command (index or relative point) rendered beneath the image; progress continues only after the overlay is visually cleared.



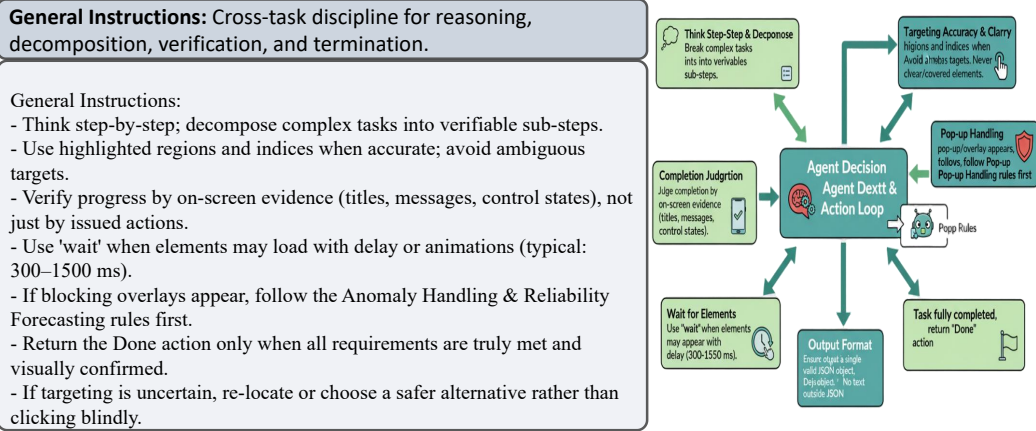


Figure 10: General instruction template. Defines structured reasoning, precise targeting, verification, controlled waiting, and disciplined termination to ensure robust, evidence-driven task execution.



Figure 11: Apps pop-up handling. A type-aware policy combined with hierarchical position selection (Index → Relative-in-Box → Absolute); the figure presents concrete dismissal commands for diverse pop-up cases.

## G.2 SYSTEM-LEVEL POP-UP HANDLING.

As shown in Figure 12, we handle OS-mediated interruptionssystem notifications, risk alerts, and permission requests via a type-conditioned, safety-first policy. The controller classifies the pop-up and selects the safest affordance (e.g., Cancel/Close, Allow only while in use, Deny). Execution uses our hierarchical position scheme, prioritizing element indices and backing off to Relative-in-Box or normalized Absolute coordinates only when indexing is unreliable. Each panel displays the issued command (primarily index clicks), and progress resumes only after the overlay is visually cleared to preserve task context.

## G.3 ENVIRONMENT PERTURBATION HANDLING.

As shown in Figure 13, we address environment-level disturbances (black/white screens, loading delays, and network stalls) with a lightweight stabilization routine. Detection relies on low-saliency/blank frames, near-identical consecutive layouts, or stalled progress indicators. Mitigation

For system-level pop-ups, including system notifications, risk alerts, and permission requests, PrecogUI automatically selects the safest dismissal option (e.g., Cancel, Close,, or Deny) based on the pop-up type.

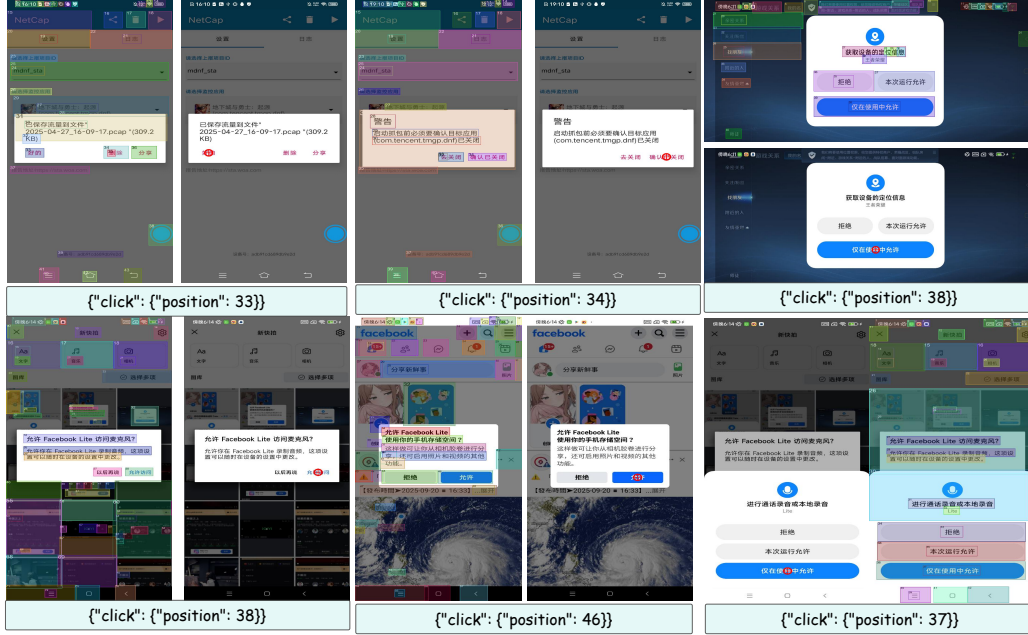


Figure 12: System pop-up handling. A type-aware policy selects safe dismissal actions and executes them with index-prioritized targeting; the figure shows concrete commands for notifications, risk alerts, and permission requests.

For black/white screens, loading delays, and network stalls, PrecogUI uses a lightweight stabilization routine, combining short waits (e.g., 200 ms) with index-prioritized safe retries.



Figure 13: Environment disturbances. A lightweight routine short waits plus index prioritized safe retries stabilizes black/white screens, delayed loads, and network stalls; the figure shows concrete wait and retry commands for representative cases.

is minimal yet effective: inject a short wait (e.g., 200 ms) to absorb transient transitions, then issue a single index-prioritized safe retry of the previous action; progress resumes only after visual evidence of recovery, otherwise control is escalated to the general anomaly rules.

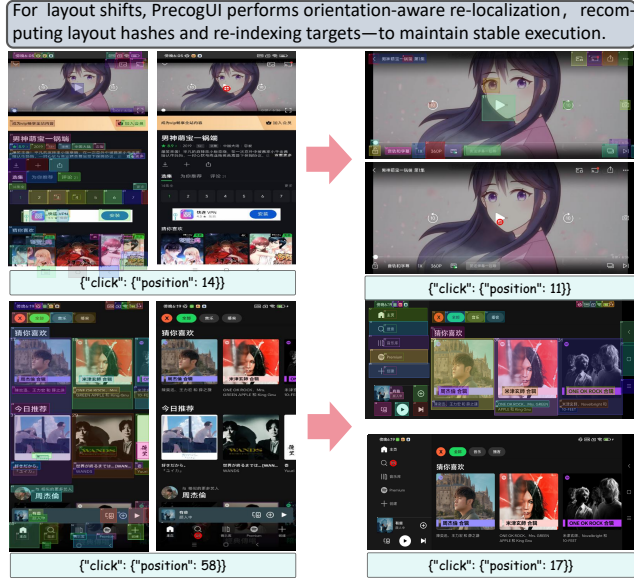


Figure 14: Layout-shift handling. PrecogUI rebuilds layout hashes and re-indexes targets under portrait/landscape transitions, executing with index-first targeting; the figure shows before/after screens with preserved action intent.

**Layout-Shift Perturbations.** As shown in Figure 14, we address orientation/gravity-induced reflows (portrait  $\leftrightarrow$  landscape) with an orientation-aware re-localization routine. Upon detecting a layout shift (aspect-ratio change and index invalidation), the agent reconstructs the symbolic layout hash, re-indexes targets, and remaps the current goal to the new arrangement by type/text cues. Execution then follows the hierarchical position policy (Index  $\rightarrow$  Relative-in-Box  $\rightarrow$  Absolute), and progress is gated by visual re-check to ensure the intended control is active after rotation.

## H ADDITIONAL DISCUSSIONS

Forecasting future layouts is central to PrecogUI: look-ahead turns reactive observe-act behavior into risk-aware planning that preempts pop-ups, freezes, and off-goal drifts, improving long-horizon stability. However, timeliness is a key constraint. Pre-execution simulation and verification add latency and compute, which can be costly for real-time use or very long tasks. In addition, experience priors can become stale as apps update; outdated remedies hurt reliability unless memory is refreshed. Future work should adopt lightweight, anytime forecasting and drift-aware memory maintenance to preserve the gains of look-ahead without sacrificing responsiveness.

## REFERENCES

- Zhe Chen, Jiannan Wu, Wenhai Wang, Weijie Su, Chen, and et al. Internvl: Scaling up vision foundation models and aligning for generic visual-linguistic tasks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 24185–24198, June 2024.
- Wenliang Dai, Junnan Li, DONGXU LI, Anthony Tiong, Junqi Zhao, Weisheng Wang, Boyang Li, Pascale N Fung, and Steven Hoi. Instructblip: Towards general-purpose vision-language models with instruction tuning. In *Advances in Neural Information Processing Systems*, volume 36, pp. 49250–49267, 2023.
- Longxi Gao, Li Zhang, Shihe Wang, Shangguang Wang, Yuanchun Li, and Mengwei Xu. Mobile-views: A large-scale mobile gui dataset, 2024.
- Xinzge Gao, Chuanrui Hu, Bin Chen, and Teng Li. Chain-of-memory: Enhancing GUI agents for cross-application navigation. *arXiv preprint arXiv:2506.18158*, June 2025.

- 
- Boyuan Gou, Ruohan Wang, Boyuan Zheng, Yanan Xie, Cheng Chang, and et al. Navigating the digital world as humans do: Universal visual grounding for gui agents. In Proceedings of the International Conference on Learning Representations (ICLR), April 2025a.
- Boyuan 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, 2025b.
- Xin Lai, Zhuotao Tian, Yukang Chen, Yanwei Li, Yuhui Yuan, Shu Liu, and Jiaya Jia. Lisa: Reasoning segmentation via large language model. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp. 9579–9589, June 2024.
- Weixian Lei, Difei Gao, and Mike Zheng Shou. Grounding multimodal large language model in GUI world. In The Thirteenth International Conference on Learning Representations, 2025.
- Junnan Li, Dongxu Li, Silvio Savarese, and Steven Hoi. BLIP-2: Bootstrapping language-image pre-training with frozen image encoders and large language models. In Proceedings of the 40th International Conference on Machine Learning, volume 202, pp. 19730–19742, 2023.
- Wei Li, William Bishop, Alice Li, Chris Rawles, Folawiyo Campbell-Ajala, Divya Tyamagundlu, and Oriana Riva. On the effects of data scale on computer control agents, 2024.
- Kevin Qinghong Lin, Linjie Li, Difei Gao, Zhengyuan Yang, Shiwei Wu, Zechen Bai, Stan Weixian Lei, Lijuan Wang, and Mike Zheng Shou. Showui: One vision-language-action model for gui visual agent. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp. 19498–19508, June 2025.
- Haotian Liu, Chunyuan Li, Yuheng Li, and Yong Jae Lee. Improved baselines with visual instruction tuning. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp. 26296–26306, June 2024.
- Yuhang Liu, Pengxiang Li, Congkai Xie, Xavier Hu, Xiaotian Han, Shengyu Zhang, Hongxia Yang, and Fei Wu. Infigui-r1: Advancing multimodal gui agents from reactive actors to deliberative reasoners, 2025.
- Quanfang Lu, Wenqi Shao, Zitao Liu, Fanqing Meng, Boxuan Li, Botong Chen, Siyuan Huang, Kaipeng Zhang, Yu Qiao, and Ping Luo. Gui odyssey: A comprehensive dataset for cross-app gui navigation on mobile devices, 2024.
- Yaxi Lu, Shenzhi Yang, Cheng Qian, Guirong Chen, Qinyu Luo, and et al. Proactive agent: Shifting llm agents from reactive responses to active assistance. In Proceedings of the International Conference on Learning Representations (ICLR), pp. 1–27, April 2025.
- Jie Ma, Pinghui Wang, Dechen Kong, Zewei Wang, Jun Liu, Hongbin Pei, and Junzhou Zhao. Robust visual question answering: Datasets, methods, and future challenges. IEEE Transactions on Pattern Analysis and Machine Intelligence, 46(8):5575–5594, 2024.
- 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, 2024a.
- Wei Han Wang, Qingsong Lv, Wenmeng Yu, Wenyi Hong, Ji Qi, and et al. Cogvlm: Visual expert for pretrained language models. In Advances in Neural Information Processing Systems, volume 37, pp. 121475–121499, 2024b.
- 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. 543557, 2024.
- Yiheng Xu, Dunjie Lu, Zhennan Shen, Junli Wang, Zekun Wang, Yuchen Mao, Caiming Xiong, and Tao Yu. Agenttrek: Agent trajectory synthesis via guiding replay with web tutorials. In The Thirteenth International Conference on Learning Representations, 2025a.

- 
- Yiheng Xu, Zekun Wang, Junli Wang, Dunjie Lu, Tianbao Xie, Amrita Saha, Doyen Sahoo, Tao Yu, and Caiming Xiong. Aguis: Unified pure vision agents for autonomous GUI interaction. In Forty-second International Conference on Machine Learning, 2025b.
- Jianwei Yang, Reuben Tan, Qianhui Wu, Ruijie Zheng, Baolin Peng, Yongyuan Liang, Yu Gu, Mu Cai, Seonghyeon Ye, Joel Jang, Yuquan Deng, and Jianfeng Gao. Magma: A foundation model for multimodal ai agents. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp. 14203–14214, June 2025.
- Xiang Yue, Yuansheng Ni, Kai Zhang, Tianyu Zheng, Ruoqi Liu, and et al. Mmmu: A massive multi-discipline multimodal understanding and reasoning benchmark for expert agi. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp. 9556–9567, June 2024.
- Zeyu Zhang, Quanyu Dai, Xiaohe Bo, Chen Ma, Rui Li, and et al. A survey on the memory mechanism of LLM-based agents. ACM Transactions on Information Systems (TOIS), 43(6):155:1–155:47, September 2025.