# SENSEACT: STRUCTURING GUI ACTIONS FOR RELIABLE PLANNING AND VERIFICATION

**Hongtian Cai**[1,2*], **Tianyi Ma**[1,2*], **Jie-Jing Shao**[1,3], **Tianyi Tang**[1], **Ivor Tsang**[1,2], **Yueming Lyu**[1†], **Haiyan Yin**[1]
[1]CFAR and IHPC, Agency for Science, Technology and Research (A*STAR), Singapore
[2]Nanyang Technological University (NTU), Singapore
[3]State Key Laboratory of Novel Software Technology, Nanjing University, China

## ABSTRACT

Reliable interaction with graphical user interfaces (GUIs) requires agents to make irreversible control commitments under partial observability, yet most existing GUI agents reduce this problem to step-by-step, perception-conditioned action prediction, leaving decisions implicit and execution unverifiable. We introduce **SenseAct**, a new paradigm for GUI agents that lifts low-level GUI observations (e.g., XML/DOM trees) into explicit control commitments specifying both controllable interface elements and their expected state transitions. In SenseAct, decision making is formulated over typed control primitives whose dependencies and ordering are explicitly represented, and whose execution is governed by programmatic post-condition predicates that define precise, observable success criteria. This formulation grounds task progress in symbolic state transitions, turning execution verification into a well-defined, deterministic decision rather than a language-based self-judgment. As a consequence of these explicit execution semantics, SenseAct can operate effectively without requiring pixel-level visual reasoning during normal execution and invokes the VLM only when symbolic state transitions violate expected control effects. On the challenging benchmarks DroidTask and AndroidLab, SenseAct achieves a reduction in VLM calls by 14.49% and 30.55% respectively while scoring higher success rates, demonstrating that internalizing programmatic verification within a closed loop control process effectively eliminates execution drift and enhances the efficiency of GUI representations.

## 1 INTRODUCTION

Graphical user interfaces (GUIs) are a central testbed for general-purpose agents: they require executing long-horizon, irreversible actions under partial observability, while success must be determined from evolving interface state rather than from free-form text. Recent interactive benchmarks (e.g., WebArena, OSWorld, AndroidWorld) make this gap explicit by evaluating agents on real interfaces with execution-based success checking, where plan–execution inconsistency remains a dominant failure mode (Zhou et al., 2023; Xie et al., 2024; Rawles et al., 2024).

The prevailing recipe for GUI agents extends language-agent paradigms (e.g., ReAct) to multimodal settings, casting interaction as *perception-conditioned action prediction*: at each step, the agent interprets visual observations and autoregressively emits the next primitive action (Yao et al., 2022). Systems such as AppAgent and SeeClick instantiate this approach for real applications, enabling gesture-level execution from screenshots without privileged back-end APIs (Zhang et al., 2023; Cheng et al., 2024). While effective at short horizons, this formulation tightly conflates (i) perception, (ii) decision making, (iii) execution, and (iv) success judgment into a single generation loop making errors hard to localize and easy to compound.

Crucially, scaling perception does not resolve the core bottleneck. Across OSWorld/AndroidWorld and web GUI benchmarks, many trajectories that are visually grounded and linguistically plausible

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

| Interaction Axis | Existing GUI Agents | SenseAct (Ours) |
|---|---|---|
| Perception | Pixels / UI trees | Action-conditioned symbolic state |
| Decision Space | Action sequence | **Structured control graph (SCG)** |
| Decision Mechanism | Autoregressive action prediction | **Programmatic control construction** |
| Execution Criterion | Implicit language judgment | **Post-condition predicates over symbolic state transitions** |
| Vision Invocation | Per-step (always-on) | Exception-driven (control mismatch only) |

Table 1: SenseAct replaces language-mediated action prediction with **programmatic control construction**. Perception no longer directly drives actions; instead, it yields control-relevant symbolic state that constrains decision making. Consequently, SenseAct operates in a *vision-free regime by default*, invoking visual reasoning only when symbolic control expectations fail.

still fail to realize the intended UI state transitions (Xie et al., 2024; Rawles et al., 2024; He et al., 2024; Koh et al., 2024). This suggests that the dominant failure is not merely *mis-seeing*, but *mis-controlling*: agents lack an explicit representation of (a) what control commitment is being made, (b) how commitments compose with dependencies, and (c) what observable evidence certifies successful execution. Verifier-centric and process-supervised approaches begin to address this gap, but typically remain coupled to generator-centric action sequences rather than making control semantics first-class (Dai et al., 2025; Lu et al., 2025).

We therefore reframe GUI interaction as **control synthesis under partial observability** with **explicit execution semantics**. Concretely, instead of generating an opaque action sequence, an agent should construct an inspectable control object whose primitives have typed pre/post-conditions over symbolic UI state, so that progress can be verified deterministically and replanning can be grounded in observed state deltas. Based on this view, we introduce **SenseAct**, a paradigm that lifts low-level GUI observations (e.g., XML/DOM/AX structures) into *action-conditioned symbolic state*, synthesizes a *Structured Control Graph (SCG)* whose nodes are typed control primitives and whose edges encode ordering/dependency constraints, and executes each primitive under *programmatic post-condition predicates* over symbolic state transitions. SenseAct thus decouples perception, control construction, execution, and verification, while coupling them through a shared control interface. A direct consequence is that SenseAct can run *vision-free by default*, invoking VLM reasoning only when symbolic transitions violate expected control effects.

The contributions of this paper are three-fold:

- We identify that long-horizon GUI failures are dominated by missing control semantics and unverifiable execution, even when perception is strong.
- We formalize GUI interaction as control synthesis under partial observability with explicit, checkable execution criteria, separating control construction from control execution.
- We propose **SenseAct**, which operationalizes this formulation via Structured Control Graphs and post-condition verification, enabling robust long-horizon interaction with exception-driven vision.

## 2 Related Works

### 2.1 GUI Agents and Multimodal Interaction

Recent progress in GUI agents has been largely driven by extending language-based agent paradigms, most notably ReAct, to interactive visual environments. In this line of work, GUI interaction is typically realized through a unified textual loop that interleaves natural-language reasoning with low-level environment actions (Yao et al., 2022). Multimodal agents such as AppAgent and SeeClick instantiate this paradigm in mobile and cross-platform settings, enabling agents to operate real applications directly from visual observations using human-like gestures (Zhang et al., 2023; Cheng et al., 2024). Across these systems, perception, decision making, and execution are tightly coupled within autoregressive action generation, with control decisions emerging implicitly from step-by-step interaction rather than being represented as explicit objects.

In parallel, a series of realistic GUI benchmarks and environments, including WebArena, OSWorld, AndroidWorld, AndroidLab, and macOSWorld, have substantially advanced the evaluation of GUI agents by emphasizing reproducible task setups and execution-based success checking over real interfaces (Zhou et al., 2023; Xie et al., 2024; Rawles et al., 2024; Xu et al., 2024; Yang et al., 2025). Algorithmic extensions, such as structured reasoning with history summarization, explicit reflection, and action verification, further improve robustness within this paradigm (Liu et al., 2025a; Dai et al., 2025). Nevertheless, these approaches continue to treat GUI interaction primarily as reactive action selection, leaving the structure of control decisions and the semantics of successful execution implicit within language-driven inference.

### 2.2 UI Perception and Representation

A recurring empirical observation is that GUI agents fail less from misrecognizing visual elements than from lacking *action-conditioned perceptual insights*. Effective representations must encode not only what is present on the screen, but also what is controllable and how the interface state is expected to change under interaction (Deka et al., 2017; Koh et al., 2024; Rawles et al., 2024). Early work such as Rico demonstrated the value of pairing screenshots with view hierarchies to expose structural cues, including layout, widget types, and containment, which are often ambiguous from pixels alone (Deka et al., 2017). Subsequent studies further show that perceptual semantics play a decisive role: screen-level summaries (Wang et al., 2021), widget captioning (Li et al., 2020), and icon understanding (Zang et al., 2021) all reduce action ambiguity by clarifying affordances rather than merely improving detection accuracy.

At the same time, purely descriptive or compressed representations remain fragile under realistic GUI conditions. Detection-centric pipelines localize actionable elements but often underspecify why an action is appropriate, motivating joint localization and semantics models (Wang et al., 2023). UI-specific multimodal models and grounding approaches highlight the importance of queryable, referential representations that align perception with executable actions (You et al., 2024; Li et al., 2024). While screenshot-only learning can recover actionable insights without privileged metadata (Cheng et al., 2024), benchmarks such as ScreenSpot-Pro and UI-I2E-Bench reveal systematic failures on small targets, long-tail widgets, and professional layouts, underscoring the need for high-resolution, diagnostically faithful representations (Li et al., 2025; Liu et al., 2025b). Collectively, these findings suggest a unifying design principle: UI perception for agents must surface control-relevant predicates, including target identity, action validity, and expected state transitions, so that decision making can be guided and verified rather than inferred post hoc from language or heuristics.

### 2.3 Planning and Execution in Agent Systems

In interactive environments such as GUIs, however, these advances expose a sharper requirement: planning structure alone is insufficient without explicit execution semantics. Recent efforts on structured workflow representations further suggest that making planning explicit and verifiable is a necessary step toward reliable execution, rather than relying on implicit or code-level formulations (Zheng et al., 2025). Empirical studies on OSWorld, AndroidWorld, WebVoyager, and VisualWebArena consistently show that many trajectories that appear reasonable at the language level fail to produce the intended interface state changes (Xie et al., 2024; Rawles et al., 2024; He et al., 2024; Koh et al., 2024). This has motivated verifier-driven designs and process-level supervision, where actions are evaluated against checkable conditions rather than narrative plausibility (Dai et al., 2025; Lu et al., 2025; Xiang et al., 2026). Moreover, failures in long-horizon interaction further reveal the need for fine-grained attribution and repair mechanisms, where execution breakdowns can be localized and corrected rather than treated as trajectory-level errors (Li et al., 2026). Collectively, these results suggest that effective planning–execution loops require not only structured plans, but also programmatic criteria that determine whether execution has actually realized the intended state transitions, so that subsequent planning can be grounded in observable outcomes.

## 3 Methodology

SenseAct formulates GUI interaction as a problem of control construction with explicit execution semantics, rather than as reactive action generation. As illustrated in Figure 1, the agent main-

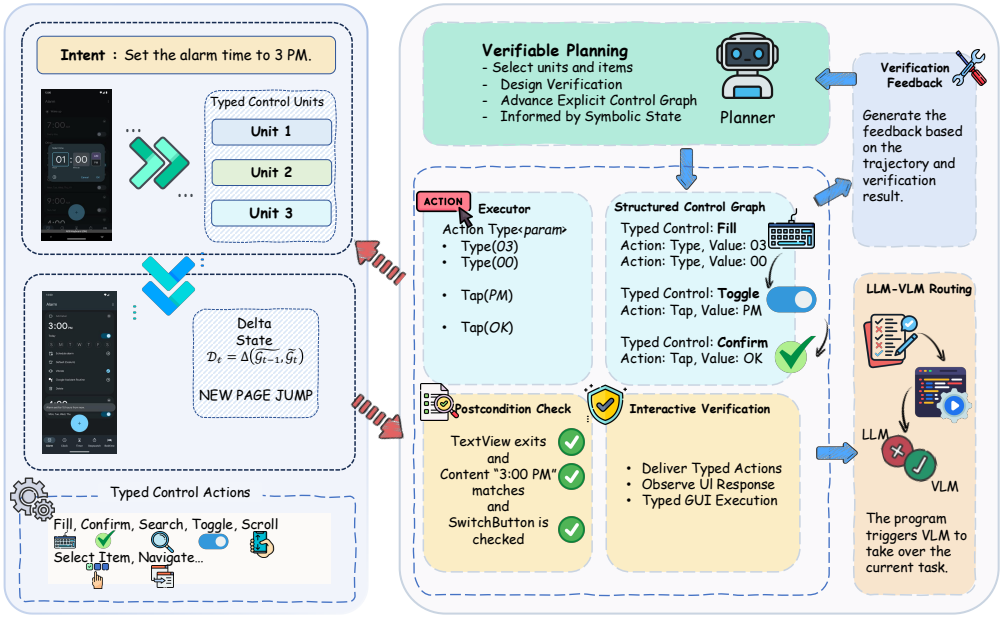


Figure 1: **System framework of SenseAct GUI agent.** SenseAct structures GUI interaction into explicit, typed control actions and verifiable planning primitives. Given a high-level goal, the agent first constructs a structured GUI action graph that abstracts raw UI perception into typed actions (e.g., click, fill, select, check). An LLM then advances an explicit control graph informed by the symbolic GUI state, producing a structured action plan. During execution, postcondition checks programmatically verify whether intended UI effects are realized. When symbolic execution becomes insufficient, a gated LLM–VLM routing mechanism selectively invokes visual reasoning to resolve localized perceptual ambiguity. Failure cases and repair trajectories are recorded in memory to support informed replanning.

tains a tightly coupled loop in which perception, decision making, and execution are linked through well-defined symbolic interfaces. Given a raw UI state, SenseAct first induces a control-relevant abstraction that exposes admissible control units and their hierarchical organization, defining what can be acted upon and at what granularity. Decision making then operates over these abstractions by synthesizing a structured control graph that encodes control intent, dependencies, and ordering of execution.

Each control primitive is executed and evaluated through type-specific post-condition predicates over symbolic state transitions, providing a deterministic criterion for success or failure. Verification outcomes directly constrain subsequent control construction and, when violated, trigger refinement of the symbolic abstraction. Through this design, perception defines the control interface, control structure governs execution, and execution semantics anchor planning updates, yielding a grounded and verifiable interaction loop.

### 3.1 Typed Control Units and Structured Control Graphs

SenseAct represents GUI interaction by explicitly constructing a page-level control structure, rather than treating the UI as a passive perceptual input for action generation. The key observation is that a UI page already constrains *what can be controlled* and *how control effects unfold* through its hierarchical organization and interaction affordances. SenseAct makes these constraints explicit by systematically transforming the raw XML UI tree into typed control units and organizing them into a Structured Control Graph (SCG), which captures the admissible control semantics of the current page under a given intention.

**Control-Preserving Reduction of the UI Hierarchy.** At each time step $t$, the UI is parsed into a hierarchical structure consisting of UI elements and their containment relations. While this hierarchy accurately reflects visual layout, it is not directly usable for control: many nodes are execution-irrelevant, whereas others participate in control only through their structural or semantic roles. We therefore apply a deterministic, control-preserving reduction that removes nodes incapable of influencing execution while retaining those that constrain or induce UI state transitions. This reduction defines the maximal subset of the UI tree that can serve as a control substrate for the current page.

**Control-Relevant Nodes.** We formalize this substrate by identifying *control-relevant nodes*.

**Definition 3.1** (Control-Relevant Node)**.** A UI node $v$ is control-relevant if it satisfies

$$\text{actionable}(v) \ \vee \ \text{semantic}(v) \ \vee \ \text{structural}(v),$$

where actionability indicates direct interactability, semantic relevance captures task-informative elements (e.g., labels or values), and structural salience reflects layout roles that constrain valid execution (e.g., scroll containers or list structures). We denote the resulting set as $\mathcal{V}_t^{\text{ctrl}}$.

This criterion is defined with respect to execution semantics rather than visual salience: a node is retained if and only if its presence can change, constrain, or disambiguate the outcome of a control decision on the current page.

**Typing Interactive Nodes by Affordance.** Each control-relevant node $v \in \mathcal{V}_t^{\text{ctrl}}$ is then lifted into a typed interactive node $\mathcal{N}$ is defined as the set of argumented nodes with each item given as $(v, \tau_n)$, where $\tau_n \in \mathcal{T}_{\text{node}}$ is determined by the node's interaction affordances (e.g., clickable, scrollable, editable) and specifies the admissible primitive interactions supported by the node. The resulting set $\mathcal{N}_t$ constitutes the atomic control alphabet of the page. This typing step abstracts away presentation-specific details and exposes a stable interface between UI perception and the construction of control action sequences.

**Aggregation into Typed Control Units.** Atomic interactions at the node level are often insufficient to characterize execution semantics, as many UI effects arise only when multiple elements are operated jointly (e.g., selecting an item within a list or modifying a compound setting). To align representation granularity with execution outcomes, SenseAct aggregates typed interactive nodes into *typed control units*.

A typed control unit is defined as:

$$u = (S_i, \tau_i), \quad S_i \subseteq \mathcal{N}_t, \ S_i \neq \emptyset, \tag{1}$$

where t is the timestep , $i$ is the control unit id and $S_i$ is a set of typed interactive nodes whose joint satisfaction is required to realize a coherent interface effect, and $\tau_i \in \mathcal{T}_{\text{unit}}$ specifies the control operator type. Control units are constructed such that executing any strict subset of $S_i$ fails to induce the intended semantic outcome, enforcing execution-level atomicity. The control type $\tau_i$ further determines a predictable class of symbolic UI state transitions, establishing a stable link between control intent and observable execution effects. We denote the resulting set of typed control units as $\mathcal{U}_t$. Details on the algorithm is presented in Algorithm 1.

**Structured Control Graph Construction.** Given the set of typed control units $\mathcal{U}_t$ and a high-level intention $g$, SenseAct constructs a Structured Control Graph that encodes the admissible control structure of the current UI page.

**Definition 3.2** (Structured Control Graph)**.** The Structured Control Graph (SCG) at step $t$ is defined as:

$$\text{SCG}_t(g) = (\mathcal{U}_t, \ \mathcal{E}_t, \ \mathcal{T}, \ \Delta),$$

where $\mathcal{U}_t$ is the set of typed control units, $\mathcal{E}_t \subseteq \mathcal{U}_t \times \mathcal{U}_t$ is a directed dependency relation encoding admissible control orderings, $\mathcal{T}$ is the control type system, and $\Delta$ is a type-indexed symbolic state transition model. For each control type $\tau$, $\Delta_\tau$ specifies the expected class of UI state changes induced by executing any control unit of type $\tau$.

The SCG does not prescribe a single action sequence. Instead, it defines the space of admissible control progressions under the current UI and intention, making explicit which control commitments are valid, which depend on prior effects, and how execution outcomes should be interpreted.

**Execution Interface.** At runtime, the SCG induces a dynamically evolving execution frontier consisting of control units whose dependency constraints have been satisfied. Execution outcomes are evaluated against the expected state transitions specified by $\Delta$, and verified transitions enable further traversal of the graph. This formulation grounds control in explicit execution semantics rather than implicit language-based judgment. The verification mechanism governing state transition checking is described in the next section.

### 3.2 Structural Planning with Programmatic Verification

Section 3.1 lifts the raw XML UI hierarchy into a Structured Control Graph (SCG) that explicitly encodes *what can be controlled*, *how control effects compose*, and *what symbolic state changes* each control type is expected to induce. Given this SCG, SenseAct reformulates planning as *control synthesis under an explicit semantic contract*, rather than action prediction or language-mediated reasoning. As a result, planning correctness is defined prior to execution.

**Planning over an explicit control contract.** At interaction step $t$, planning is performed over $\mathrm{SCG}_t(g) = (\mathcal{U}_t, \mathcal{E}_t, \mathcal{T}, \Delta)$ constructed for the current UI page and screen-scoped intention $g$. Instead of reasoning over pixels or low-level actions, the planner operates on typed control units $u \in \mathcal{U}_t$, whose admissible orderings are constrained by $\mathcal{E}_t$ and whose semantic effects are specified by the type system $\mathcal{T}$ and transition model $\Delta$. In this sense, the SCG acts as a control contract that bounds all planning decisions.

**Control commitment as the planning decision.** A planning step produces a *control commitment*, defined as a finite, ordered sequence of typed control units,

$$c_t = (u_{t,1}, \ldots, u_{t,k}), \qquad u_{t,k} \in \sqcap_t, \tag{2}$$

representing a screen-local commitment to realizing specific control effects. This shifts planning from deciding *what action to execute next* to deciding *which control effects the agent commits to making true* on the current page.

**Structural admissibility.** The SCG constrains admissible commitments through dependency edges $\mathcal{E}_t$. Let $\mathrm{Pred}(u)$ denote the prerequisite set of a control unit $u$. A commitment $c_t$ is structurally admissible if it respects the partial order imposed by the SCG:

$$\forall k,\ \forall u \in \mathrm{Pred}(u_{t,k}), \quad u \in \{u_{t,1}, \ldots, u_{t,k}\}. \tag{3}$$

This condition enforces consistency with page-level control semantics, independent of execution outcomes or language rationales.

**Type-conditioned semantic target.** Each control unit $U$ carries a control type $\tau(u) \in \mathcal{T}$. The type system specifies an expected class of symbolic UI state transitions via $\Delta(\cdot, \cdot)$. Given the current symbolic UI state $s_t$, executing $u$ is expected to induce a state change consistent with:

$$\Delta_{\tau(u)}(s_{t-1}, s_t), \tag{4}$$

where $\Delta_{\tau(u)}$ is the transation from $s_{t-1}$ to $s_t$ and which defines *what must change* in the UI for the control to be considered successful. Crucially, this semantic target is fixed by the control type and is therefore part of the planning decision itself.

**Programmatic verification over SCG contracts.** Planning in SenseAct is verifiable because each planned control unit carries an explicit *semantic contract*—a type-conditioned expectation over symbolic UI state transitions—that is fixed *before* execution. For any commitment $c_t = (u_{t,1}, \ldots, u_{t,K})$, admissibility is ensured by the SCG dependency constraints (Eq. 3), and the intended effect of each unit is specified by the type-indexed transition model $\Delta_{\tau(u_{t,k})}$ (Eq. 4). Consequently, correctness reduces to verifying whether the observed state delta matches the pre-specified contract, without relying on language-based self-evaluation.

We operationalize this contract via a type-indexed programmatic verification operator that evaluates structured predicates on the *compressed, group-level* UI substrate and the observed transition. Let $\tilde{\mathcal{G}}_t$ denote the compressed UI graph (with grouped units) and let $\delta_t := (s_t \rightarrow s_{t+1})$ denote the observed symbolic state transition. We define a contract-driven verifier:

$$\begin{aligned} \mathrm{Verify}_{\mathrm{prog}}(c_t; \tilde{\mathcal{G}}_t, \delta_t) \;&:=\; \bigwedge_{k=1}^{K} \mathcal{P}_{\tau(u_{t,k})}\Big(\tilde{\mathcal{G}}_t, \delta_t; \Delta_{\tau(u_{t,k})}\Big) \\ &\in\; \{0,1\}, \end{aligned} \tag{5}$$

where $\mathcal{P}_\tau(\cdot)$ is a deterministic, type-specific predicate family instantiated from the contract $\Delta_\tau$ (e.g., target existence and uniqueness, exact/semantic text match, enabledness, and state-flag consistency). Note that we do not require $\mathcal{P}_{\tau(u_{t,k})}$ to be complete; inconclusive verification is treated as an explicit outcome that triggers controlled fallback rather than being silently accepted as success. Only when $\mathrm{Verify}_{\mathrm{prog}}$ is inconclusive due to missing or ambiguous symbolic evidence do we defer to an LLM-based verifier to resolve residual ambiguity; this preserves a "programmatic-first" verification policy.

### 3.3 Execution Feedback-Guided Replanning

The final verification outcome is converted into structured *feedback certificates* that are injected back into the planner and SCG construction, enabling closed-loop replanning under verification guidance.

**Feedback certificate generation.** To stabilize planning under partial observability and noisy UI dynamics, we convert execution outcomes into a structured *feedback certificate* that can be reused across steps and tasks. Given the pre- and post-execution compressed UI graphs $\tilde{\mathcal{G}}_{t-1}$ and $\tilde{\mathcal{G}}_t$, we compute a symbolic delta summary by matching nodes via stable fingerprints (e.g., resource identifiers, sanitized XPaths, and bounds) and extracting atomic attribute changes (text edits and interaction-state flips):

$$\mathcal{D}_t = \Delta(\tilde{\mathcal{G}}_{t-1}, \tilde{\mathcal{G}}_t). \tag{6}$$

To capture macro-level transitions (e.g., page jumps), we compute a match ratio:

$$\rho_t = \frac{\Big|M(\tilde{\mathcal{G}}_{t-1}, \tilde{\mathcal{G}}_t)\Big|}{\max\Big(|\tilde{V}_{t-1}|, 1\Big)}, \tag{7}$$

where $M(\cdot,\cdot)$ is the set of matched nodes and $\tilde{V}_{t-1}$ is the node set of $\tilde{\mathcal{G}}_{t-1}$ and assign a transition label $z_t \in \{\textsc{NoChange}, \textsc{PartialChange}, \textsc{NewPageJump}\}$ using fixed thresholds. The tuple,

$$\mathsf{Cert}_t := (g_t,\ a_t,\ z_t,\ \mathcal{D}_t)$$

constitutes an execution feedback certificate, where $g_t$ is the current intent and $a_t$ is the executed trace. We then use an LLM only as a *semantic compressor* to map $\mathsf{Cert}_t$ into a concise, structured certificate message in a restricted DSL (serialized to JSON) for downstream consumption.

**Certificate memory and retrieval.** We maintain a certificate memory $\mathcal{M} = \{\mathsf{CertMsg}_j\}$ where $\mathsf{CertMsg}_j$ is structured certificate message and retrieve relevant certificate messages for a new intent $g$ by scoring similarity between $g$ and the message key:

$$\textsc{Retrieve}(g; \mathcal{M}) = \underset{\mathsf{CertMsg}_j \in \mathcal{M}}{\arg\operatorname{topk}} \ \textsc{Sim}\big(g, \textsc{Key}(\mathsf{CertMsg}_j)\big)\,. \tag{8}$$

The retrieved certificates are injected into the planner and SCG construction as explicit constraints or repair hints (e.g., "executed but no state change", "likely page jump", or "value not updated"), enabling replanning guided by verifiable execution evidence rather than free-form reflection.

### 3.4 Programmatic VLM Fallback via Execution Mismatch

SenseAct is designed to operate in a vision-free regime by default, relying on symbolic UI abstractions, structured control graphs, and programmatic verification to guide interaction without

**Algorithm 1:** SenseAct: Verifiable Control for GUI Interaction

**Input :** GUI state $\mathcal{U}_t$, task query $Q$, history buffer $\mathcal{H}$
**Output:** Executed GUI actions or terminal state

$\mathcal{A}_t \leftarrow$ EXTRACTACTIONS$(\mathcal{U}_t)$ ; // Typed GUI action abstraction
$\mathcal{G}_t \leftarrow$ BUILDCONTROLGRAPH$(\mathcal{A}_t, Q)$ ; // Actionable control graph
$\pi_t \leftarrow$ PLANCONTROL$(\mathcal{G}_t, \mathcal{U}_t)$ ; // LLM-based control synthesis
**foreach** $a \in \pi_t$ **do**
  EXECUTE$(a)$;
  $\mathcal{U}_{t+1} \leftarrow$ OBSERVESTATE();
  **if** $\neg$CHECKPOSTCONDITION$(a, \mathcal{U}_{t+1})$ **then**
    $\mathcal{S}_t \leftarrow$ ACTIONINTENTIONSCORE$(\mathcal{H})$;
    **if** $\mathcal{S}_t \geq \tau$ **then**
      $\mathcal{U}_{t+1} \leftarrow$ INVOKEVLM$(\mathcal{U}_t)$ ; // Perceptual fallback
    $\mathcal{H} \leftarrow \mathcal{H} \cup \{(a, \mathcal{U}_{t+1})\}$;
    **break**;
**if** GOALSATISFIED$(g, \mathcal{U}_{t+1})$ **then**
  **return** SUCCESS;
**else**
  **return** REPLAN;

pixel-level reasoning. However, symbolic control may become insufficient when execution repeatedly fails to induce the expected state transition despite pursuing the same control objective. Such failures typically arise from visually grounded ambiguities—e.g., subtle widget states, visually similar targets, or rendering-dependent affordances—that are not represented in symbolic descriptors. Crucially, these cases can be detected from execution evidence rather than from language-based uncertainty.

To handle this failure mode efficiently, we introduce a *programmatic fallback gate* that selectively invokes a vision-language model only when a control–execution mismatch is observed. Formally, given an executed action $a_t$ and the observed UI state delta $\Delta_t$, we define a routing trigger:

$$\mathcal{G}_{\text{vlm}}(a_t, \Delta_t) = \mathbb{I}[\text{act}(a_t) \wedge \neg\text{obs}(\Delta_t)], \tag{13}$$

which activates VLM-based perception only when an action is issued but its expected symbolic effect fails to materialize. Under this condition, the VLM is used solely for localized perceptual disambiguation, after which control returns to the standard LLM-based planning pipeline. This design treats vision as a *gated perceptual fallback* rather than a persistent reasoning component, preserving efficiency while providing visual grounding precisely when symbolic execution stalls.

# 4 EXPERIMENTS

## 4.1 EXPERIMENTAL SETUP

### 4.1.1 BENCHMARK DATASETS

We evaluate our method on two benchmark datasets for mobile agents: DroidTask Wen et al. (2024) and AndroidLab Xu et al. (2025). The DroidTask dataset consists of 158 tasks across 13 commonly used Android applications, including Calendar, Clock, and Dialer. After filtering out tasks with compatibility issues, we retain 143 tasks spanning 12 compatible apps. The AndroidLab dataset contains more challenging operation-level tasks in applications with complex and dynamic UI layouts, such as Maps, Music, and Zoom. We include all 138 operation tasks across 9 apps provided by AndroidLab, which require multi-step interaction and fine-grained UI reasoning.

## 4.2 EVALUATION METRICS

We evaluate agent performance along two complementary dimensions: *task success* and *reduction rate*.

| Methods | DroidTask | | AndroidLab | |
|---|---|---|---|---|
| | Success Rate | Reduction Rate | Success Rate | Reduction Rate |
| GPT-4o Baseline —— *VLM + LLM* | **74.13%** | 0.00% | **44.90%** | 0.00% |
| Gemma 2-9B Baseline —— *Text Only* | 32.87% | **100.00%** | 17.35% | **100.00%** |
| Qwen 2.5-7B Baseline —— *Text Only* | 14.69% | **100.00%** | 5.10% | **100.00%** |
| LLaMA 3.1-8B Baseline —— *Text Only* | 9.79% | **100.00%** | 11.22% | **100.00%** |
| CORE (GPT-4o [VLM] + Gemma 2-9B [LLM]) | 69.23% | 55.60% | 37.76% | 34.96% |
| CORE (GPT-4o [VLM] + Qwen 2.5-7B [LLM]) | 61.54% | 41.89% | 41.84% | 29.97% |
| CORE (GPT-4o [VLM] + LLaMA 3.1-8B [LLM]) | 49.65% | 40.56% | 34.69% | 31.65% |
| SenseAct (GPT-4o [LLM] —— *Text Only*) | 68.53% | **100%** | **49.28%** | **100%** |
| SenseAct (GPT-4o [VLM] + GPT-4o [LLM]) | **69.93%**[*] | **70.09%** | **53.62%** | **65.51%** |

[*] For DroidTask, the LLM-only and LLM+VLM scores are close because the reduction rate is high and VLM is rarely invoked.

Table 2: Comparison with state of the art baselines regarding task success rate and UI exposure reduction. **SenseAct** consistently outperforms the strong **CORE** baseline in VLM reduction by **14.49%** on DroidTask and **30.55%** on AndroidLab while simultaneously achieving superior success rates. Across both benchmarks, **SenseAct (LLM-only)** maintains competitive success rates, a result that underscores the inherent information density and effectiveness of our proposed symbolic UI representation. These findings validate that SenseAct effectively filters perceptual redundancy without compromising the semantic grounding required for complex mobile interactions.

**Task Success.** A task is considered successful if the agent completes the instruction and reaches a UI state that satisfies the task-specific success criteria. For DroidTask, task success is determined using the official evaluation scripts provided by the benchmark. For AndroidLab, we adopt an operation-level evaluation protocol, where a task is marked as successful only if all required operations are correctly executed in the prescribed order.

**Reduction Rate.** Reduction rate measures the extent to which an agent reduces visual perception overhead during task execution. Specifically, we define reduction rate as the relative reduction in the number of image uploads for vision-language model inference along a single task execution trajectory, compared to baseline methods. A higher reduction rate indicates more efficient interaction with reduced reliance on expensive visual inputs.

### 4.3 Baselines

We primarily compare SenseAct against CORE Fan et al. (2025), a recent mobile UI agent that aims to reduce visual perception costs by coordinating a cloud-based vision–language model with a local language model. CORE serves as our main baseline due to its close alignment with our goal of minimizing visual exposure while preserving task success.

Unless otherwise specified, all comparative results in this paper are reported against CORE under identical evaluation protocols. Additional models are included only in aggregate results to provide broader context on overall performance trends, rather than for detailed pairwise analysis.

### 4.4 Benchmark Results

The overall results are presented in Table 4.1.1, and we present the per-app scores for each benchmark in Table 4 from Appendix.

We evaluate SenseAct under both text-only and multimodal settings. In the text-only regime, SenseAct achieves higher task success on AndroidLab, indicating stronger symbolic reasoning and robustness in long-horizon tasks, while remaining competitive with the strongest baselines on DroidTask. In the multimodal setting, SenseAct attains the highest success rate on AndroidLab and maintains competitive performance on DroidTask, while substantially reducing visual perception usage. These results indicate that performance gains are achieved without increased reliance on visual inputs.

Table 3: **Ablation study on AndroidLab**. This result highlights the crucialness of incorporating continuous environment feedback within the verification loop.

| VARIANT | ANDROIDLAB (%) |
|---|---|
| SENSEACT | **49.28%** |
| W/O EXECUTION FEEDBACK | 20.83% |

**Overall Performance Characteristics.** As shown in Table 4.1.1, SENSEACT consistently balances task success and perception efficiency across both benchmarks. Unlike methods that depend on persistent visual grounding, SENSEACT demonstrates that high success rates can be achieved with significantly reduced UI exposure.

**Symbolic-only and Conditional Visual Settings.** Under the symbolic-only setting, **SenseAct (GPT-4o [LLM] —— Text Only)** substantially outperforms all text-only baselines on both DroidTask and AndroidLab, demonstrating that effective GUI control can be achieved without pixel-level observations. When augmented with conditional visual reasoning, **SenseAct (GPT-4o [VLM] + VLM [LLM])** further improves task success, achieving the best performance on AndroidLab while reducing UI exposure by 65.51Visual perception is invoked only when symbolic execution evidence is insufficient, enabling recovery from ambiguity without continuous visual processing.

**Efficiency–Performance Trade-off.** Across both benchmarks, SENSEACT occupies a favorable trade-off region, preserving most of the benefits of visual reasoning while significantly reducing perception overhead. These results validate the core design principle of SENSEACT: visual perception should serve as a targeted fallback rather than a default mode.

### 4.5 ABLATION STUDY

**Sampling-based evaluation protocol.** Due to the large combinatorial space of UI states and control commitments, the ablation experiments are conducted under a sampling-based evaluation protocol. At each trial, tasks, initial UI states, and control commitments are sampled from the corresponding benchmark distributions, rather than being exhaustively enumerated. This design enables scalable and statistically meaningful evaluation while preserving representative execution diversity.

**Effect of Verification and Insights.** Table 3 reports the ablation results on AndroidLab. Removing the verification–insight module leads to a clear degradation in task success, indicating that execution-level feedback plays an essential role in SenseAct. Without explicit verification signals and state-change insights, the agent is unable to reliably detect execution failures or ambiguous UI transitions, resulting in error accumulation in long-horizon tasks. These results confirm that verification and insights are critical for grounding symbolic reasoning in observable UI evidence, rather than relying solely on intent-level planning.

## 5 CONCLUSION

This paper revisits GUI interaction from a control-centric perspective and argues that the primary bottleneck lies not in perception, but in the absence of explicit, verifiable control semantics. We formulate interaction as control synthesis over a *Structured Control Graph (SCG)* of typed control units, with execution governed by type-conditioned post-condition verification. This decouples perception, control construction, and execution, while linking them through a shared symbolic interface that makes control commitments explicit and their effects programmatically checkable. We further introduce memory-driven feedback certificates that encode execution outcomes as structured evidence, enabling grounded replanning without heuristic reflection. Empirically, this formulation reduces execution drift and improves success rates on long-horizon tasks. More importantly, it points to a shift in agent design: reliability arises not from better action generation, but from structuring control and enforcing verifiable execution. Overall, our results suggest that reliable agent systems require elevating control structure, execution semantics, and verification feedback to first-class abstractions, offering a complementary path to model scaling through explicit, verifiable interfaces.

## ACKNOWLEDGMENTS

This research is supported by the National Research Foundation, Singapore under its AI Singapore Programme (AISG Award No: AISG-NMLP-2024-003), and the National Research Foundation, Singapore and Infocomm Media Development Authority under its Trust Tech Funding Initiative. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not reflect the views of the National Research Foundation, Singapore, the Agency for Science, Technology and Research, or the Infocomm Media Development Authority.

## REFERENCES

Kanzhi Cheng, Qiushi Sun, Yougang Chu, Fangzhi Xu, Yantao Li, Jianbing Zhang, and Zhiyong Wu. Seeclick: Harnessing GUI grounding for advanced visual GUI agents. *CoRR*, abs/2401.10935, 2024. doi: 10.48550/ARXIV.2401.10935. URL https://doi.org/10.48550/arXiv.2401.10935.

Gaole Dai, Shiqi Jiang, Ting Cao, Yuanchun Li, Yuqing Yang, Rui Tan, Mo Li, and Lili Qiu. Advancing mobile GUI agents: A verifier-driven approach to practical deployment. *CoRR*, abs/2503.15937, 2025. doi: 10.48550/ARXIV.2503.15937. URL https://doi.org/10.48550/arXiv.2503.15937.

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 Krzysztof Gajos, Jennifer Mankoff, and Chris Harrison (eds.), *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology, UIST 2017, Quebec City, QC, Canada, October 22 - 25, 2017*, pp. 845–854. ACM, 2017. doi: 10.1145/3126594.3126651. URL https://doi.org/10.1145/3126594.3126651.

Gucongcong Fan, Chaoyue Niu, Chengfei Lyu, Fan Wu, and Guihai Chen. CORE: Reducing UI exposure in mobile agents via collaboration between cloud and local LLMs. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.

Hongliang He, Wenlin Yao, Kaixin Ma, Wenhao Yu, Yong Dai, Hongming Zhang, Zhenzhong Lan, and Dong Yu. Webvoyager: Building an end-to-end web agent with large multimodal models. *CoRR*, abs/2401.13919, 2024. doi: 10.48550/ARXIV.2401.13919. URL https://doi.org/10.48550/arXiv.2401.13919.

Jing Yu Koh, Robert Lo, Lawrence Jang, Vikram Duvvur, Ming Chong Lim, Po-Yu Huang, Graham Neubig, Shuyan Zhou, Ruslan Salakhutdinov, and Daniel Fried. Visualwebarena: Evaluating multimodal agents on realistic visual web tasks. *CoRR*, abs/2401.13649, 2024. doi: 10.48550/ARXIV.2401.13649. URL https://doi.org/10.48550/arXiv.2401.13649.

Kaixin Li, Ziyang Meng, Hongzhan Lin, Ziyang Luo, Yuchen Tian, Jing Ma, Zhiyong Huang, and Tat-Seng Chua. Screenspot-pro: GUI grounding for professional high-resolution computer use. *CoRR*, abs/2504.07981, 2025. doi: 10.48550/ARXIV.2504.07981. URL https://doi.org/10.48550/arXiv.2504.07981.

Yang Li, Gang Li, Luheng He, Jingjie Zheng, Hong Li, and Zhiwei Guan. Widget captioning: Generating natural language description for mobile user interface elements. *CoRR*, abs/2010.04295, 2020. URL https://arxiv.org/abs/2010.04295.

Yuyang Li, Yiran Dou, Jie-Jing Shao, Yueming Lyu, Ivor Tsang, and Haiyan Yin. Skilltracer: Structural failure attribution and refinement of agentic skills in long-horizon web tasks. In *Proceedings of the ICLR 2026 Workshop on Multi-Agent Learning and Its Opportunities in the Era of Generative AI (MALGAI)*, 2026. URL https://openreview.net/forum?id=OiyEjThGeZ.

Zhangheng Li, Keen You, Haotian Zhang, Di Feng, Harsh Agrawal, Xiujun Li, Mohana Prasad Sathya Moorthy, Jeff Nichols, Yinfei Yang, and Zhe Gan. Ferret-ui 2: Mastering universal user interface understanding across platforms. *CoRR*, abs/2410.18967, 2024. doi: 10.48550/ARXIV.2410.18967. URL https://doi.org/10.48550/arXiv.2410.18967.

Tao Liu, Chongyu Wang, Rongjie Li, Yingchen Yu, Xuming He, and Song Bai. Gui-rise: Structured reasoning and history summarization for GUI navigation. *CoRR*, abs/2510.27210, 2025a. doi: 10.48550/ARXIV.2510.27210. URL https://doi.org/10.48550/arXiv.2510.27210.

Xinyi Liu, Xiaoyi Zhang, Ziyun Zhang, and Yan Lu. Ui-e2i-synth: Advancing GUI grounding with large-scale instruction synthesis. *CoRR*, abs/2504.11257, 2025b. doi: 10.48550/ARXIV.2504.11257. URL https://doi.org/10.48550/arXiv.2504.11257.

Yifei Lu, Fanghua Ye, Jian Li, Qiang Gao, Cheng Liu, Haibo Luo, Nan Du, Xiaolong Li, and Feiliang Ren. Codetool: Enhancing programmatic tool invocation of llms via process supervision. *CoRR*, abs/2503.20840, 2025. doi: 10.48550/ARXIV.2503.20840. URL https://doi.org/10.48550/arXiv.2503.20840.

Christopher Rawles, Sarah Clinckemaillie, Yifan Chang, Jonathan Waltz, Gabrielle Lau, Marybeth Fair, Alice Li, William E. Bishop, Wei Li, Folawiyo Campbell-Ajala, Daniel Toyama, Robert James Berry, Divya Tyamagundlu, Timothy P. Lillicrap, and Oriana Riva. Androidworld: A dynamic benchmarking environment for autonomous agents. *CoRR*, abs/2405.14573, 2024. doi: 10.48550/ARXIV.2405.14573. URL https://doi.org/10.48550/arXiv.2405.14573.

Bryan Wang, Gang Li, Xin Zhou, Zhourong Chen, Tovi Grossman, and Yang Li. Screen2words: Automatic mobile UI summarization with multimodal learning. *CoRR*, abs/2108.03353, 2021. URL https://arxiv.org/abs/2108.03353.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *CoRR*, abs/2305.16291, 2023. doi: 10.48550/ARXIV.2305.16291. URL https://doi.org/10.48550/arXiv.2305.16291.

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, 2024.

Keyi Xiang, Tianyi Tang, Jie-Jing Shao, Yueming Lyu, Ivor Tsang, Yew-Soon Ong, and Haiyan Yin. Nesyproact: Proactive neural-symbolic control for web agents. In *Proceedings of the ICLR 2026 Workshop on Agents in the Wild (AIWILD)*, 2026. URL https://openreview.net/forum?id=9xteuRjHr7.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, Yitao Liu, Yiheng Xu, Shuyan Zhou, Silvio Savarese, Caiming Xiong, Victor Zhong, and Tao Yu. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments. *CoRR*, abs/2404.07972, 2024. doi: 10.48550/ARXIV.2404.07972. URL https://doi.org/10.48550/arXiv.2404.07972.

Yifan Xu, Xiao Liu, Xueqiao Sun, Siyi Cheng, Yun-Hao Liu, Hanyu Lai, Shudan Zhang, Dan Zhang, Jie Tang, and Yuxiao Dong. Androidlab: Training and systematic benchmarking of android autonomous agents. *CoRR*, abs/2410.24024, 2024. doi: 10.48550/ARXIV.2410.24024. URL https://doi.org/10.48550/arXiv.2410.24024.

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

Pei Yang, Hai Ci, and Mike Zheng Shou. macosworld: A multilingual interactive benchmark for GUI agents. *CoRR*, abs/2506.04135, 2025. doi: 10.48550/ARXIV.2506.04135. URL https://doi.org/10.48550/arXiv.2506.04135.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. *CoRR*, abs/2210.03629, 2022. doi: 10.48550/ARXIV.2210.03629. URL https://doi.org/10.48550/arXiv.2210.03629.

Keen You, Haotian Zhang, Eldon Schoop, Floris Weers, Amanda Swearngin, Jeffrey Nichols, Yinfei Yang, and Zhe Gan. Ferret-ui: Grounded mobile UI understanding with multimodal llms. *CoRR*, abs/2404.05719, 2024. doi: 10.48550/ARXIV.2404.05719. URL https://doi.org/10.48550/arXiv.2404.05719.

Xiaoxue Zang, Ying Xu, and Jindong Chen. Multimodal icon annotation for mobile applications. *CoRR*, abs/2107.04452, 2021. URL https://arxiv.org/abs/2107.04452.

Chi Zhang, Zhao Yang, Jiaxuan Liu, Yucheng Han, Xin Chen, Zebiao Huang, Bin Fu, and Gang Yu. Appagent: Multimodal agents as smartphone users. *CoRR*, abs/2312.13771, 2023. doi: 10.48550/ARXIV.2312.13771. URL https://doi.org/10.48550/arXiv.2312.13771.

Chengqi Zheng, Jianda Chen, Yueming Lyu, Wen Zheng Terence Ng, Haopeng Zhang, Yew-Soon Ong, Ivor Tsang, and Haiyan Yin. Mermaidflow: Redefining agentic workflow generation via safety-constrained evolutionary programming, 2025. URL https://arxiv.org/abs/2505.22967.

Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. Webarena: A realistic web environment for building autonomous agents. *CoRR*, abs/2307.13854, 2023. doi: 10.48550/ARXIV.2307.13854. URL https://doi.org/10.48550/arXiv.2307.13854.

## A Formal Definitions and Theoretical Foundations

**Layout-aware Block Formation.** Algorithm 1 defines a layout-aware transformation from a raw UI XML hierarchy to a compact structured representation. Let the original UI hierarchy be represented as an XML tree $\mathcal{T} = (V, E)$, where each node $v \in V$ corresponds to a UI element and each edge $e \in E$ denotes a parent–child relation. The algorithm applies a sequence of sparsification and merge operations to remove irrelevant nodes and normalize interaction attributes, resulting in a reduced tree $\tilde{\mathcal{T}} = (\tilde{V}, \tilde{E})$.

The reduced tree $\tilde{\mathcal{T}}$ induces a partition of the UI into $k = |\mathcal{B}|$ layout-consistent blocks $\mathcal{B} = \{b_1, b_2, \ldots, b_k\}$, where each block $b_i$ is defined as a maximal connected subtree rooted at a retained parent node after the merge stage. These blocks form a structured abstraction of the original UI layout and serve as the atomic units for subsequent symbolic control graph (SCG) construction.

**Algorithm 1** Layout-aware Block Formation via XML Processing

**Input:** Raw XML string `xml`
**Params:** `level` $\in \{1, 2\}$, `remove_system_bar`, `use_bounds`, `merge_switch`
**Output:** Processed UI tree $T$ with node ids `nxxxx`
$root \leftarrow$ ParseXML(`xml`)
Initialize `mapCount`, `node_to_xpath`, `node_to_name`
Set flags: `use_bounds`, `merge_switch`, `remove_system_bar`
**if** `level` $\geq 1$ **then**
  XML_Sparse($root$)
**end if**
**if** `level` $\geq 2$ **then**
  Merge_None_Act($root$)
**end if**
Reindex_And_Pack($root$)
**return** $root$

**Contract-driven Execution Verification.** SenseAct formalizes GUI interaction as a sequence of verifiable control commitments governed by explicit semantic contracts. As illustrated in Figure 2, the process begins with a high-level intent $g$, from which the agent induces a typed control unit $u_t$ (with type $\tau(u_t) \in \mathcal{T}$) based on the Structured Control Graph (SCG). This unit declares a semantic control contract prior to execution, specifying an expected transition class $\Delta_{\tau(u_t)}$ that defines the intended symbolic state change. After the agent executes $u_t$ and observes the actual symbolic transition $\delta_t$, a programmatic verification operator $\varphi_{\tau(u_t)}$ evaluates the observed delta against the pre-specified contract. This deterministic check yields an outcome of success, failure, or inconclusive ($\perp$). If a contract violation or ambiguity is detected, the feedback is used to trigger replanning or refinement of the control unit. Furthermore, if visual ambiguity arises during execution—where symbolic transitions fail to materialize as expected—the system triggers a VLM fallback to resolve the discrepancy through localized pixel-level reasoning

**DSL for Verification Feedback.** To formally represent the execution outcomes and diagnostic evidence gathered during interaction, SenseAct utilizes a Domain-Specific Language (DSL) for Verification Feedback Generation. Every Feedback instance within the SenseAct framework strictly adheres to a structured format designed to capture the causal relationship between agent actions and observable UI state transitions. This representation ensures that replanning is grounded in verifiable evidence rather than free-form linguistic reflection:

- $.Scope(Context)$:: Defines the functional area or specific page (e.g., "Alarm Settings") where the interaction occurs.
- $.State(UIState)$:: Specifies the static pre-condition, representing the UI state that must exist prior to the action, using abstract roles (e.g., "search icon", "editable text field").
- $Prereq(Dependency)$ : Details dynamic pre-conditions, including historical dependencies (e.g., "User Logged In") or the confirmed existence of specific UI elements.
- $Action(Input)$: The core "Pivot Action" or control unit executed by the agent to trigger a state transition.

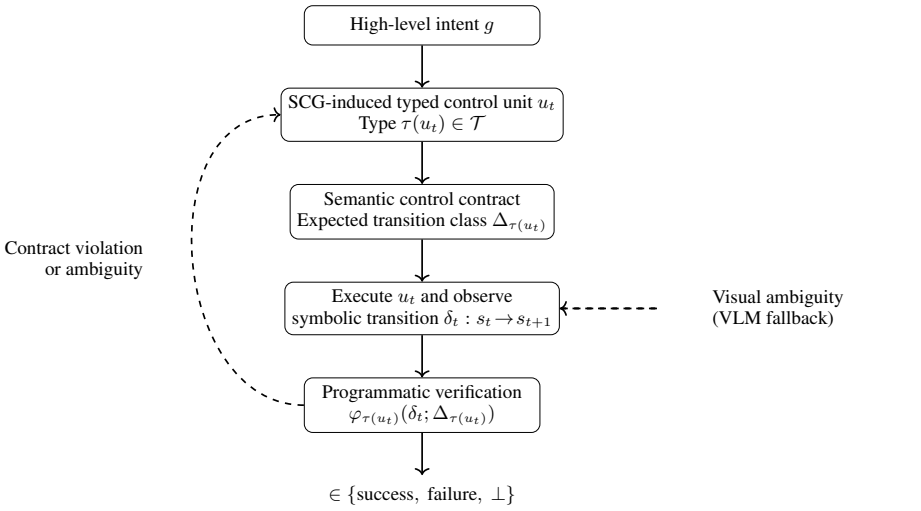


Figure 2: **Contract-driven execution verification.** A typed control unit induced by the Structured Control Graph (SCG) declares a type-indexed semantic contract $\Delta_\tau$ before execution. Observed symbolic state transitions are programmatically verified against this contract via $\varphi_\tau$, grounding execution correctness in explicit control semantics rather than language-based self-judgment.

- $Effect(\text{Delta})$ : The observed outcome following execution, including symbolic state deltas ($\mathcal{D}_t$), UI transitions (e.g., NEWPAGEJUMP), or error feedback.
- $Confidence(Level)$ : A causal certainty score—High for explicit causality (e.g., programmatic success) or Low for inferred or black-box behaviors where visual reasoning may be required.

## B Prompting and Templates

**Prompt 1: Intent Planner fontupper**

**Role.** You are an *Intent Planner* for a mobile GUI agent. Given the global task, execution history, and the current UI state, you must generate *exactly one* next intent that is achievable from the *current UI screen*, together with an expectation describing observable UI evidence after execution.
You do *not* output low-level actions (e.g., tap, click, scroll).
**Inputs.** The planner receives four inputs: (1) a natural language description of the global task; (2) an intent history recording previously attempted screen-scoped intents and their execution outcomes; (3) UI functional groups extracted from the current screen, where each group contains typed UI elements with identifiers and visible attributes; and (4) weak execution insights summarizing recent UI changes, verification results, and missing information.
**Screen Scope and State Identity.** Each intent must be strictly scoped to the *current UI screen*. The resulting UI state may navigate to a different screen, but the intent itself must describe a capability available on the current screen.
The notion of *same UI state* refers to the same screen surface (e.g., same page/activity with similar visible structure or key texts). If exact state identity is unavailable, intent history should be treated as weak but mandatory guidance for avoiding repetition.
**Intent Progression and Anti-Repeat.** Intent history must be used to avoid repeating intents of the same semantic type (e.g., SEARCH, FILTER, OPEN_SEARCH) on the same UI state when such attempts did not produce new evidence or state changes. If repetition is detected, the planner must change strategy (e.g., CREATE, ADD, or NAVIGATE), or terminate exploration if no capability remains.

**Capability Priority.** When multiple high-level UI entries are available, the planner must select the option that most directly satisfies the global task. In ambiguous cases, per-item or per-entry views are preferred over global summaries.
If required intent values are not explicitly confirmed on the current UI, the planner must prefer *evidence-surfacing* or *navigation* intents over terminal or commit-like intents. This prioritization is internal and must not be exposed in the output.
**Search Constraints (Hard Rule).** Search is treated as a secondary capability. A search intent may be generated *only* when there is explicit evidence that searchable target items already exist on the current UI (e.g., a visible list/grid of items, visible text indicating existing entries, or prior successful creation inferred from insights).
If no such evidence exists, the planner must not generate a search intent and must instead prefer: (1) CREATE / ADD a new entry, (2) OPEN a page or section that enables creation, or (3) NAVIGATE to an appropriate entry-point page.
This rule overrides keyword matching in the task description.
**Use of Insights.** Execution insights provide weak but mandatory evidence for decision making. The planner must use insights to: (i) determine whether the previous intent caused a UI change, (ii) identify missing required entities or values, and (iii) decide whether to continue exploration or terminate planning. Insights must not be ignored when determining FINISH or continuation.
**Expectation Specification.** Each intent must be accompanied by an expectation that specifies verifiable UI evidence observable *after* execution. Expectations describe only newly introduced or changed UI elements and must not rely on UI facts already visible before execution.
**Output Format.** The output must be a single JSON object with exactly two fields: `intent` and `expectation`. The expectation is defined as a logical composition (`AND` / `OR`) of one or more atomic UI conditions using predefined UI check types. Any deviation from this structure is considered invalid.

Prompt 2: SCG-Plan

**Role (HARD).** You are a *strict SCG-Hint Verifier* for a mobile GUI agent. Your **only** responsibility is to decide whether exploration on the *current UI surface* should **STOP** or **CONTINUE**. You **must not** judge task success, correctness, or generate actions.
**Core Semantics (HARD).** `FINISH` is *not* a success signal. It only indicates that the current exploration context is closed and the executor should stop acting on this UI surface. A **STOP** decision *must* explicitly recommend `FINISH` as the next step. A **CONTINUE** decision *must not* recommend `FINISH`.
**Inputs (HARD).** You receive: (1) the current intent, (2) the post-action UI state (compressed XML / LLM tree), (3) the executed action trace (with bounds if available), and (4) a program-generated state delta summary. You must base decisions **only** on these inputs. You may **not** invent UI elements, states, actions, or values.
**Surface Definition (HARD).** A "UI surface" is the current interaction context (screen, dialog, sheet, permission prompt, or overlay). The surface is considered the **same** if the visible structure and key affordances remain consistent per the delta summary (e.g., same title/header region, same primary widgets, no context switch signals).
**Editable vs. Sealed (HARD).** A surface is *editable* if any editing affordances exist, including any of: text inputs, toggles, add/edit/delete buttons, selection controls, or any widget with executable actions. A surface is *sealed* only if it is clearly read-only (e.g., confirmation/receipt/summary) and **no** editing affordances remain. **When uncertain, default to editable.**
**Required Values (HARD).** Any concrete user-specified value mentioned in the intent is a *required value*. A value is considered *present* only if it appears as actual content in the current UI state (not just as a label, hint, or placeholder). **Value presence alone is insufficient to STOP if the surface remains editable.**
**Global Decision Gate (MUST RUN FIRST).** Decide **STOP** if any of the following holds:

[leftmargin=1.2em]

1. **Context Switch:** the UI surface changed (new screen/dialog/sheet), or a new interaction context is reached (per delta summary or clear structural shift).
2. **Commit → Sealed:** a commit-like action (e.g., Save/Done/Confirm/Apply/Join) was executed and the resulting surface is sealed (non-editable).
3. **Exhaustion on Same Surface:** exploration on the same surface is exhausted: repeated interactions produce **no observable UI delta** and **no further evidence** can be surfaced.

Otherwise, decide **CONTINUE**.

**No-Effect Exhaustion (HARD, BOUNDS-BASED).** If the trace shows repeated interactions targeting the **same UI region** (high overlap in bounds, e.g., overlap $\geq 0.85$) and the delta summary indicates **no UI change**, this constitutes a strong no-effect signal.

[leftmargin=1.2em]

- If the surface is **sealed/non-editable**, this no-effect signal **triggers STOP**.
- If the surface is **editable**, you must **CONTINUE** but mark `progress="stuck"` and require a **strategy change**, rather than repeating the same interaction region.

**Verifier–Executor Responsibility Split (HARD).** The verifier decides whether exploration should STOP or CONTINUE. Concrete termination actions (e.g., same-bounds FINISH rules) are enforced by the downstream executor and planner.

**Stop vs. Continue Invariants (HARD).**

[leftmargin=1.2em]

- You must **never** output `finish_ok=true`. Always output `finish_ok=false`.
- **STOP** must include an explicit instruction that the next step should `FINISH`.
- **CONTINUE** must include at least one evidence-based reason indicating why further exploration is still possible.

**Output Format (STRICT).** Output exactly one JSON object:

```
{
  "decision": "STOP" | "CONTINUE",
  "finish_ok": false,
  "progress": "progressing" | "stuck" | "unknown",
  "reason": {
    "rule": "<triggered_rule_id>",
    "summary": "<short evidence-based explanation>",
    "evidence": {
      "surface_same": true|false,
      "surface_editable": true|false,
      "delta": "<quoted or paraphrased delta summary>",
      "repeated_bounds": true|false,
      "required_values_present": [ ... ],
      "required_values_missing": [ ... ]
    },
    "next_step": "FINISH" | "CONTINUE_EXPLORATION"
  }
}
```

## Prompt 3: Verify

**Role.** You are a *strict SCG-Hint Verifier* for a mobile GUI agent. Your only responsibility is to decide whether exploration on the *current UI surface* should **STOP** or **CONTINUE**. You do *not* decide task success and do *not* judge correctness.

**Stop/Continue Semantics. STOP** means the next UIDAG construction round should output a `FINISH`-only UIDAG. **CONTINUE** means the system should attempt another SCG step on the *same* UI surface. **FINISH is not a success signal**; it only closes the current exploration context.

Exploration must continue as long as the UI surface is unchanged and (i) additional evidence can still be surfaced, or (ii) required intent values are unconfirmed, or (iii) the context remains editable. Exploration must stop only when a surface/context boundary is reached, or the surface is sealed (read-only/committed), or exploration is exhausted.

**Inputs (must rely only on these).**

- `INTENT`: the intent to verify.
- `COMPRESSED_XML_AFTER_ACTION`: current UI state after the last action.
- `ACTION_TRACE`: executed actions so far.
- `STATE_DELTA`: program-generated UI delta summary.
- Optional: `VERIFY_CHECKED`, `VERIFY_OPERATOR`, `VERIFY_PASSED`.

You must not invent UI states, elements, or actions.

**Global Output Constraints (absolute).**

- Always output `finish_ok = false`.

- Never output `success = "SUCCESS"`.
- If you decide **STOP**, `reason.evidence` must include the exact substring: `"Next step should FINISH"`.

**Helpers: Required Values.** Extract `REQUIRED_VALUES` from `INTENT` (user-provided concrete strings/numbers/emails). Define `VALUE_PRESENT(v)` to be true iff `v` appears as actual content in `COMPRESSED_XML_AFTER_ACTION` (labels/placeholders do not count). Case-insensitive match is allowed; phone numbers ignore spaces/hyphens.
**Helpers: Editability and Sealing (hard repair).** *Value presence alone is not sufficient to stop.* If the UI is still editable (any edit affordance exists), you must continue.
Define `EDITING_ACTIVE` true iff any holds: (A) focused input / caret / recent typing context; or (B) editable affordances exist, including: keywords in text/content-desc/resource-id (e.g., edit/modify/change/add/remove/delete/clear/new/create), or input-like widgets (EditText/TextInput/AutoComplete/Spinner/CheckBox/Switch/SeekBar). If `EDITING_ACTIVE` is true, the surface is *not sealed*.
Define `READ_ONLY_SURFACE_HINT` true iff the UI resembles a summary/review page (no input widgets) and lacks edit controls, or `STATE_DELTA` indicates a commit led to a non-editable page. If unsure, default to `EDITING_ACTIVE=true` (continue).
**Stop Types.** STOP has two meanings: **STOP-A** (sealed evidence on current surface: read-only/summary/committed and not editable) and **STOP-B** (context boundary reached: surface switched/closed/exhausted). Both must use `reason.category="other"` and include `"Next step should FINISH"`.
**Decision Logic (strict order).**

1. **Surface Change / Boundary** [STOP-B]: if `STATE_DELTA.new_surface=true` or `dismissed_surface=true`, or a modal/dialog/picker appears, or navigation to a different page is implied, then STOP-B.
2. **Blockers** [force CONTINUE]: if a permission dialog / blocking popup / error-toast is indicated by `STATE_DELTA` or UI evidence, then CONTINUE with `reason.category="blocked_by_surface"`.
3. **Sealed by Value + Read-only** [STOP-A]: if all `REQUIRED_VALUES` are present, `EDITING_ACTIVE=false`, `READ_ONLY_SURFACE_HINT=true`, and no blocker/error, then STOP-A.
4. **Commit closes context** [STOP-B]: define `COMMIT_ACTION` as last action implying Save/-Done/OK/Apply/Confirm (case-insensitive). If `COMMIT_ACTION=true` and no blocker/error and any of: (`new_surface` / `dismissed_surface` / `screen_changed` / `READ_ONLY_SURFACE_HINT`) holds, then STOP-B. If commit happened but the UI still looks editable, CONTINUE.
5. **Partial / Not Sealed** [CONTINUE]: if some required values are missing, or all are present but `EDITING_ACTIVE=true`, or read-only is not clearly true, then CONTINUE with `reason.category="info_missing"`.
6. **No-effect** [CONTINUE]: if the surface did not change and `STATE_DELTA` indicates no meaningful text/value/selection changes and no blocker, then CONTINUE with `reason.category="no_effect"`.
7. **Bounds-repeat Guard** [hard]: define `REPEAT_REGION` if the last two interaction actions target the same region (bounds overlap $\geq 0.85$), and `DELTA_EMPTY` if `STATE_DELTA` shows no surface change and no changes. If `REPEAT_REGION && DELTA_EMPTY`:
   - if `EDITING_ACTIVE=false` *or* `READ_ONLY_SURFACE_HINT=true`, then STOP-B.
   - else (still editable), CONTINUE but set `no_repeat=true` to force a different strategy next round.
8. **Default** [CONTINUE]: otherwise CONTINUE (insufficient evidence).

**Delta Summary Filling.** Populate `delta_summary` from `STATE_DELTA` when available; if missing, set booleans to `null` and lists to `[]`.
**Final Output Format (strict JSON).** Output exactly one JSON object:

```
{
  "success": "FAILURE" | "UNCERTAIN",
  "progress": "PROGRESSED" | "NO_CHANGE" | "REGRESSED" | "UNKNOWN",
  "finish_ok": false,
  "no_repeat": true | false,
  "delta_summary": {
    "screen_changed": true | false | null,
    "new_surface": true | false | null,
```

```
    "dismissed_surface": true | false | null,
    "text_changes": [string, ...],
    "value_changes": [string, ...],
    "selection_changes": [string, ...],
    "errors_or_blocks": [string, ...]
  },
  "reason": {
    "category": "info_missing" | "no_effect" | "wrong_target" |
                "wrong_order" | "intent_mismatch" |
                "ambiguous_expectations" | "blocked_by_surface" |
                "other",
    "evidence": [string, ...]
  }
}
```

No markdown. No extra keys. JSON only.

## C Details of Datasets

Table 4: Per-application task success rates (%) on DroidTask and AndroidLab.

| **DroidTask** | | |
|---|---|---|
| App | CORE (GPT-4o + Gemma 2-9B) | SenseAct (GPT-4o LLM-Only) |
| gallery | 90.0 | 88.9 |
| contacts | 84.0 | 50.0 |
| notes | 58.5 | 57.1 |
| calendar | 71.0 | 76.5 |
| app launcher | 79.8 | 100.0 |
| file manager | 42.5 | 68.8 |
| camera | 68.0 | 46.7 |
| voice recorder | 68.0 | 72.7 |
| clock | 82.0 | 84.6 |
| dialer | 74.0 | 40.0 |
| SMS messenger | 68.0 | 46.7 |
| music player | 68.0 | 87.5 |
| Average | 69.23 | 68.53 |
| **AndroidLab** | | |
| App | CORE (GPT-4o + Qwen 2.5-7B) | SenseAct (GPT-4o LLM-Only) |
| bluecoins | 30.0 | 40.0 |
| calendar | 37.4 | 21.4 |
| cantook | 17.5 | 50.0 |
| clock | 39.5 | 55.5 |
| contacts | 38.4 | 53.3 |
| map | 60.5 | 40.0 |
| pimusic | 17.5 | 33.3 |
| setting | 64.6 | 69.6 |
| zoom | 80.0 | 80.0 |
| Average | 41.84 | 49.28 |

**Per-Application Analysis.** Table 4 reports per-application task success rates on DroidTask and AndroidLab. On DroidTask, SenseAct does not uniformly outperform CORE across all applications. Instead, it achieves higher success rates on applications such as *App Launcher*, *File Manager*, *Music Player*, and *Calendar*, while underperforming on others including *Contacts*, *Dialer*, and *Camera*. As a result, the overall average success rate of SenseAct on DroidTask remains comparable to CORE, reflecting a trade-off between gains and losses across different application types.

On AndroidLab, SenseAct improves task success on the majority of applications, including *Cantook*, *Clock*, *Contacts*, *PiMusic*, and *Settings*, leading to a higher overall average success rate. Performance remains comparable on visually intensive applications such as *Zoom*, indicating that reducing visual perception does not degrade performance in these cases. These per-application results explain the aggregate trends observed in Table 4.1.1 and highlight that SenseAct's improvements are driven by consistent gains across a broad subset of AndroidLab applications rather than isolated outliers.