CONGUIBENCH: CONTROLLED DIAGNOSTICS FOR MULTI-CONSTRAINT GUI AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Reliable multi-constraint execution requires graphical user interface (GUI) agents to distinguish goal wording, requested procedure, and realized task state, distinctions that terminal success alone cannot reveal. We introduce **ConGUIBench**, comprising 118 two- to four-constraint shopping and travel tasks across four web platforms. Controlled Non-Procedural Goal (NPG) and Explicit Procedure (EP) variants, together with state interventions, evaluate goal invariance, procedural controllability, and state adaptivity. Across nine open-weight models on 118 tasks each, five sampled executions per instruction show that reordering equivalent NPG requirements produces 2.8–14.0% of excess cross-order success disagreement beyond same-instruction variability. Under EP, models match the requested order in 79.6–96.0%, yet only 17.6–63.6% of executions both follow the order and succeed. Cross-condition analysis shows that EP sharpens instruction-specific route alignment and increases ISR for eight of nine models, but the completion gains are modest. Agents also respond to visible progress, but suppressing one otherwise valid action lowers paired completion for every model; Oracle state information reduces this loss for most models. An auditor-maintained constraint checklist raises model-equal ISR by 3.7 points under EP and 9.2 points under NPG while also reducing across-run variability on average, highlighting constraint-state tracking as an important reasoning bottleneck. These results expose unstable goal execution, conditional procedural control, and brittle recovery, motivating their separate evaluation in GUI agents.

1 INTRODUCTION

Graphical user interface (GUI) agents are increasingly expected to fulfill shopping and travel requests that combine several user requirements, such as a product category, a brand, and a price range (Zhang & Zhang, 2024; Ma et al., 2024b; Hong et al., 2024; Zhang et al., 2024; Lyu et al., 2025). Reliable execution requires interpreting both what the user wants and how the current interface relates to that goal: mention order need not prescribe action order, an explicit sequence does, and completed or ineffective actions change what remains to be done. We therefore ask: *how reliably do GUI agents respond to goal wording, requested procedure, and realized task state?*

We formalize three complementary requirements. **Goal invariance** requires stable success across equivalent constraint mention orders. **Procedural controllability** requires execution to respect an explicitly requested feasible order. **State adaptivity** requires subsequent actions to reflect visible progress and action outcomes. These properties can diverge: following a requested route does not ensure stable goal completion or recovery from an action that fails to take effect.

Existing evaluations measure action prediction, task completion, and intermediate progress (Deng et al., 2023; Zhou et al., 2024; Koh et al., 2024; Rawles et al., 2025; Pan et al., 2024; Ma et al., 2024a; Yao et al., 2022). However a terminal success score does not by itself establish whether performance is stable across equivalent instructions, whether execution responds to a requested procedure, or whether subsequent actions account for actual progress. These questions require controlled instruction and state variations together with separate outcome and behavior measurements.

To address these questions, we introduce **ConGUIBench** as a controlled experimental setting for this study. It comprises 118 two- to four-constraint shopping and travel tasks across four web platforms (Figure 1). The benchmark combines matched *Non-Procedural Goal* (NPG) and *Explicit*

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

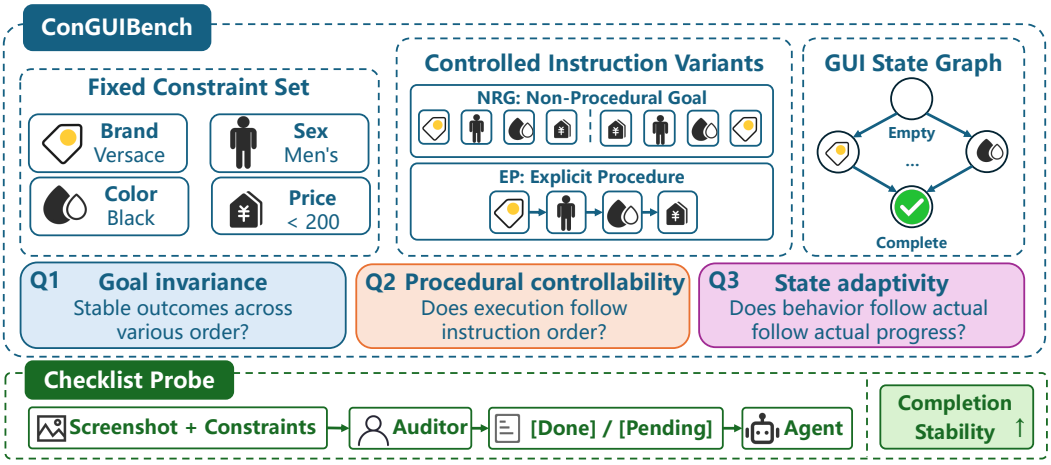

Figure 1: Overview of ConGUIBench and its diagnostics. A fixed constraint set is expressed as matched NPG and EP variants and evaluated over validated GUI state transitions. NPG permutations test goal invariance, EP instructions test procedural controllability, and controlled state and action-outcome variations test state adaptivity. Separately, an auditor-maintained checklist probes whether explicit constraint-state information improves completion and execution stability.

Procedure (EP) instructions, controlled state interventions, and validated GUI transition graphs to measure completion, procedural adherence, and adaptation to realized progress at the constraint level. Separately, we probe constraint-state tracking by supplying an auditor-maintained checklist of satisfied and pending constraints and measuring whether this additional information improves task completion and execution stability under both instruction formats.

Across nine open-weight models, equivalent NPG reorderings produce 2.8–14.0% of excess success disagreement. Under EP, requested-order adherence reaches 79.6–96.0%, but only 17.6–63.6% of executions both follow the order and succeed. Direct NPG-EP comparison confirms sharper instruction-specific route alignment under EP, accompanied by only modest gains. Agents also use visible progress, yet suppressing one valid action lowers paired completion; Oracle state information reduces this loss for most models. An explicit checklist improves model-equal ISR by 3.7 points under EP and 9.2 points under NPG, while reducing the across-replicate ISR standard deviation by 1.1 and 2.0 points, respectively. Together, these findings expose unstable goal execution, conditional procedural control, and brittle recovery, while identifying the maintenance and use of constraint state as an important target for stronger agent reasoning.

Our contributions are:

- We formulate goal invariance, procedural controllability, and state adaptivity as complementary requirements for reliable multi-constraint GUI execution, distinguishing outcome stability from responsiveness to procedure and progress.
- We develop ConGUIBench as a controlled experimental setting that combines instruction-order variants, validated constraint states, and action-outcome interventions with measurements of completion and execution behavior.
- We document cross-order outcome instability, conditional adherence to explicit procedures, and vulnerability to failed execution despite usable visible progress, and show that externalizing constraint state improves completion and sampling stability, motivating stronger state-tracking reasoning in GUI agents.

2 Related Work

Task completion and fine-grained progress. Mind2Web (Deng et al., 2023) studies web agents from demonstrations, while WebArena and VisualWebArena (Zhou et al., 2024; Koh et al., 2024) evaluate end-to-end web interaction. OSWorld (Xie et al., 2024) and AndroidWorld (Rawles et al.,

Table 1: Comparison of representative GUI-agent and grounding benchmarks by environment and evaluation focus. ConGUIBench focuses on constraint-level goal satisfaction and robustness to controlled instruction-order variations.

Benchmark	Environment	Evaluation focus
Mind2Web (Deng et al., 2023)	Web	Demonstration generalization
WebArena (Zhou et al., 2024)	Web	End-to-end completion
OSWorld (Xie et al., 2024)	Desktop/web	Open-ended completion
WebCanvas (Pan et al., 2024)	Web	Intermediate-state progress
DeepShop (Lyu et al., 2025)	Web shopping	Aspect and holistic completion
ScreenSpot-Pro (Li et al., 2025)	GUI screenshots	Element grounding
ConGUIBench	Web	**Constraint states and order robustness**

2025) extend task evaluation to desktop and mobile environments. Beyond terminal success, WebCanvas (Pan et al., 2024) annotates intermediate web states, AgentBoard (Ma et al., 2024a) evaluates milestone progress, and DeepShop (Lyu et al., 2025) reports aspect-level and holistic outcomes for multi-requirement shopping tasks. ConGUIBench builds on this fine-grained perspective but controls equivalent constraint orderings and validates alternative execution paths, allowing progress to be examined across controlled instruction and state variations.

GUI grounding and constraint-state tracking. SeeClick (Cheng et al., 2024) studies cross-platform instruction-to-element localization, and ScreenSpot-Pro (Li et al., 2025) evaluates grounding in high-resolution professional interfaces. Other work broadens grounding and action-prediction evaluation across desktop interfaces and implicit instructions (Nayak et al., 2025; Xie et al., 2026; Liu et al., 2025). Such local predictions are necessary for interaction, but they do not establish whether an attempted action changed the interface or whether previously satisfied constraints remain accounted for over a trajectory. Our evaluation therefore separates target attempts from verified completion and intervenes on visible progress and action outcomes. The checklist experiment further tests whether supplying explicit satisfied/pending state information improves completion and execution stability.

Constraint following and order sensitivity. IFEval and FollowBench (Zhou et al., 2023; Jiang et al., 2024) evaluate adherence to verifiable constraints in text generation, establishing that aggregate response quality can obscure failures on individual requirements. Interactive GUI execution adds a distinction between incidental mention order and an explicitly requested procedure, as well as dependence on the state produced by earlier actions. ConGUIBench holds the constraint set fixed while varying these two instruction formats and permits any annotated valid route when no procedure is required. This supports separate evaluation of goal invariance, procedural controllability, and state adaptivity. Table 1 summarizes its positioning relative to representative task-completion, progress, and grounding benchmarks.

3 Problem Formulation

Goals, observations, and execution. A task t specifies a set of independently verifiable constraints $C_t = \{c_{t,1}, \ldots, c_{t,m_t}\}$ and an initial GUI state s_0. Each constraint is a predicate $c_{t,k}(s) \in \{0, 1\}$ indicating whether a user requirement holds in state s. Define the satisfied and pending sets as

Mention order and procedural order. Let π denote a permutation of the task constraints. A *Non-Procedural Goal* instruction $I^{\mathrm{NPG}}_{t,\pi}$ lists the requirements in order π without imposing a temporal relation between their execution. We consider only vetted variants with the same goal semantics. An *Explicit Procedure* instruction $I^{\mathrm{EP}}_{t,\pi}$ additionally specifies the intended execution order through procedural language. More generally, a procedure can be represented by a precedence relation $\prec_I$ over constraints; NPG imposes no such relation, while the EP conditions studied here specify a total order. Interface dependencies independently restrict which actions are feasible.

Our task semantics concern achieving and maintaining required states. A constraint already satisfied at the start of an episode need not be re-executed; procedural requirements apply to the remaining

constraints. For example, an A–B–C procedure with B initially satisfied requires A followed by C. Tasks requiring a particular historical event, such as toggling a setting off and on even when it is already on, fall outside this formulation. We consider procedures that admit a feasible completion from the evaluated initial state.

Goal invariance. For a fixed agent, task, initial state, and execution budget, let

$$p_t(\pi) = \Pr_{\mu}\big[D_t(s_T) = C_t \mid I^{\mathrm{NPG}}_{t,\pi}, s_0\big]\,. \tag{1}$$

Goal invariance requires $p_t(\pi)$ to remain stable across equivalent mention orders. It permits different valid routes and different terminal pages satisfying the same requirements. Stability must be assessed alongside success: uniformly failing executions do not constitute reliable goal pursuit. Our experiments compare disagreement across evaluated mention orders with a baseline estimated from independent repetitions of the same instruction, for both terminal success and constraint satisfaction. These contrasts assess order-associated outcome variation beyond repeat-run variability under a fixed sampling configuration.

Procedural controllability. Procedural controllability requires the agent to respect $\prec_I$ and to adjust execution when that relation changes. For a trajectory that completes the task, let $\tau(c)$ be the step at which an initially pending constraint is completed. Under the monotone progress supported by our annotated transitions, adherence requires

$$c \prec_I c' \implies \tau(c) < \tau(c'), \qquad c, c' \in R_t(s_0). \tag{2}$$

An unfinished trajectory may follow a valid procedural prefix without achieving the goal. Accordingly, completion and adherence are distinct outcomes. Our procedural experiments operationalize adherence through identifiable first distinct target attempts. This measures responsiveness at the target-selection level; it does not establish that each target was completed before the next was attempted, or that every intervening action respected the procedure.

State adaptivity. State adaptivity requires subsequent behavior to reflect actual remaining requirements. For direct constraint-completion actions, the appropriate next target belongs to the feasible pending set, additionally respecting any procedural predecessors:

$$\mathcal{E}_t(s, I) = \{c \in R_t(s) : c \text{ is feasible in } s,\ \nexists c' \in R_t(s) \text{ with } c' \prec_I c\}. \tag{3}$$

Inspection and navigation may support later progress without directly completing a constraint. If an action has no effect, its target remains pending; retrying it can therefore be appropriate. Repeating an already satisfied requirement without need, skipping an unsatisfied requirement, and declaring success before all requirements hold represent different adaptation failures. We probe this requirement through partially completed initial states and a forced no-effect action. Pending-target selection and local recovery are behavioral diagnostics, while terminal completion measures whether execution ultimately succeeds. These probes examine distinct aspects of adaptation; the present experiments do not exhaustively test its interaction with explicit procedures.

4 Experiments

We test three properties of multi-constraint GUI execution: invariance to non-procedural mention order, control by explicit procedural order, and adaptation to visible progress and action outcomes.

4.1 Experimental Setup

4.2 Benchmark and Evaluation Protocol

Benchmark. ConGUIBench comprises 118 tasks across four platforms and two languages: 68 English tasks from Amazon and Booking, and 50 Chinese tasks from JD and XieCheng. Enumerating all constraint orders yields 688 NPG instructions and 688 matched EP instructions. Each task is paired with a graph of validated GUI states and transitions for its annotated completion orders. Construction, annotation, and platform statistics appear in Appendix B.

Strict transition protocol. Evaluation uses the finite annotated state graph. A click is accepted only within the candidate region of an annotated outgoing transition; a click outside all such regions terminates the episode, leaving unsatisfied constraints with zero credit. Agents may follow any annotated valid completion order. Completion scores therefore jointly reflect multi-constraint execution and action-level reliability within validated paths. A constraint is completed when the resulting state satisfies it.

Outcome metrics. *Constraint satisfaction rate (CSR)* averages the fraction of constraints satisfied at termination, preserving partial credit. *Instruction success rate (ISR)* measures whether all constraints in an instruction are satisfied. Formal definitions of CSR and ISR are provided in Appendix B.3. Order and state-adaptation diagnostics are defined with their respective experiments; procedural adherence is evaluated separately from goal completion.

4.2.1 Models and Inference

Evaluated models. We evaluate nine open-weight models, including Qwen3-VL at 2B, 4B, 8B, and 32B scales; GLM-4.6V-Flash; UI-Mate-9B and UI-Mate-27B; UI-Venus-1.5-2B and UI-Venus-1.5-8B (Bai et al., 2025; Hong et al., 2025; Ding et al., 2026; Team et al., 2026). This set spans both general-purpose multimodal models and GUI-specialized agents.

Inference protocol. For each model, we follow its official model card or release documentation for inference configuration, including prompt formatting, image preprocessing, and action serialization. All agents receive the same task instruction and agent-visible GUI state. Evaluation follows the state-graph protocol and completion metrics in Section 4.2, with experiment-specific interventions described below. Open-weight models are evaluated on NVIDIA A100 80GB GPUs, while proprietary models are accessed through their official APIs from the same evaluation pipeline.

4.2.2 Instruction Conditions and Statistical Conventions

Instruction conditions. We use two instruction conditions. *Non-Procedural Goal* (NPG) states the constraints without requiring an execution order; *Explicit Procedure* (EP) specifies an order with cues such as "first," "second," and "finally". All Q1–Q3 diagnostics use the same nine model families. Reported differences are percentage-point changes.

4.3 Instruction Order Shapes Route Selection

Q1 tests whether incidental mention order changes outcomes. Q2 tests whether explicit order controls execution. The cross-analysis directly compares the two conditions.

4.3.1 Q1: Does Goal Completion Remain Invariant to Mention Order?

Q1 permutes constraint mention order in equivalent NPG instructions. Each variant receives five sampled runs and one greedy run. We compare cross-order with same-instruction disagreement in terminal success (Δ_{order}) and constraint states (Δ_{state}); Appendix D gives the estimators and task-bootstrap procedure.

High completion does not eliminate mention-order sensitivity. Figure 2 shows positive excess disagreement: $\Delta_{\text{order}} = 2.8$–$14.0\%$, with every marginal 95% interval above zero, and $\Delta_{\text{state}} = 3.3$–$12.7\%$. This sensitivity persists at the highest observed performance level: UI-Venus-1.5-8B has the highest ISR and CSR point estimates in Table 2 (56.7% and 68.9%), yet its Δ_{order} and Δ_{state} remain positive at 4.9% [2.9, 8.1] and 4.1% [3.2, 6.3]. Thus, stronger completion does not eliminate changes in success or satisfied constraints caused by reordering the same requirements.

4.3.2 Q2: How Reliably Do Agents Follow Explicit Execution Orders?

Q2 evaluates feasible EP orders with five sampled runs per variant. COCR measures requested-order adherence among runs with identifiable full target-attempt sequences, ISR measures terminal completion irrespective of order, and SOCR counts the subset of successful executions whose

Model	ISR	CSR
Qwen3-VL-2B	21.3	33.9
Qwen3-VL-4B	38.2	51.4
Qwen3-VL-8B	40.1	53.5
Qwen3-VL-32B	43.6	52.7
GLM-4.6V-Flash	50.6	64.6
UI-Mate-9B	41.5	52.9
UI-Mate-27B	45.9	52.0
UI-Venus-1.5-2B	37.0	51.1
UI-Venus-1.5-8B	56.7	68.9

Table 2: Q1 sampled ISR and CSR (%). Five runs are averaged per instruction; tasks are weighted equally.

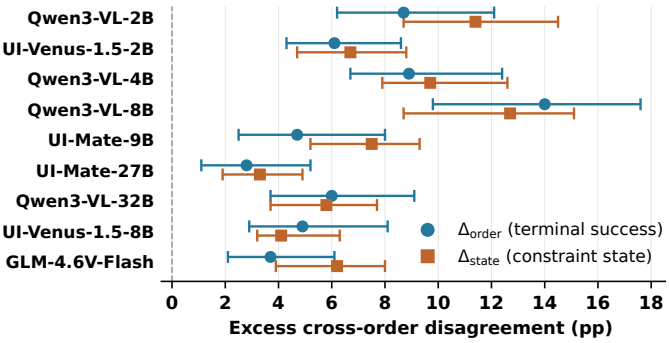

Figure 2: Q1 excess disagreement in terminal success (blue) and constraint states (orange), with marginal 95% task-bootstrap intervals. Values are cross-minus-within disagreement (%), not success-rate drops.

Table 3: Q2 sampled execution on 118 tasks per model. Entries are percentages with marginal 95% task-bootstrap intervals. COCR is conditional on full-sequence identifiability; SOCR counts the subset of ISR-successful executions that also match the requested order.

Model	COCR	ISR	SOCR
Qwen3-VL-2B	79.6 [70.9, 87.7]	20.4 [14.9, 26.4]	17.6 [12.7, 23.4]
Qwen3-VL-4B	88.7 [84.4, 92.8]	46.8 [38.8, 54.7]	42.1 [34.8, 49.6]
Qwen3-VL-8B	89.5 [85.4, 93.4]	49.4 [41.8, 57.4]	44.6 [37.1, 52.2]
Qwen3-VL-32B	95.2 [92.2, 97.8]	49.3 [41.3, 58.0]	47.0 [39.0, 55.2]
GLM-4.6V-Flash	90.9 [86.0, 95.2]	57.4 [49.6, 65.4]	54.4 [46.7, 62.0]
UI-Mate-9B	95.0 [91.9, 97.6]	44.6 [37.5, 52.1]	42.2 [35.5, 49.2]
UI-Mate-27B	96.0 [93.1, 98.4]	48.2 [40.4, 56.2]	46.3 [38.6, 54.3]
UI-Venus-1.5-2B	88.5 [83.1, 93.2]	41.1 [34.3, 48.9]	38.4 [31.6, 45.7]
UI-Venus-1.5-8B	93.1 [89.8, 95.9]	66.8 [59.8, 73.2]	63.6 [56.6, 70.3]

target-attempt sequence also matches the requested order. Appendix E gives formal definitions and scenario-level results.

High order matching is conditional on realizing an identifiable sequence. Table 3 shows COCR values of 79.6%–96.0%. ISR is 20.4%–66.8%, while the stricter SOCR is 17.6%–63.6%. This indicates that most successful executions also produce an identifiable sequence matching the requested order. Taken together, the results suggest that explicit procedures effectively guide target ordering when a full route is realized, whereas producing a complete successful execution remains the larger limitation.

4.3.3 Cross-Analysis: Is Order Sensitivity Selective and Useful?

We compare NPG and EP on shared tasks. OwnMFR and CrossMFR measure matches to the current and competing instruction orders; WVOA and XVOA measure route agreement within and across variants. Completion utility is the paired task-level change $\Delta\mathrm{ISR} = \mathrm{ISR}_{\mathrm{EP}} - \mathrm{ISR}_{\mathrm{NPG}}$. Appendix F provides the platform-level breakdowns.

NPG routes remain shaped by instruction order, but less strongly than under EP. Table 4 shows that NPG execution is still shaped by instruction order: OwnMFR is 41.8%–66.5%, consistently above CrossMFR at 13.5%–25.5%. NPG has lower OwnMFR and WVOA but higher CrossMFR and XVOA. Explicit procedures therefore sharpen instruction-specific route alignment, while non-procedural mention order continues to exert a weaker influence on execution.

Stronger route control is accompanied by partial but uneven completion gains. Table 4 shows higher ISR under EP for eight of nine models, with gains of up to 9.3 points and intervals above zero

Table 4: Cross-analysis of NPG and EP execution. OwnMFR measures whether an identifiable execution route matches the current instruction order, whereas CrossMFR measures a match to a competing variant's order. WVOA and XVOA measure route agreement between runs of the same and different instruction variants, respectively. Route cells report the NPG rate followed by the NPG−EP difference in parentheses (red: negative; green: positive); the final column reports $\Delta\text{ISR} = \text{ISR}_{\text{EP}} - \text{ISR}_{\text{NPG}}$. Rates are percentages and differences are percentage points.

Model	OwnMFR	CrossMFR	WVOA	XVOA	ΔISR [95% CI]
Qwen3-VL-2B	41.8 (-37.8)	25.5 (+21.4)	96.3 (-3.0)	47.0 (+34.8)	−1.0 [−4.7, 3.2]
Qwen3-VL-4B	50.0 (-38.7)	18.4 (+16.0)	86.2 (-11.1)	35.9 (+27.8)	+8.6 [+3.9, +13.6]
Qwen3-VL-8B	61.6 (-27.9)	14.6 (+12.3)	96.5 (-3.4)	36.0 (+29.4)	+9.3 [+4.4, +14.7]
Qwen3-VL-32B	55.1 (-40.1)	19.5 (+18.6)	87.6 (-12.2)	41.7 (+39.4)	+5.7 [+2.1, +9.9]
GLM-4.6V-Flash	59.8 (-31.1)	14.6 (+12.4)	82.9 (-15.0)	37.1 (+31.5)	+6.7 [+2.4, +11.5]
UI-Mate-9B	66.5 (-28.5)	13.5 (+12.6)	80.1 (-17.9)	27.2 (+24.6)	+3.1 [−0.1, +6.7]
UI-Mate-27B	62.7 (-33.3)	17.6 (+16.9)	82.3 (-16.8)	29.1 (+27.5)	+2.3 [−0.0, +4.9]
UI-Venus-1.5-2B	58.1 (-30.4)	17.7 (+15.9)	81.1 (-15.3)	33.5 (+24.3)	+4.2 [+0.7, +8.0]
UI-Venus-1.5-8B	61.5 (-31.6)	17.1 (+15.8)	79.5 (-18.7)	36.0 (+29.8)	+8.9 [+3.7, +14.3]

for six models. The improvement is nevertheless modest and not universal. These results suggest that explicit route control can improve completion, but its current benefit remains heterogeneous.

4.3.4 Q3: Do Agents Adapt to the Actual Task State?

Q3 measures adaptation to initial progress and unrealized actions. The initial-state diagnostic pairs matched *Empty* states, with no satisfied constraints, and *Partial* states, with a satisfied subset. A first identifiable action is *valid-next* if it targets a pending constraint; SAG is the paired Partial-minus-Empty change in this rate. The action-outcome diagnostic contrasts a normal baseline (A) with an intervention (B) that suppresses the first valid action and leaves its target pending. Among applicable B cases, *Local response* records whether the next action retries that target or selects another pending constraint, while *Paired B−A* measures the change in ISR.

An independent 2×2 follow-up crosses state information (*Static*/*Dynamic*) with action outcome (*Normal*/*Blocked*). Static repeats only the task instruction; Dynamic additionally supplies Oracle *satisfied*/*pending* labels for every constraint as a diagnostic. Normal executes valid clicks, whereas Blocked suppresses the first valid click, injects a no-effect observation, and grants one extra step. The Static–Dynamic contrast isolates the value of persistent constraint-state information beyond the immediate failure feedback shared by both Blocked conditions. Oracle mitigation is

$$\Delta_{\text{mit}} = (\text{ISR}_{\text{SN}} - \text{ISR}_{\text{SB}}) - (\text{ISR}_{\text{DN}} - \text{ISR}_{\text{DB}}), \tag{4}$$

where S/D denote Static/Dynamic and N/B denote Normal/Blocked. The two diagnostics use separate samples; Appendix G gives full details.

Visible-state sensitivity does not ensure recovery from an unrealized action. Table 5(a) shows positive SAG for all models (+6.6%–+18.9%), confirming that visible progress redirects target selection. Yet after one valid action is suppressed, locally consistent responses range from 45.9% to 94.4% while paired completion falls by 4.4%–56.1% for every model. Local adaptation at the next step therefore does not ensure task recovery when attempted actions and actual progress diverge.

Oracle state reduces but does not eliminate the recovery gap. Under Static prompting, blocking the first valid action reduces ISR by 3.5%–30.5% (Table 5(b)). Oracle state reduces this penalty for eight of nine models by 1.5%–19.2%, but the Dynamic drop remains 1.3%–19.1% for every model; Qwen3-VL-2B instead worsens by 1.3%. Explicit state information is therefore a partial bottleneck rather than a sufficient solution, leaving failures in recovery planning, grounding, or execution.

4.4 Error Profiles Across NPG and EP

To identify which failure modes limit multi-constraint execution and how they differ between NPG and EP, we classify evaluator-rejected clicks by comparing their predicted coordinates with semantic

Table 5: Q3 state adaptation. Rates are percentages; SAG, paired B−A, Static/Dynamic drops, and mitigation are percentage-point differences. Local response is conditional on applicable B cases, and negative paired B−A denotes lower completion under intervention B. Panel (b) uses a separate sample; Appendix G reports full results.

(a) State-adaptation diagnostic

Model	SAG	Local response	Paired B−A
Qwen3-VL-2B	+8.8	79.9	−12.2
Qwen3-VL-4B	+11.9	86.8	−6.6
Qwen3-VL-8B	+14.1	94.4	−4.8
Qwen3-VL-32B	+18.9	92.3	−4.4
GLM-4.6V-Flash	+9.3	45.9	−56.1
UI-Mate-9B	+7.8	64.9	−28.0
UI-Mate-27B	+10.3	78.0	−15.6
UI-Venus-1.5-2B	+6.6	82.4	−7.6
UI-Venus-1.5-8B	+13.4	79.0	−11.8

(b) Oracle-state follow-up

Model	Static drop	Dynamic drop	Mitigation
Qwen3-VL-2B	4.2	5.5	−1.3
Qwen3-VL-4B	7.1	5.5	+1.6
Qwen3-VL-8B	9.2	4.0	+5.2
Qwen3-VL-32B	3.5	1.3	+2.2
GLM-4.6V-Flash	30.5	11.3	+19.2
UI-Mate-9B	23.6	11.5	+12.1
UI-Mate-27B	16.9	7.7	+9.2
UI-Venus-1.5-2B	11.0	6.3	+4.7
UI-Venus-1.5-8B	20.6	19.1	+1.5

Table 6: Model-equal error profiles under EP and NPG. Rates are percentages and $\Delta = \mathrm{NPG} - \mathrm{EP}$ is in percentage points; the final column counts models with higher NPG rates. Panel (b) is conditional on *Other* and lists non-exhaustive subtypes, so its rates need not sum to 100%.

(a) Overall error types

Error type	EP	NPG	Δ	NPG>EP models
Grounding near miss	28.7	27.6	−1.1	3/9
Completed-target repetition	11.8	14.0	+2.3	7/9
Search bar/header	4.4	3.2	−1.2	3/9
Product/room region	6.8	8.4	+1.6	7/9
Other	48.3	46.7	−1.6	4/9

(b) Diagnostic subtypes within Other

Error subtype	EP	NPG	Δ	NPG>EP models
Requires scrolling	44.2	51.8	+7.6	9/9
No clickable gold target	6.8	2.8	−4.0	0/9
Wide near-miss	32.5	34.2	+1.7	6/9
Other-screen hit	5.4	1.3	−4.0	1/9

screen regions. Clicks near but outside pending targets are *grounding near misses*; those associated with already satisfied constraints are *completed-target repetitions*; and clicks in the search/header or product/room regions receive the corresponding labels. Remaining clicks are assigned to *Other* and further diagnosed as requiring scrolling, lacking a clickable gold target, qualifying as a near miss only under a relaxed threshold (*wide near-miss*), or hitting a target region from another screen. Table 6 compares EP and NPG error composition using model-equal averages over the nine models. Panel (b) provides selected diagnostics within *Other*, rather than an exhaustive second-level taxonomy. Appendix H reports the per-model distributions and sample sizes.

Grounding remains a persistent bottleneck. Grounding near misses are the largest named error type in Table 6(a), accounting for 28.7% of EP errors and 27.6% of NPG errors. Localization errors also remain prominent within *Other*: *wide near-misses*, identified only after relaxing the distance threshold, account for 32.5% under EP and 34.2% under NPG. Neither pattern is consistently reduced by EP, indicating that explicit procedures do little to improve coordinate-level grounding.

Explicit procedures reduce some execution-organization failures. Completed-target repetition accounts for 11.8% of EP errors and 14.0% of NPG errors, with higher NPG rates for seven of nine models. Within *Other*, targets requiring scrolling account for 44.2% under EP and 51.8% under NPG, with the same direction for all nine models. Premature clicks in product or room regions are also more frequent under NPG for seven models (8.4% versus 6.8%). These patterns suggest that EP helps agents maintain progress, continue off-screen search, and delay final selection.

Table 7: Matched checklist effects under EP and NPG. Accuracy gain is $\Delta\mathrm{ISR}$; stability gain G_σ is the reduction in across-replicate ISR standard deviation. All entries are percentage-point gains, so positive values indicate improvement. The final row averages models equally.

	Accuracy gain (ΔISR)		Stability gain (G_σ)	
Model	EP	NPG	EP	NPG
Qwen3-VL-2B	$+9.5$	$+14.9$	-1.4	$+2.9$
Qwen3-VL-4B	$+1.4$	$+10.6$	$+1.2$	$+4.4$
Qwen3-VL-8B	$+1.4$	$+10.0$	$+0.3$	$+1.3$
Qwen3-VL-32B	$+2.1$	$+7.9$	-1.0	$+1.5$
GLM-4.6V-Flash	-0.3	$+4.7$	$+2.6$	$+0.1$
UI-Mate-9B	$+7.9$	$+8.6$	$+1.6$	-4.1
UI-Mate-27B	-1.3	$+1.4$	-1.2	-0.2
UI-Venus-1.5-2B	$+6.0$	$+12.1$	$+4.1$	$+2.7$
UI-Venus-1.5-8B	$+6.2$	$+12.2$	$+3.7$	$+9.1$
Model-equal mean	$\mathbf{+3.7}$	$\mathbf{+9.2}$	$\mathbf{+1.1}$	$\mathbf{+2.0}$

4.5 Can Explicit Checklists Improve Multi-Constraint Execution?

Motivated by the progress-tracking failures above, we externalize constraint-state tracking through a checklist and test whether it improves execution accuracy and stability under EP and NPG. The checklist condition (CL) augments the agent context with *satisfied/pending* labels produced by a lightweight 4B auditor over the ground-truth constraint list; noCL omits this record. We report accuracy gain as $\Delta\mathrm{ISR} = \mathrm{ISR}_{\mathrm{CL}} - \mathrm{ISR}_{\mathrm{noCL}}$ and stability gain as $G_\sigma = \sigma_{\mathrm{noCL}} - \sigma_{\mathrm{CL}}$, where σ is the across-replicate ISR standard deviation. Positive values indicate improvement for both measures. Appendix I reports full results, and Appendix J provides the auxiliary prompt templates.

Checklists improve execution accuracy. Table 7 shows positive ISR changes for seven of nine models under EP and all nine under NPG. Model-equal ISR improves by 3.7 and 9.2 points, respectively. The larger and more consistent gains under NPG indicate that explicit constraint tracking is especially useful when the instruction does not prescribe a procedure.

Checklists generally improve execution stability. CL reduces the across-replicate ISR standard deviation for six of nine models under EP and seven of nine under NPG. The model-equal reduction is 1.1 points for EP and 2.0 points for NPG, indicating that the checklist generally makes outcomes less dependent on sampling variation. Together, the accuracy and stability gains point to the consistent maintenance and use of constraint state as an important reasoning target for GUI agents; the externally maintained checklist diagnoses this need rather than serving as a complete solution.

5 Conclusion

ConGUIBench separates three properties that terminal success conflates: stability across equivalent goal expressions, responsiveness to requested procedures, and adaptation to realized task state. Across nine evaluated agents, incidental mention order changes outcomes, while explicit procedures sharpen route alignment but yield only modest and uneven completion gains; high conditional order adherence therefore does not imply reliable end-to-end execution. Agents respond to visible progress yet recover poorly when an attempted action fails to take effect. Error profiles further show that procedural instructions reduce some progress-maintenance, search, and premature-selection failures without resolving coordinate-level grounding. An auditor-maintained checklist improves completion and sampling stability, but its reliance on an external auditor and ground-truth constraint decomposition motivates agents that maintain and update constraint state within their own reasoning. Reliable multi-constraint execution requires this capability alongside stronger recovery planning, grounding, and action execution; these properties should be evaluated separately rather than inferred from terminal success alone.

AI USE STATEMENT

In this work, we used generative AI tools to assist with constructing task instructions, implementing methods in code, designing and critiquing research methodology and experiments, translating text, interpreting experimental results, and proposing or refining hypotheses. We additionally used these tools to draft portions of the manuscript, suggest its structure and title, and search for and collect relevant information. We did not use generative AI for qualitative or thematic data analysis, or for cleaning or reformatting the dataset; the remaining required-disclosure categories were not applicable to this work. All AI-assisted task instructions were subject to the same manual review and quality-control process described in the paper. The authors also reviewed and, where necessary, revised AI-assisted code, interpretations, and manuscript text, and verified relevant information against its sources. We take full responsibility for the final content of this work, including all text, claims, code, and artifacts produced with the assistance of generative AI.

ETHICS STATEMENT

This work does not involve human subjects, personal data, or consequential real-world transactions such as purchases or reservations. ConGUIBench is intended solely for research on the reliability of GUI agents. Its coverage is limited to four shopping and travel platforms in two languages and should not be interpreted as representative of all users, domains, or interfaces.

REPRODUCIBILITY STATEMENT

We document the benchmark composition, task construction, annotation process, and quality-control procedure in Appendix B, with the state-transition annotation details in Appendix B.2. Section 4.1 describes the shared model and inference setup, while Appendix B.3 specifies the transition protocol, outcome metrics, and aggregation rules. The experiment-specific appendices report eligibility conditions, sampling configurations, estimators, interventions, uncertainty procedures, and supporting numerical results for Q1–Q3 and the checklist analysis. The complete constraint-parser and checklist-auditor prompt templates are reproduced in Appendix J. Together, these descriptions provide the information needed to reconstruct the evaluation protocol and independently verify the reported analyses.

REFERENCES

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

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

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

Zihan Ding, Longxu Dou, Qi Gao, Xiangwu Guo, Shengchao Hu, Zilong Huang, Zihang Jiang, Lei Ke, Mengcheng Lan, Weixian Lei, et al. Ui-mate: Advancing open-weight foundation gui agents with in-context demonstrations. *arXiv preprint arXiv:2608.15930*, 2026.

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

Wenyi Hong, Wenmeng Yu, Xiaotao Gu, Guo Wang, Guobing Gan, Haomiao Tang, Jiale Cheng, Ji Qi, Junhui Ji, Lihang Pan, et al. Glm-4.5 v and glm-4.1 v-thinking: Towards versatile multimodal reasoning with scalable reinforcement learning. *arXiv preprint arXiv:2507.01006*, 2025.

Yuxin Jiang, Yufei Wang, Xingshan Zeng, Wanjun Zhong, Liangyou Li, Fei Mi, Lifeng Shang, Xin Jiang, Qun Liu, and Wei Wang. Followbench: A multi-level fine-grained constraints following benchmark for large language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 4667–4688, 2024.

Jing Yu Koh, Robert Lo, Lawrence Jang, Vikram Duvvur, Ming Lim, Po-Yu Huang, Graham Neubig, Shuyan Zhou, Russ Salakhutdinov, and Daniel Fried. Visualwebarena: Evaluating multimodal agents on realistic visual web tasks. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 881–905, 2024.

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. In *Proceedings of the 33rd ACM International Conference on Multimedia*, pp. 8778–8786, 2025.

Xinyi Liu, Xiaoyi Zhang, Ziyun Zhang, and Yan Lu. Ui-e2i-synth: Advancing gui grounding with large-scale instruction synthesis. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 15668–15684, 2025.

Yougang Lyu, Xiaoyu Zhang, Lingyong Yan, Maarten de Rijke, Zhaochun Ren, and Xiuying Chen. Deepshop: A benchmark for deep research shopping agents. *arXiv preprint arXiv:2506.02839*, 2025.

Chang Ma, Junlei Zhang, Zhihao Zhu, Cheng Yang, Yujiu Yang, Yaohui Jin, Zhenzhong Lan, Lingpeng Kong, and Junxian He. Agentboard: An analytical evaluation board of multi-turn llm agents. *Advances in neural information processing systems*, 37:74325–74362, 2024a.

Xinbei Ma, Zhuosheng Zhang, and Hai Zhao. Coco-agent: A comprehensive cognitive mllm agent for smartphone gui automation. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 9097–9110, 2024b.

Shravan Nayak, Xiangru Jian, Kevin Qinghong Lin, Juan A Rodriguez, Montek Kalsi, Rabiul Awal, Nicolas Chapados, M Tamer Özsu, Aishwarya Agrawal, David Vazquez, et al. Ui-vision: A desktop-centric gui benchmark for visual perception and interaction. *arXiv preprint arXiv:2503.15661*, 2025.

Yichen Pan, Dehan Kong, Sida Zhou, Cheng Cui, Yifei Leng, Bing Jiang, Hangyu Liu, Yanyi Shang, Shuyan Zhou, Tongshuang Wu, et al. Webcanvas: Benchmarking web agents in online environments. *arXiv preprint arXiv:2406.12373*, 2024.

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

Venus Team, Changlong Gao, Zhangxuan Gu, Yulin Liu, Xinyu Qiu, Shuheng Shen, Yue Wen, Tianyu Xia, Zhenyu Xu, Zhengwen Zeng, et al. Ui-venus-1.5 technical report. *arXiv preprint arXiv:2602.09082*, 2026.

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

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

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. Webshop: Towards scalable real-world web interaction with grounded language agents. *Advances in Neural Information Processing Systems*, 35:20744–20757, 2022.

Shaoqing Zhang, Zhuosheng Zhang, Kehai Chen, Xinbei Ma, Muyun Yang, Tiejun Zhao, and Min Zhang. Dynamic planning for llm-based graphical user interface automation. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 1304–1320, 2024.

Zhuosheng Zhang and Aston Zhang. You only look at screens: Multimodal chain-of-action agents. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 3132–3149, 2024.

Jeffrey Zhou, Tianjian Lu, Swaroop Mishra, Siddhartha Brahma, Sujoy Basu, Yi Luan, Denny Zhou, and Le Hou. Instruction-following evaluation for large language models. *arXiv preprint arXiv:2311.07911*, 2023.

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

Table 8: Composition of ConGUIBench. The instruction column counts order variants per condition: each NPG variant has a matched EP instruction. States and transitions form the execution graph.

			Instructions		Execution State Graph	
Platform	Domain	Tasks	Instructions	Avg. Constraints	States	Transitions
Amazon	Shopping	30	142	2.57	196	257
JD	Shopping	30	220	2.87	264	376
Booking	Travel	38	192	2.39	296	335
XieCheng	Travel	20	134	2.65	145	180
Total	—	**118**	**688**	**2.60**	**901**	**1,148**

A Limitations and Future Work

Limitations. Our findings concern two- to four-constraint tasks from four shopping and travel platforms; longer workflows, form filling, multi-tab interaction, and other domains remain untested. Evaluation is restricted to annotated feasible transitions, and off-graph actions terminate an episode, so completion characterizes execution within validated paths rather than open-web recovery. Although NPG variants are vetted for goal equivalence, reordering can require grammatical reformulation; moreover, NPG and EP differ in length and discourse form as well as procedural specification. The measured contrasts therefore characterize instruction-order and format interventions rather than isolated positional or procedural-marker effects.

Future work. Reliable GUI agents should be invariant to order when it does not change the goal, responsive when the user requests a procedure, and adaptive to the state their actions actually produce. On the modeling side, agents need to maintain constraint state within their own reasoning and combine it with recovery planning, reliable grounding, and action verification rather than treating any single component as sufficient.

B Dataset Construction and Evaluation Details

B.1 Design Goals and Overview

ConGUIBench evaluates how GUI agents execute multi-constraint requests: whether they satisfy a *set* of user requirements, rather than merely reach a plausible final page. Its design serves three goals: **partial-progress measurement**, **controlled linguistic-order analysis**, and **execution diagnosis**. First, it records the satisfaction of each requirement, so a failed instruction can be distinguished from partial progress. Second, it holds the task-level constraint set fixed while varying the order in which those requirements are stated, enabling a controlled analysis of order-associated differences without prescribing an execution order. Third, it records validated interface states and transitions, enabling diagnosis of the execution path that produced each constraint outcome.

Dataset statistics. Among the 118 tasks, 59 contain two constraints, 47 contain three, and 12 contain four, giving 307 constraint slots. Enumerating all constraint orders yields 688 variants per instruction condition. Table 8 gives the platform-level composition.

Formally, a task t is defined by a constraint set $C_t = \{c_{t,1}, \ldots, c_{t,m_t}\}$, where $m_t \in \{2, 3, 4\}$. We express the same set through all of its vetted linguistic permutations,

$$\mathcal{I}_t = \{\pi(C_t) \mid \pi \in S_{m_t}\}, \tag{5}$$

where S_{m_t} denotes the permutations of m_t elements. Each permutation is vetted for semantic equivalence and realized as both an NPG instruction and a matched EP instruction. NPG variants change constraint position without prescribing execution order, whereas EP variants explicitly request the corresponding order.

Instruction order and execution order. Instruction order refers to the order in which constraints are expressed. Candidate instructions are not retained through mechanical text rearrangement: each realization must preserve the intended constraint order while remaining semantically equivalent and

Table 9: Instruction-length statistics for EP and NPG. Entries with distributions report mean (median; range); the remaining entries report means. NPG/EP is the ratio of the corresponding means.

Measure	EP	NPG	NPG/EP
EN characters	305.0 (297.5; 169–478)	85.2 (83.0; 43–144)	0.28
ZH characters	85.8 (85.0; 58–124)	34.5 (34.0; 18–49)	0.40
EN words	51.6	14.2	0.28
ZH Han characters	75.6	29.0	0.38
All characters	192.2	59.1	0.31

Table 10: Constraint-type distribution in ConGUIBench. Percentages are over 307 constraint instances and sum to 100.1% because of rounding.

Constraint type	Instances	Share
Filter	279	90.9%
Sort	23	7.5%
Form value	3	1.0%
Direct selection	2	0.7%

natural under the platform's usage conventions. Unsuitable formulations are removed, and only jointly approved realizations enter the dataset. Execution order is defined separately by the state-transition graph, which records the feasible constraint-completion paths identified and validated during collection. NPG and EP use the same graph and may traverse any annotated path; CSR and ISR accept completion through any such path. Under EP, the requested order is not enforced by restricting transitions but is evaluated separately through COCR and SOCR. Q2 uses the full retained EP set and applies no additional feasibility-based exclusion at analysis time.

Instruction length. Table 9 reports descriptive length statistics for the analyzed instruction sets. NPG mean character counts are 28% and 40% of their EP counterparts in English and Chinese, respectively, and 31% across all instructions. The NPG–EP comparisons therefore combine explicit procedural specification with differences in length and discourse format; they are not length-controlled estimates of the effect of procedural markers alone.

ConGUIBench has three nested units of analysis. A **constraint** is an independently verifiable requirement, an **instruction** is one vetted linguistic ordering of a constraint set, and a **task** groups all instructions derived from the same set. This hierarchy supports the task-, instruction-, and constraint-level metrics defined in Section 4.2.

B.2 Data Sources and Annotation

We collect tasks from four real-world websites where users routinely issue multi-constraint requests: Amazon and JD for online shopping, and Booking and XieCheng for travel reservation. The 68 Amazon and Booking tasks use English, while the 50 JD and XieCheng tasks use Chinese. Figure 3 illustrates the annotation pipeline: each constraint set yields vetted linguistic orderings for order-sensitivity evaluation and a state-transition graph covering the execution orders identified during collection.

Task construction. Each task starts from a fixed initial page and combines two to four constraints. A constraint set must be jointly satisfiable through at least one validated path. The 307 constraint instances comprise 279 filters, 23 sorting operations, three form values, and two direct selections (Table 10). At the task level, 93 tasks contain only filters, 21 combine filters with sorting, two combine form values with sorting, one contains only direct selections, and one combines a filter with a form value. The resulting benchmark primarily evaluates multi-filter execution while retaining a smaller set of sorting, value-entry, and direct-selection operations.

Instruction quality control. Two annotators independently review every instruction realization. They verify that it contains the intended constraint set in the specified mention order, preserves the original goal semantics, and follows natural human usage. For EP, they additionally verify that

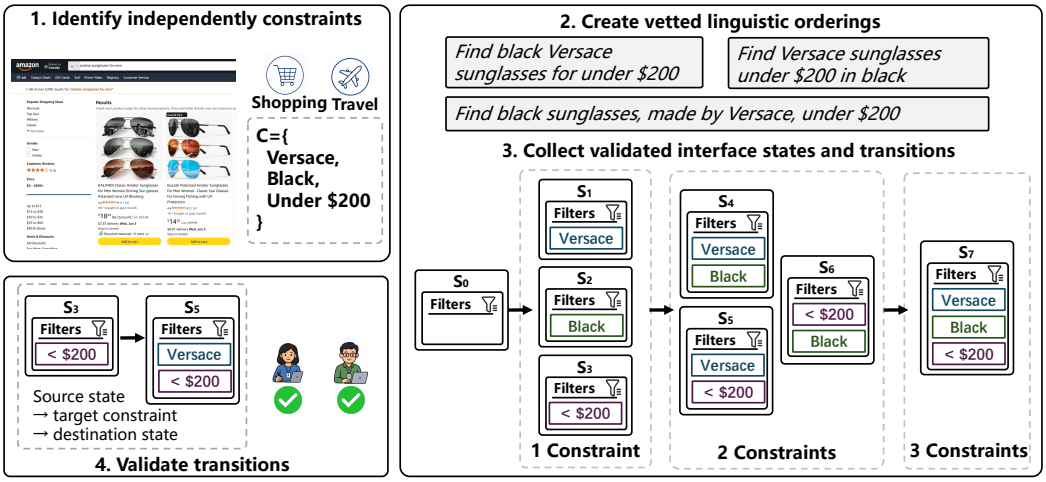

Figure 3: Data construction and annotation in ConGUIBench. Starting from a multi-constraint goal, we create vetted linguistic orderings that differ only in constraint position, and annotate valid GUI state transitions for feasible constraint-completion orders. The resulting state-transition graph supports constraint-level verification while allowing agents to execute constraints in any annotated valid order.

the requested execution order matches the intended constraint order; for NPG, they verify that the instruction does not introduce a procedural requirement. An instruction is retained only when both annotators approve it.

Constraint application is not necessarily commutative in a GUI: applying one filter can change visible controls, available options, or the intermediate page from which another constraint can be handled. We therefore collect the interface states and transitions induced by annotated feasible constraint-completion orders for each constraint set. These records form a state-transition graph whose nodes represent satisfied-constraint sets and whose edges represent validated completions of an additional constraint. At evaluation time, agents may satisfy constraints in any annotated valid order; their current satisfied-constraint set determines the relevant state and valid next transitions, independently of the order in which constraints were stated. The graph is a validated representation of annotated feasible paths, not a claim that every possible interaction trajectory has been exhaustively enumerated.

Transition quality control. Each transition is annotated with its task identifier, the satisfied-constraint sets before and after the transition, the agent-visible page states, and a candidate click region. Both annotators independently inspect the source state, assigned constraint, click region, and resulting state. A transition is retained only when both confirm that the source permits the intended constraint, the region corresponds to its interface control, and activating it produces the annotated constraint-state change without invalidating previously satisfied constraints. Disputed regions or transitions are corrected and independently reviewed again. We further verify that every nonterminal state has at least one outgoing transition, whereas states satisfying all constraints have none. This validation prevents invalid intermediate pages, silently reset filters, or unrelated page changes from being labeled as successful, and supplies the constraint-level supervision needed to distinguish partial completion from complete failure. As summarized in Table 8, ConGUIBench contains 118 tasks, 688 NPG instructions, and 688 matched EP instructions, supported by 901 validated interface states and 1,148 transitions.

B.3 Evaluation Protocol and Metrics

Strict transition protocol. Under the baseline protocol, ConGUIBench is evaluated on the finite graph of annotated UI states and validated transitions. At each state, a click is accepted only when its coordinate falls within the candidate region of an annotated outgoing transition. A click outside every such region has no benchmark-observed successor, so the episode terminates and all sub-

sequently unsatisfied constraints receive zero credit. CSR and ISR therefore measure completion within validated paths and jointly reflect multi-constraint execution and action-level reliability; they do not estimate recovery from unannotated actions in an open web environment. The controlled no-effect interventions in Q3 are the only exception: they deliberately suppress an evaluator-valid action and allow execution to continue, while ordinary invalid agent actions retain the strict termination rule.

Completion and target-attempt sequences. We distinguish successful constraint completion from attempted target selection. A *constraint completion sequence* records the chronological order of constraints newly satisfied by validated state transitions; no separate confirmation action is required. A *target-attempt sequence* instead records the first occurrence of each identifiable constraint target, whether or not the corresponding action changes the state. Q2 operationalizes procedural adherence using the latter sequence and therefore measures target-selection order rather than proving that each constraint was completed before the next was attempted. Terminal outcomes depend only on the satisfied-constraint set at the end of the episode.

Terminal outcome metrics. ConGUIBench first evaluates each execution at the constraint level. Let $y_{t,i,r,k} \in \{0,1\}$ indicate whether the k-th constraint is satisfied at termination for task t, evaluated instruction i, and repetition r. Each instruction contains m_t constraints, and the per-execution outcomes are

$$\mathrm{CS}(t,i,r) = \frac{1}{m_t}\sum_{k=1}^{m_t} y_{t,i,r,k}, \qquad \mathrm{IS}(t,i,r) = \prod_{k=1}^{m_t} y_{t,i,r,k}. \tag{6}$$

For a given analysis, let $\mathcal{J}_t$ be its eligible evaluated instructions for task t, $N_t = |\mathcal{J}_t|$, and R the number of sampled repetitions per instruction. The reported completion metrics average repetitions and instructions within each physical task before weighting tasks equally:

$$\mathrm{CSR} = \frac{1}{|\mathcal{T}|}\sum_{t\in\mathcal{T}} \frac{1}{N_t R}\sum_{i\in\mathcal{J}_t}\sum_{r=1}^{R} \mathrm{CS}(t,i,r), \tag{7}$$

$$\mathrm{ISR} = \frac{1}{|\mathcal{T}|}\sum_{t\in\mathcal{T}} \frac{1}{N_t R}\sum_{i\in\mathcal{J}_t}\sum_{r=1}^{R} \mathrm{IS}(t,i,r). \tag{8}$$

Here $\mathcal{T}$ is the set of physical tasks. *Constraint satisfaction rate (CSR)* preserves credit for partial completion, whereas *instruction success rate (ISR)* requires every constraint in an execution to be satisfied. Experiment-specific eligibility, pairing, conditional denominators, and deviations from the balanced repetition design are stated with the corresponding analysis.

Order and sequence diagnostics. Q1 compares cross-order disagreement with same-instruction repeat-run disagreement in terminal success and terminal constraint states. Q2 evaluates requested-order adherence from identifiable first-occurrence target-attempt sequences using COCR and SOCR, separately from order-agnostic ISR. The NPG–EP cross-analysis compares route matching and agreement as well as paired task-level changes in ISR. Q3 defines its state-adaptation and intervention metrics alongside the corresponding experiments.

C Representative Multi-Trajectory Tasks

Figures 4–7 illustrate the annotated state-transition structure on all four platforms. Each example fixes one three-constraint goal and shows two complete feasible orders in full. The colored trajectories reach the same terminal constraint set through different intermediate states; dashed boxes indicate the remaining annotated feasible orders. These examples visualize why ConGUIBench evaluates the realized satisfied-constraint state rather than assuming that the linguistic mention order determines a unique valid route.

D Terminal Outcome Stability for Q1

Sample and decoding conditions. Q1 uses nine models evaluated on 118 tasks each, retaining tasks with at least two equivalent NPG mention orders. Each instruction is executed five times with

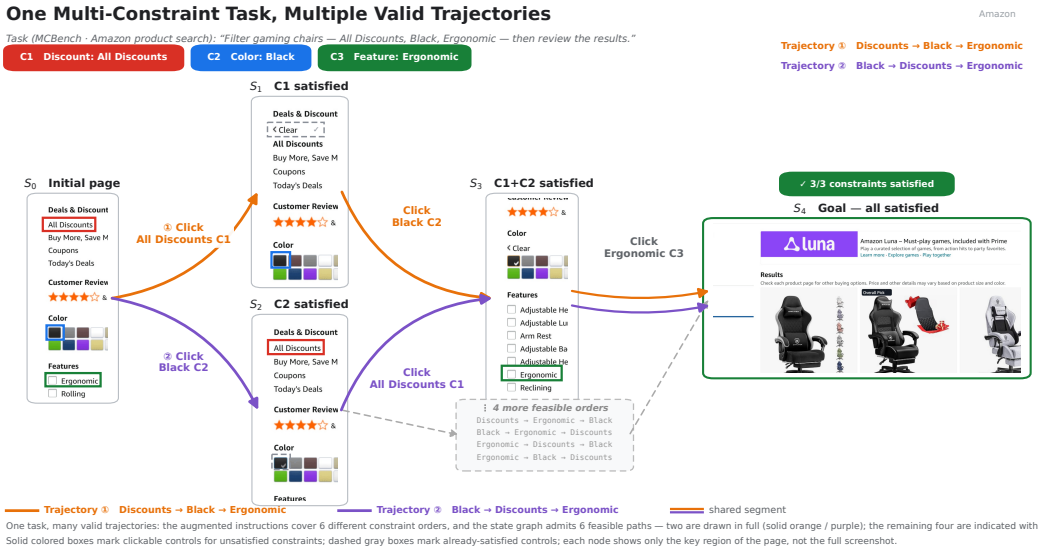

Figure 4: Representative Amazon shopping task with three constraints. The orange and purple trajectories apply the discount and color constraints in different orders before satisfying the shared ergonomic-feature constraint. Both are valid paths to the same terminal goal; the dashed list denotes the other annotated feasible orders.

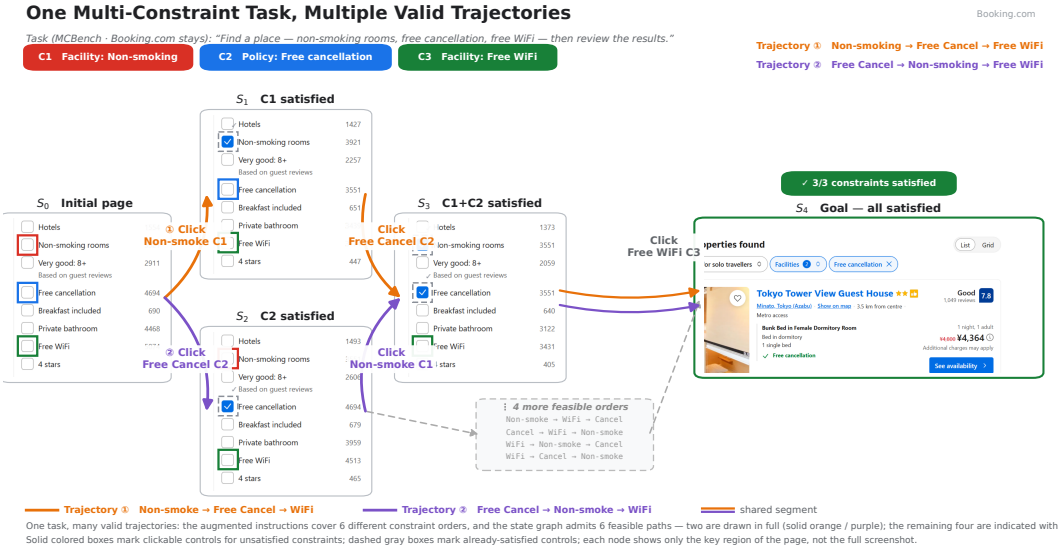

Figure 5: Representative Booking.com task with three accommodation constraints. Two feasible trajectories exchange the first two operations—non-smoking and free cancellation—and then converge on the free-WiFi constraint and the same fully satisfied goal state.

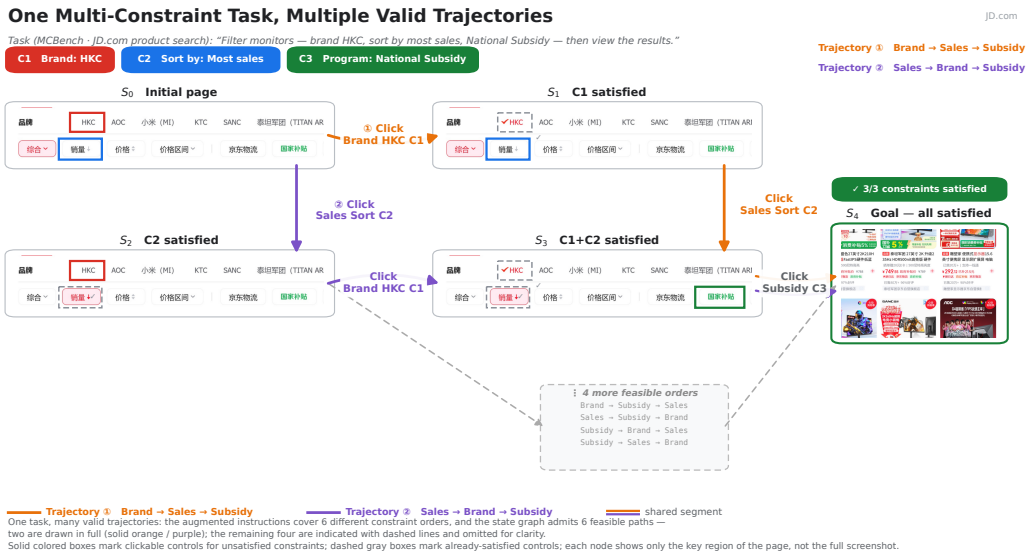

Figure 6: Representative JD shopping task with three monitor-search constraints. Two feasible trajectories exchange the first two operations—selecting the HKC brand and sorting by sales—before applying the National Subsidy program and reaching the same fully satisfied result state.

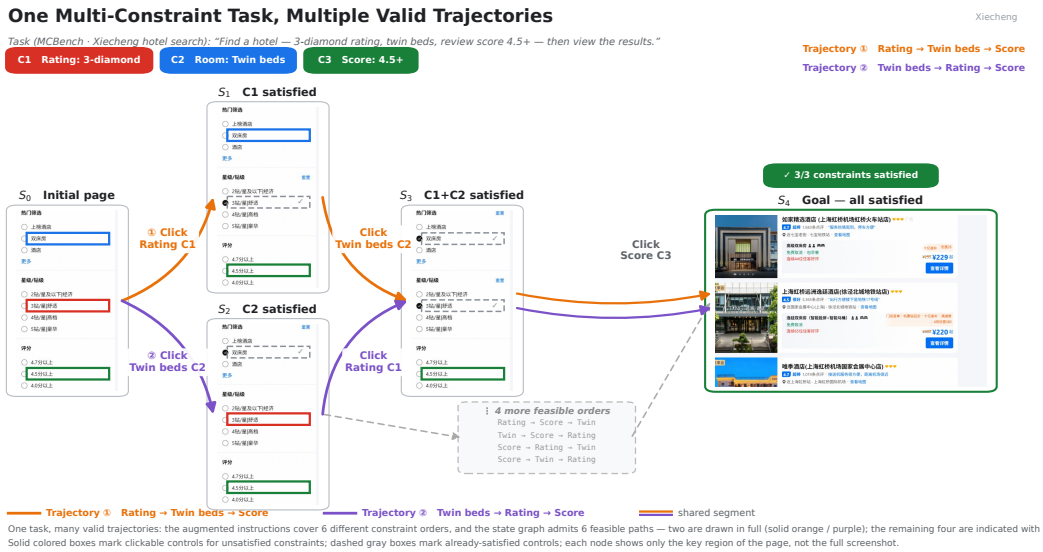

Figure 7: Representative Xiecheng task with rating, room-type, and review-score constraints. The two displayed trajectories reach the same result state through different feasible orders, while the dashed list summarizes the remaining annotated orders.

stochastic decoding using temperature 0.7, top-$p = 0.95$, and top-$k = 50$, and once with greedy decoding. The same sampling configuration is used for all models. The repeat-run analysis uses the five independent sampled executions; the greedy execution is a separate descriptive condition. For ISR and CSR, we average repetitions within each instruction, orders within each task, and then tasks equally. For greedy decoding, the first averaging step is unnecessary. These are mean single-execution outcomes, not success after selecting the best of five attempts.

Within- and cross-order estimators. Fix a model and task t, let $K_t \geq 2$ be the number of evaluated mention orders, and let $R = 5$ be the number of sampled executions per order. Write $Y_{t,i,r} \in \{0, 1\}$ for terminal instruction success under order i in repetition r. With $d_Y(u, v) = \mathbb{I}[u \neq v]$, define

$$\widehat{D}_{\text{within},t} = \frac{1}{K_t} \sum_{i=1}^{K_t} \frac{1}{\binom{R}{2}} \sum_{r<r'} d_Y(Y_{t,i,r}, Y_{t,i,r'}), \tag{9}$$

$$\widehat{D}_{\text{between},t} = \frac{1}{\binom{K_t}{2}} \sum_{i<j} \frac{1}{R^2} \sum_{r=1}^{R} \sum_{r'=1}^{R} d_Y(Y_{t,i,r}, Y_{t,j,r'}), \tag{10}$$

$$\widehat{\Delta}_{\text{order},t} = \widehat{D}_{\text{between},t} - \widehat{D}_{\text{within},t}. \tag{11}$$

Each within-order comparison uses ten distinct run pairs; each pair of orders uses 25 cross-order run pairs. Self-comparisons are excluded. We average the task-level quantities equally over the 118 tasks, so tasks with more evaluated orders do not receive greater weight.

Under independent repetitions, let $p_{t,i} = \Pr(Y_{t,i,r} = 1)$ for the fixed sampling policy. Same-order disagreement has expectation $2p_{t,i}(1-p_{t,i})$, while disagreement between orders i, j has expectation $p_{t,i} + p_{t,j} - 2p_{t,i}p_{t,j}$. Equal weighting yields

$$\mathbb{E}[\widehat{\Delta}_{\text{order},t}] = \frac{1}{\binom{K_t}{2}} \sum_{i<j} (p_{t,i} - p_{t,j})^2. \tag{12}$$

The contrast therefore targets variation in success probabilities across evaluated orders after accounting for the repeat-run disagreement baseline. Its finite-sample estimate can be negative and is not clipped at zero. It is neither a directional success-rate change nor a worst-order success gap.

Terminal constraint-state distance. Let $\mathbf{z}_{t,i,r} \in \{0, 1\}^{m_t}$ contain the terminal satisfaction labels, aligned by constraint identity across mention orders. Replace d_Y in the estimators above with

$$d_{\text{state}}(\mathbf{z}, \mathbf{z}') = \frac{1}{m_t} \sum_{k=1}^{m_t} \mathbb{I}[z_k \neq z'_k]. \tag{13}$$

This gives within- and cross-order state distances and their difference $\widehat{\Delta}_{\text{state},t}$. The expectation identity in Equation 12 then holds after additionally averaging the squared differences in each constraint's marginal satisfaction probability over constraints. This measures marginal constraint-outcome variation; it does not characterize every possible difference in the joint distribution of terminal states.

Uncertainty and scope. The reported 95% intervals use 2,000 bootstrap resamples of physical tasks with replacement, preserving all orders, repetitions, and models within each sampled block. Both disagreement estimates and their difference use the same sampled blocks. Run pairs share trajectories and are not treated as independent observations. The intervals are marginal, not simultaneous across models and metrics; we do not infer model rankings from differences between point estimates. Five repetitions support aggregate diagnostics but provide limited precision for individual order-success probabilities. Results concern the evaluated order sets and sampling configurations within the annotated execution graph; they do not identify a cognitive mechanism or extrapolate to all possible paraphrases.

Full numerical results. Table 11 supplements the sampled completion rates in Table 2 with greedy-decoding results. Table 12 retains the within-order baseline, cross-order disagreement, and excess disagreement for both terminal success and constraint states, with their reported intervals. All proportions are multiplied by 100 for presentation. Differences and constituent means are rounded separately, so displayed subtraction can differ by 0.1 points.

Table 11: Q1 completion under greedy decoding, in percent. One run is evaluated per instruction; orders are averaged within tasks before tasks are weighted equally. Sampled results are reported in Table 2.

Model	ISR	CSR
Qwen3-VL-2B	26.0	38.3
Qwen3-VL-4B	33.2	43.4
Qwen3-VL-8B	41.6	53.6
Qwen3-VL-32B	43.8	52.4
GLM-4.6V-Flash	49.9	64.3
UI-Mate-9B	39.5	51.2
UI-Mate-27B	45.0	52.6
UI-Venus-1.5-2B	36.5	49.3
UI-Venus-1.5-8B	57.3	69.4

Table 12: Q1 disagreement in terminal success and constraint states across 118 tasks per model. Each cell reports the estimate followed by its marginal 95% physical-task bootstrap interval based on 2,000 replicates. Within- and between-order values are percentages; excess disagreement is their difference in percentage points. Estimates and differences are rounded separately.

	D_{within}		D_{between}		Excess disagreement	
Model	Success	State	Success	State	Δ_{order}	Δ_{state}
Qwen3-VL-2B	9.8 [8.0,12.3]	10.3 [8.5,12.0]	18.5 [15.2,23.0]	21.7 [18.0,25.4]	8.7 [6.2,12.1]	11.4 [8.7,14.5]
Qwen3-VL-4B	9.2 [7.1,11.3]	9.5 [7.4,11.4]	18.2 [14.8,22.4]	19.2 [16.1,22.7]	8.9 [6.7,12.4]	9.7 [7.9,12.6]
Qwen3-VL-8B	4.4 [3.2,6.1]	3.4 [2.4,4.5]	18.4 [13.9,22.7]	16.0 [11.8,18.9]	14.0 [9.8,17.6]	12.7 [8.7,15.1]
Qwen3-VL-32B	6.1 [4.1,7.9]	6.3 [3.9,7.0]	12.1 [8.7,15.6]	12.0 [8.2,14.0]	6.0 [3.7,9.1]	5.8 [3.7,7.7]
GLM-4.6V-Flash	9.1 [7.3,11.8]	11.5 [9.6,13.8]	12.9 [10.4,16.6]	17.7 [14.4,21.0]	3.7 [2.1,6.1]	6.2 [3.9,8.0]
UI-Mate-9B	15.5 [13.5,19.1]	16.4 [15.0,19.5]	20.2 [17.6,25.0]	23.9 [21.4,27.4]	4.7 [2.5,8.0]	7.5 [5.2,9.3]
UI-Mate-27B	10.9 [9.1,13.8]	8.5 [6.9,10.4]	13.7 [11.2,17.7]	11.9 [9.4,14.3]	2.8 [1.1,5.2]	3.3 [1.9,4.9]
UI-Venus-1.5-2B	13.3 [11.0,16.2]	12.6 [10.2,14.1]	19.4 [16.3,23.6]	19.3 [16.0,21.6]	6.1 [4.3,8.6]	6.7 [4.7,8.8]
UI-Venus-1.5-8B	21.0 [17.2,23.2]	20.2 [16.5,21.8]	25.8 [21.9,29.3]	24.4 [20.6,26.7]	4.9 [2.9,8.1]	4.1 [3.2,6.3]

E Procedural-Order Diagnostics for Q2

Metrics and denominators. Q2 evaluates nine models on 118 physical tasks per model, with five sampled executions for each feasible EP instruction. All sampled executions use temperature 0.7, top-$p = 0.95$, and top-$k = 50$. Let $C_{t,i,r}$ indicate terminal success for task t, instruction i, and repetition r; $O_{t,i,r}$ indicate that the full first-occurrence target sequence is identifiable; and $A_{t,i,r}$ indicate that this sequence matches the requested order. For N_t instructions and $R = 5$ repetitions, the task-level metrics are

$$\mathrm{ISR}_t = \frac{1}{N_t R} \sum_{i,r} C_{t,i,r}, \tag{14}$$

$$\mathrm{SOCR}_t = \frac{1}{N_t R} \sum_{i,r} C_{t,i,r} O_{t,i,r} A_{t,i,r}, \tag{15}$$

$$\mathrm{COCR}_t = \frac{\sum_{i,r} O_{t,i,r} A_{t,i,r}}{\sum_{i,r} O_{t,i,r}}. \tag{16}$$

ISR and SOCR average all tasks equally. SOCR adds identifiable requested-order adherence to the success requirement and is therefore a subset of ISR. COCR pools identifiable runs within each task and then averages tasks with a nonzero denominator equally; tasks without an identifiable full sequence are excluded rather than assigned zero. COCR does not condition on terminal success.

E.1 Scenario-Level Sampled Results

Table 13 reports conditional order adherence by platform. Because COCR is undefined without an identifiable full sequence, each entry also reports the number of contributing physical tasks. Table 14 reports the stricter SOCR on all tasks; the corresponding platform-level ISR values appear in the

Table 13: Scenario-level conditional order compliance rate (COCR, %). Parentheses give the number of physical tasks with at least one identifiable full target sequence. Overall results are reported in Table 3.

Model	Amazon	Booking	JD	Xiecheng
Qwen3-VL-2B	73.1 (27)	100.0 (5)	79.1 (21)	100.0 (4)
Qwen3-VL-4B	88.7 (29)	98.0 (5)	84.8 (29)	95.5 (10)
Qwen3-VL-8B	88.7 (30)	100.0 (5)	89.0 (30)	88.3 (12)
Qwen3-VL-32B	94.1 (30)	100.0 (5)	95.0 (30)	97.8 (6)
GLM-4.6V-Flash	95.0 (30)	84.7 (23)	94.6 (30)	84.5 (14)
UI-Mate-9B	93.2 (30)	98.5 (22)	94.2 (30)	100.0 (1)
UI-Mate-27B	94.5 (30)	100.0 (5)	96.9 (30)	95.8 (3)
UI-Venus-1.5-2B	89.5 (30)	74.8 (14)	95.1 (27)	87.0 (11)
UI-Venus-1.5-8B	94.2 (30)	90.0 (33)	96.4 (30)	90.4 (13)

Table 14: Scenario-level successful-order compliance rate (SOCR, %) under EP. SOCR counts the subset of successful runs that also follow the requested order. Rates average sampled executions, instructions, and physical tasks within each platform. Corresponding ISR values are reported in Table 15; overall results appear in Table 3.

Model	Amazon	Booking	JD	Xiecheng
Qwen3-VL-2B	35.9	10.4	15.9	6.7
Qwen3-VL-4B	69.6	12.9	66.9	19.2
Qwen3-VL-8B	74.1	13.2	65.3	28.9
Qwen3-VL-32B	84.4	13.2	80.4	5.2
GLM-4.6V-Flash	77.5	22.0	75.3	49.7
UI-Mate-9B	79.5	20.0	60.1	2.0
UI-Mate-27B	83.8	11.8	81.1	3.3
UI-Venus-1.5-2B	66.8	15.3	59.8	7.5
UI-Venus-1.5-8B	81.3	52.6	79.2	24.9

cross-analysis (Table 15). Overall values are omitted because Table 3 already reports them with task-bootstrap intervals. The platform-level point estimates below are descriptive because corresponding intervals are unavailable.

Conditional order adherence does not ensure end-to-end control. COCR is generally high once a full target sequence is identifiable, while SOCR and the corresponding ISR vary substantially across platforms. Completion is typically stronger on Amazon and JD than on Booking and Xiecheng, and SOCR generally tracks ISR closely. This pattern indicates that ordering an identifiable sequence is often less limiting than realizing a complete successful sequence. However, several high COCR values on Booking and Xiecheng are supported by few eligible tasks—in one case only a single task—and should therefore be interpreted descriptively.

F Scenario-Level Cross-Analysis of NPG and EP

Tables 15 and 16 disaggregate the NPG–EP comparison by platform. Each cell reports NPG followed by the signed EP−NPG difference in parentheses; positive values therefore indicate a higher EP estimate.

Completion varies substantially across platforms. Amazon and JD generally yield higher ISR and CSR than Booking and Xiecheng (Table 15). Relative to EP, NPG ISR is lower for eight of nine models on Amazon, all nine on Booking, and seven of nine on JD, whereas Xiecheng is mixed and increases for six models. The aggregate EP–NPG completion difference therefore masks substantial platform heterogeneity.

Mention-order sensitivity is not confined to one platform. Under NPG, Δ_{order} is positive in 34 of the 36 model–platform cells, with one zero and one negative value, while Δ_{state} is positive in all

Table 15: Cross-analysis of completion by platform. Each cell reports the NPG rate (%) followed by the EP−NPG difference in percentage points; positive parenthetical values indicate higher completion under EP. Results average sampled executions, instructions, and tasks within each platform. Overall results are omitted.

(a) Instruction success rate (ISR, %)

Model	Amazon	Booking	JD	Xiecheng
Qwen3-VL-2B	46.9 (−2.6)	4.1 (+6.3)	27.5 (−9.4)	6.4 (+0.3)
Qwen3-VL-4B	69.3 (+7.7)	6.0 (+7.2)	54.4 (+22.9)	28.7 (−9.0)
Qwen3-VL-8B	73.5 (+9.1)	6.5 (+6.7)	53.1 (+19.8)	34.7 (−1.2)
Qwen3-VL-32B	80.6 (+9.0)	6.6 (+6.6)	77.2 (+7.0)	8.1 (−2.8)
GLM-4.6V-Flash	71.8 (+10.3)	15.6 (+7.4)	72.5 (+7.6)	52.8 (−1.4)
UI-Mate-9B	75.8 (+9.2)	17.9 (+2.1)	64.9 (−1.0)	0.0 (+2.0)
UI-Mate-27B	84.9 (+3.6)	8.6 (+3.2)	82.4 (+1.6)	3.5 (−0.2)
UI-Venus-1.5-2B	68.2 (+4.5)	11.6 (+4.3)	60.2 (+3.2)	3.6 (+4.6)
UI-Venus-1.5-8B	74.1 (+11.8)	42.8 (+12.7)	73.9 (+8.4)	31.4 (−5.1)

(b) Constraint satisfaction rate (CSR, %)

Model	Amazon	Booking	JD	Xiecheng
Qwen3-VL-2B	66.4 (−8.6)	13.5 (+2.9)	39.7 (−0.7)	15.2 (−0.9)
Qwen3-VL-4B	83.7 (−0.2)	15.4 (+2.5)	63.5 (+19.6)	53.1 (−9.0)
Qwen3-VL-8B	83.7 (+3.8)	16.4 (+2.2)	64.1 (+16.8)	62.8 (−3.6)
Qwen3-VL-32B	89.5 (+3.4)	16.6 (+2.0)	84.7 (+4.4)	18.2 (−4.5)
GLM-4.6V-Flash	81.8 (+3.9)	32.7 (+3.2)	83.0 (+3.4)	73.1 (−4.8)
UI-Mate-9B	83.4 (+6.5)	35.8 (−3.7)	76.4 (−0.7)	4.2 (+1.8)
UI-Mate-27B	90.9 (+0.8)	15.9 (−0.3)	87.3 (+2.4)	9.3 (−0.8)
UI-Venus-1.5-2B	82.6 (0.0)	22.6 (−1.6)	77.0 (+0.9)	19.2 (+4.1)
UI-Venus-1.5-8B	82.9 (+5.7)	58.2 (+3.7)	83.0 (+5.3)	46.7 (−4.8)

36 cells (Table 16). Magnitudes vary across models and platforms and are often largest on Amazon or JD. Mention-order sensitivity is therefore present across all four platforms rather than driven by a single scenario.

G State-Adaptation Diagnostics for Q3

State-conditioned selection (A). *Valid-next* means that the first identifiable target is pending and does not repeat a completed constraint; ASRR measures such completed-constraint repetitions. These diagnostics concern whether an action target is still pending, whereas Q2 compares the sequence of first distinct target attempts with a requested order. SAG is the paired valid-next difference between a partial state and the same task's empty-state control. It measures the change in pending-target selection associated with visible partial progress, not direct evidence of an internal checklist. Table 17 reports the available model-level point estimates.

No-effect intervention (B). The intervention is applicable only when an initial evaluator-valid action can be suppressed. Applicability is reported separately from the locally consistent response rate, which is conditional on applicable cases. A locally consistent response means either retrying the still-pending target or switching to another pending constraint; it does not require eventual completion of the suppressed target. The paired B−A completion difference is evaluated on applicable matched pairs; negative values indicate lower completion under the no-effect intervention.

Analysis support and aggregation. Q3 reports nine models on L2 tasks from Amazon, Booking, JD, and Xiecheng. The main analysis uses 5,167 common complete A/B conditions per model, with 2,564–4,184 applicable B pairs. Eight models have complete raw coverage; UI-Mate-9B is missing 50 of 10,434 expected records. Estimates are task-equal; SAG and paired B−A are absolute percentage-point differences. The available summary contains point estimates, and the physical-task bootstrap interval endpoints remain to be restored before inferential claims are made.

Table 16: Cross-analysis of order sensitivity by platform. Each cell reports NPG (EP−NPG). Values are proportions: $\Delta_{\text{order}} = D_{\text{between}} - D_{\text{within}}$, and Δ_{state} is the analogous excess terminal-state distance. Overall results and platform-specific uncertainty intervals are omitted.

(a) Excess terminal-success disagreement (Δ_{order})

Model	Amazon	Booking	JD	Xiecheng
Qwen3-VL-2B	0.108 (+0.109)	0.003 (+0.037)	0.221 (−0.108)	0.014 (+0.024)
Qwen3-VL-4B	0.157 (+0.045)	0.035 (−0.035)	0.127 (+0.004)	0.035 (+0.074)
Qwen3-VL-8B	0.220 (−0.008)	0.026 (−0.026)	0.206 (−0.006)	0.136 (−0.083)
Qwen3-VL-32B	0.097 (+0.044)	0.026 (−0.026)	0.101 (−0.075)	0.009 (−0.007)
GLM-4.6V-Flash	0.040 (+0.066)	0.040 (−0.027)	0.054 (0.000)	0.006 (+0.002)
UI-Mate-9B	0.065 (−0.005)	0.076 (−0.073)	0.024 (−0.013)	0.000 (+0.030)
UI-Mate-27B	0.025 (+0.021)	0.058 (−0.042)	0.001 (+0.009)	0.017 (−0.009)
UI-Venus-1.5-2B	0.051 (+0.089)	0.041 (−0.043)	0.133 (−0.055)	0.005 (−0.003)
UI-Venus-1.5-8B	0.037 (+0.028)	0.108 (−0.017)	0.038 (−0.008)	−0.029 (+0.046)

(b) Excess terminal-state distance (Δ_{state})

Model	Amazon	Booking	JD	Xiecheng
Qwen3-VL-2B	0.100 (+0.164)	0.013 (+0.032)	0.294 (+0.036)	0.054 (+0.043)
Qwen3-VL-4B	0.113 (+0.082)	0.033 (−0.003)	0.175 (−0.024)	0.075 (+0.201)
Qwen3-VL-8B	0.167 (−0.013)	0.021 (+0.002)	0.203 (+0.004)	0.152 (+0.001)
Qwen3-VL-32B	0.055 (+0.040)	0.019 (+0.007)	0.103 (−0.033)	0.066 (+0.054)
GLM-4.6V-Flash	0.041 (+0.086)	0.090 (+0.065)	0.051 (+0.050)	0.057 (+0.080)
UI-Mate-9B	0.050 (+0.016)	0.110 (−0.014)	0.059 (+0.028)	0.073 (+0.057)
UI-Mate-27B	0.013 (+0.032)	0.041 (+0.003)	0.020 (+0.003)	0.070 (+0.026)
UI-Venus-1.5-2B	0.037 (+0.111)	0.037 (−0.019)	0.128 (0.000)	0.078 (+0.036)
UI-Venus-1.5-8B	0.023 (+0.046)	0.079 (+0.070)	0.036 (+0.032)	0.006 (+0.118)

Table 17: Supporting Q3 state-adaptation statistics. A completion, valid-next, ASRR, and B applicability are percentages. The main Q3 table reports SAG, local response, and the paired B−A completion change.

Model	A completion	A valid-next	A ASRR	B applicability
Qwen3-VL-2B	45.1	62.6	5.7	61.7
Qwen3-VL-4B	58.0	75.2	3.8	75.8
Qwen3-VL-8B	62.9	80.5	3.9	80.8
Qwen3-VL-32B	61.7	75.7	2.9	75.3
GLM-4.6V-Flash	60.4	76.8	5.3	79.0
UI-Mate-9B	48.3	72.1	7.5	60.7
UI-Mate-27B	52.9	77.1	6.2	67.1
UI-Venus-1.5-2B	50.7	67.3	8.1	68.2
UI-Venus-1.5-8B	61.3	74.6	4.0	75.8

Oracle-state follow-up. Static prompting repeats only the full task instruction at each step, whereas Dynamic prompting additionally supplies Oracle *satisfied/pending* labels for every constraint. In the Blocked conditions, the first evaluator-valid click is suppressed, the GUI remains unchanged, and a no-effect observation is injected into the execution history; later clicks proceed normally, and the run receives one extra step. Table 18 reports the four condition-specific ISR values underlying the drops and mitigation summarized in Table 5(b).

H Per-Model Error Profiles

Tables 19 and 20 provide the per-model results underlying the model-equal summary in Table 6. As in the scenario-level cross-analysis, each rate cell reports the NPG value followed by the signed EP−NPG difference in parentheses; positive values therefore indicate a higher error share under EP. In Table 20, all subtype rates are conditional on the *Other* category.

Table 18: Condition-specific ISR (%) for the Q3 Oracle-state follow-up. These values underlie the Static and Dynamic drops and their mitigation in Table 5(b). They are descriptive point estimates from a separate nine-model sample; paired intervals are not yet available.

Model	Static Normal	Static Blocked	Dynamic Normal	Dynamic Blocked
Qwen3-VL-2B	10.0	5.8	17.4	11.9
Qwen3-VL-4B	42.9	35.8	45.1	39.6
Qwen3-VL-8B	48.0	38.8	47.1	43.1
Qwen3-VL-32B	42.1	38.6	42.3	41.1
GLM-4.6V-Flash	58.0	27.5	57.7	46.4
UI-Mate-9B	36.6	13.0	41.8	30.3
UI-Mate-27B	46.6	29.7	49.9	42.2
UI-Venus-1.5-2B	37.1	26.1	39.7	33.4
UI-Venus-1.5-8B	59.7	39.0	61.8	42.7

Table 19: Per-model error-type distributions under NPG and EP. Each rate cell reports the NPG rate (%) followed by the EP−NPG difference in percentage points; positive parenthetical values indicate a higher error share under EP.

Model	Grounding near miss	Completed-target repetition	Search bar/ header	Product/room region	Other
Qwen3-VL-2B	28.1 (−5.9)	11.5 (−1.8)	3.4 (+2.1)	11.0 (+8.2)	46.0 (−2.6)
Qwen3-VL-4B	21.8 (+3.0)	21.7 (−7.4)	1.9 (+3.0)	19.1 (−9.0)	35.4 (+10.5)
Qwen3-VL-8B	11.2 (+4.0)	20.8 (−4.7)	5.5 (+0.4)	11.5 (−2.8)	51.0 (+3.1)
Qwen3-VL-32B	43.3 (+3.5)	12.7 (−2.4)	0.3 (+1.1)	1.0 (−0.8)	42.6 (−1.3)
GLM-4.6V-Flash	11.3 (+0.8)	7.2 (+2.7)	8.0 (+4.0)	11.1 (−4.5)	62.4 (−3.0)
UI-Mate-9B	33.2 (−1.5)	9.3 (+3.3)	4.0 (−0.4)	8.5 (−3.6)	44.9 (+2.2)
UI-Mate-27B	35.4 (−0.8)	10.3 (−2.0)	1.4 (−0.7)	3.6 (−2.2)	49.3 (+5.7)
UI-Venus-1.5-2B	28.2 (+0.4)	21.9 (−4.0)	2.5 (−0.4)	3.4 (+3.1)	44.0 (+0.9)
UI-Venus-1.5-8B	36.2 (+6.0)	11.0 (−4.3)	1.7 (+2.0)	6.4 (−2.8)	44.6 (−0.8)

I Detailed Results for the Checklist Intervention

The main checklist analysis compares CL and noCL using the same sampling configuration—temperature 0.7, top-$p = 0.95$, and top-$k = 50$—and only instructions observed under both conditions. Each instruction has five repeated runs. The matched set contains 688 instructions per model and format, except for UI-Venus-1.5-8B under EP, for which 671 instructions are shared. The primary CL condition uses the benchmark's ground-truth atomic constraint list and a 4B auditor to provide updated *satisfied/pending* labels. Displayed rates are rounded to one decimal place, whereas differences are computed from unrounded values. All differences below are descriptive point estimates because paired uncertainty intervals are not yet available.

I.1 Completion Accuracy

Tables 21 and 22 report the absolute CL and noCL completion rates underlying the ISR gains in Table 7. They additionally provide per-model CSR changes, whose model-equal gains are 3.1 points under EP and 5.8 points under NPG.

I.2 Across-Replicate Stability

Table 23 complements the ISR stability results in Table 7 with CSR stability. We define $G_\sigma = \sigma_{\mathrm{noCL}} - \sigma_{\mathrm{CL}}$, so positive values indicate that CL lowers the across-replicate standard deviation. The model-equal CSR reductions are 0.6 points under EP and 1.9 points under NPG; negative entries show that the benefit is not universal.

Table 20: Per-model diagnostic subtypes within the *Other* error category under NPG and EP. Each rate cell reports the conditional NPG rate (%) followed by the EP−NPG difference in percentage points; positive parenthetical values indicate a higher subtype share under EP. The listed subtypes are not exhaustive.

Model	Requires scrolling	No clickable gold target	Wide near-miss total	Other-screen hit
Qwen3-VL-2B	31.4 (−1.0)	0.6 (+3.3)	51.4 (−11.1)	0.6 (+3.7)
Qwen3-VL-4B	49.5 (−7.9)	5.8 (+9.0)	32.1 (−4.2)	0.0 (+4.6)
Qwen3-VL-8B	61.6 (−6.1)	2.8 (+6.0)	30.7 (−5.6)	0.2 (+5.1)
Qwen3-VL-32B	73.3 (−20.2)	1.5 (+3.1)	21.3 (+10.3)	0.6 (+3.8)
GLM-4.6V-Flash	45.8 (−7.3)	8.6 (+12.5)	32.0 (−7.7)	0.0 (+2.4)
UI-Mate-9B	45.9 (−1.3)	2.9 (+0.1)	36.2 (+3.0)	3.4 (−0.3)
UI-Mate-27B	62.5 (−15.4)	0.9 (+0.5)	27.3 (+7.1)	3.5 (+5.6)
UI-Venus-1.5-2B	53.6 (−5.5)	1.5 (+0.7)	30.7 (−3.4)	1.7 (+5.8)
UI-Venus-1.5-8B	42.4 (−3.8)	0.7 (+0.4)	46.3 (−3.7)	1.9 (+5.7)

Table 21: Matched checklist completion results under EP. Rates are percentages; ΔCSR is the CL−noCL percentage-point change. Per-model ΔISR is reported in Table 7. The final row averages models equally.

Model	noCL ISR	CL ISR	noCL CSR	CL CSR	ΔCSR
Qwen3-VL-2B	20.4	29.9	32.3	41.5	+9.2
Qwen3-VL-4B	46.8	48.1	55.6	56.4	+0.8
Qwen3-VL-8B	49.4	50.9	58.9	58.4	−0.5
Qwen3-VL-32B	49.3	51.5	54.6	56.9	+2.4
GLM-4.6V-Flash	57.4	57.1	66.9	66.9	0.0
UI-Mate-9B	44.6	52.5	53.4	60.6	+7.1
UI-Mate-27B	48.2	46.9	52.6	52.5	−0.1
UI-Venus-1.5-2B	41.1	47.1	51.5	54.9	+3.4
UI-Venus-1.5-8B	66.8	73.0	73.3	78.6	+5.3
Model-equal mean	**47.1**	**50.8**	**55.5**	**58.5**	**+3.1**

I.3 Sensitivity to the Constraint Source

The primary analysis supplies the auditor with ground-truth constraints. A separate EP-only comparison replaces them with constraints extracted by the auxiliary parser while keeping the 4B auditor. Table 24 shows that parsed constraints remain within two ISR points of ground truth for seven of nine models, but the effect is model dependent; UI-Venus-1.5-8B loses 5.3 ISR points. Constraint extraction is therefore a smaller bottleneck than checklist tracking for most models, but cannot be treated as lossless.

J Prompt Templates for the Checklist Intervention

The checklist pipeline has three components. An optional constraint parser converts the task instruction and starting-page screenshot into independently verifiable constraints; the primary Ground-Truth condition bypasses this stage and uses the benchmark constraint list. A 4B checklist auditor then assigns each constraint one of *satisfied*, *unsatisfied*, or *uncertain* from the current UI evidence. Finally, these judgments are rendered as `DONE`, `TODO`, and `UNSURE` and injected into the acting agent's context. The complete text templates from `constraint_parser_prompt.py` and `checklist_prompt.py` are reproduced below. Placeholders enclosed in braces are populated at runtime; typographic dashes are normalized for LaTeX listings.

Table 22: Matched checklist completion results under NPG. Rates are percentages; ΔCSR is the CL−noCL percentage-point change. Per-model ΔISR is reported in Table 7. The final row averages models equally.

Model	noCL ISR	CL ISR	noCL CSR	CL CSR	ΔCSR
Qwen3-VL-2B	21.3	36.2	33.9	46.2	+12.2
Qwen3-VL-4B	38.2	48.8	51.4	57.8	+6.4
Qwen3-VL-8B	40.1	50.1	53.5	57.1	+3.6
Qwen3-VL-32B	43.6	51.5	52.7	57.8	+5.1
GLM-4.6V-Flash	50.6	55.3	64.8	67.4	+2.6
UI-Mate-9B	41.5	50.1	52.9	59.5	+6.6
UI-Mate-27B	45.9	47.3	52.0	53.8	+1.7
UI-Venus-1.5-2B	37.0	49.1	51.1	58.1	+7.0
UI-Venus-1.5-8B	56.7	69.0	68.9	75.6	+6.8
Model-equal mean	**41.7**	**50.8**	**53.5**	**59.3**	**+5.8**

Table 23: Checklist stability gains for CSR under EP and NPG. G_σ^{CSR} is the noCL-minus-CL change in across-replicate standard deviation, expressed in percentage points; positive values indicate more stable execution with CL. ISR stability gains are reported in Table 7.

Model	EP G_σ^{CSR}	NPG G_σ^{CSR}
Qwen3-VL-2B	−0.8	+3.1
Qwen3-VL-4B	+1.0	+3.8
Qwen3-VL-8B	−0.5	−0.1
Qwen3-VL-32B	−0.8	+1.9
GLM-4.6V-Flash	+0.6	+1.5
UI-Mate-9B	+1.3	−0.9
UI-Mate-27B	−0.8	−0.1
UI-Venus-1.5-2B	+2.3	+0.7
UI-Venus-1.5-8B	+2.7	+7.1
Model-equal mean	**+0.6**	**+1.9**

J.1 Constraint Parsing

At the beginning of an episode, the parser receives the full instruction and the starting-page screenshot. It produces an atomic, screenshot-grounded constraint list in JSON. The system and user prompts are given below.

Listing 1: Constraint-parser system prompt.

```
You are a precise requirement analyst. You do NOT operate any interface. You will be given a
    natural-language task instruction for a GUI agent, together with a screenshot of the task
    's STARTING page. Your job is to decompose the instruction into a checklist of
    independent, individually-verifiable constraints --- the kind of atomic conditions that
    could later be checked one by one against a screenshot (e.g., a filter being applied, an
    option being selected, a value being entered). CRITICAL: you must actually look at the
    screenshot and ground each constraint in what is really available on that page --- use
    the screenshot to resolve vague or colloquial phrases in the instruction (e.g., 'a cheap
    option', 'a popular brand', 'the second tab') into the concrete filter/option/control
    names, labels, or values that are visibly present on the page, instead of guessing from
    the instruction text alone. If the instruction references something that cannot be found
    on the page, keep the constraint as close as possible to the instruction's original
    wording rather than inventing a page element that doesn't exist. Do not add constraints
    that are not implied by the instruction, and do not merge multiple distinct constraints
    into one item.
```

Listing 2: Constraint-parser user prompt.

```
### Task Instruction
{instruction}

### Starting Page Screenshot
```

Table 24: EP-only sensitivity to the checklist constraint source. Rates are percentages; differences are Model-Parsed minus Ground-Truth in percentage points. This auxiliary comparison is separate from the five-run matched analysis above.

Model	GT ISR	Parsed ISR	ΔISR	GT CSR	Parsed CSR	ΔCSR
Qwen3-VL-2B	31.3	32.3	+1.0	43.1	43.2	+0.1
Qwen3-VL-4B	49.8	47.8	−2.0	57.0	56.1	−0.9
Qwen3-VL-8B	51.1	51.9	+0.8	58.2	59.4	+1.2
Qwen3-VL-32B	52.0	49.8	−2.2	57.3	54.9	−2.4
GLM-4.6V-Flash	57.5	55.9	−1.6	65.8	64.6	−1.2
UI-Mate-9B	48.1	46.7	−1.4	56.3	55.0	−1.3
UI-Mate-27B	43.6	43.5	−0.1	51.0	49.9	−1.1
UI-Venus-1.5-2B	46.9	48.3	+1.4	55.0	56.1	+1.1
UI-Venus-1.5-8B	69.4	64.1	−5.3	75.0	69.6	−5.4
Model-equal mean	**50.0**	**49.0**	**−1.0**	**57.6**	**56.5**	**−1.1**

```
The image below is the screenshot of the page BEFORE any action has been taken (the task's
     starting state). Use it to ground your constraints in the actual filters/options/controls
      visible on this page.

### Your Job
Break the instruction above into a checklist of independent constraints, grounding each one in
      what you can actually see on the starting page screenshot. Each constraint should be:
  - Atomic: describes exactly ONE condition (e.g., one filter value, one form field, one
       selected option) --- do not bundle multiple conditions into one item.
  - Grounded: whenever the instruction's wording is vague or colloquial, resolve it against the
        concrete filter/option/control names, labels, or values that are actually visible in
       the screenshot (do not invent page elements that are not visible; if truly unresolvable
       from the screenshot, fall back to the instruction's own wording).
  - Verifiable: phrased so that, given a screenshot of the resulting page, one could visually
       check whether it is satisfied or not.
  - Concise: a short phrase, not a full sentence restating the whole instruction.
  - Faithful: only include constraints that are actually implied by the instruction; do not
       invent extra requirements.

If the instruction only describes a single simple action with no separately checkable sub-
     conditions, return exactly one constraint that captures it.

### Output Format (STRICT JSON ONLY --- no <tool_call>, no markdown fences, no extra text
     before or after)
{
  "constraints": ["<constraint 1>", "<constraint 2>", ...]
}
```

J.2 Checklist Auditing

The auditor never acts on the interface. At the initial step it receives the current screenshot, the full instruction, the constraint list, and the action history. From the second step onward, the implementation can instead provide screenshots from immediately before and after the latest action. The auditor must verify visible state rather than infer success from the attempted action or from matching items in an otherwise unfiltered result list. A malformed response is treated as *uncertain* by the execution pipeline.

J.2.1 Single-Image Mode

Listing 3: Checklist-auditor system prompt for single-image state verification.

```
You are a meticulous UI state auditor. You do NOT operate the interface. Your only job is to
     look at the current screenshot and decide, for each constraint in a checklist, whether it
      is currently satisfied on the page, based only on visible evidence (e.g., an applied
     filter chip, a selected option highlighted, a value shown in a form field). Never assume
     a constraint is satisfied just because a previous action mentioned it --- verify it
     against the CURRENT screenshot. CRITICAL: you must distinguish TWO very different kinds
     of evidence: (1) VALID evidence --- the filter/search control itself is visibly selected,
      checked, highlighted, or its value is shown as an active applied filter chip; (2)
     INVALID evidence --- some items in an unfiltered result list happen to have an attribute/
     label that matches the constraint (e.g., a product card shows a 'brand' or 'promo' tag)
```

```
    even though no filter control shows as selected. An unfiltered list almost always
    contains a mix of items, so a matching item appearing in the list does NOT mean the
    constraint has been applied --- only the control's own selected/applied state counts.
    When in doubt, prefer 'unsatisfied' or 'uncertain' over 'satisfied'. IMPORTANT: you have
    NO tools and NO functions available, and you must NOT call any tool or function. Do NOT
    output a <tool_call> block under any circumstances (e.g., never invent or call something
    like 'compare_images' or any other tool). You must directly output your final answer as
    raw JSON text --- nothing else.
```

Listing 4: Checklist-auditor user prompt for single-image state verification.

```
### Overall User Task
{instruction}

### Full Constraint Checklist (independent constraints, any order)
{constraint_list}

### Action History So Far (may be incomplete or unreliable; verify against the screenshot)
{history_str}
{first_step_warning}
### Your Job
For EACH constraint above, inspect the CURRENT screenshot and decide its status:
 - "satisfied": there is clear visible evidence on the current page that this constraint has
     been applied/selected/entered.
 - "unsatisfied": there is clear visible evidence that this constraint has NOT been applied
     yet.
 - "uncertain": the screenshot does not provide enough evidence either way.

Be conservative: only mark "satisfied" when you can point to concrete visual evidence that a
    filter/search CONTROL itself is selected/applied (e.g., a filter chip, a checked checkbox
    /radio, text visibly entered in a field). Do NOT mark "satisfied" just because some item
    in an unfiltered list happens to have a matching attribute --- that is not evidence the
    constraint's control has been applied. Do not mark "satisfied" purely because the history
     log claims so.

You have NO tools/functions available. Do NOT output a <tool_call> block or try to call any
    function (e.g., do not invent a tool like 'compare_images') --- just look at the
    screenshot yourself and answer directly.

### Output Format (STRICT JSON ONLY --- no <tool_call>, no markdown fences, no extra text
    before or after)
{
  "checklist": [
    {"constraint": "<constraint text exactly as given above>", "status": "satisfied|unsatisfied
        |uncertain", "evidence": "<one short sentence, or empty string if uncertain>"}
    ... one entry per constraint, same order as given ...
  ]
}
```

At step 0, the `first_step_warning` placeholder is replaced by the following guard against prematurely marking filters as complete. It is empty on later single-image calls.

Listing 5: Step-0 warning inserted into the single-image auditor prompt.

```
### IMPORTANT: This Is Step 0 --- No Action Has Been Taken Yet
The user has NOT performed any action so far; this screenshot is the task's STARTING page. It
    is therefore extremely unlikely that any filter/search-type constraint is already
    satisfied --- do NOT be misled by items in the (unfiltered) result list that happen to
    match a constraint's attribute; that is normal noise in any unfiltered list, not evidence
     of an applied filter. Only mark a constraint "satisfied" at this step if the starting
    page ITSELF, by construction, already meets it (e.g., the instruction only asks to be on
    a certain site/page and the screenshot confirms that). Otherwise, prefer "unsatisfied" or
     "uncertain".
```

J.2.2 Before/After Difference Mode

The difference mode supplies two images in order: the state immediately before the last action and the current state immediately after it. The action history excludes the final action, which is supplied separately through `last_action_str`.

Listing 6: Checklist-auditor system prompt for before/after state verification.

```
You are a meticulous UI state auditor. You do NOT operate the interface. You will be shown TWO
     screenshots in order: Image 1 = the screen BEFORE the agent's last action, Image 2 = the
```

```
     screen AFTER the agent's last action (i.e., the CURRENT screen). You will also be told
    what the last action was. Your job is to compare the two screenshots to see exactly what
    changed, and decide, for each constraint in a checklist, whether it is currently
    satisfied on Image 2 (the current page), based only on visible evidence. Use the before/
    after comparison to verify whether the last action actually produced the intended effect
    --- an action that was attempted does not mean it succeeded; only trust what you can see
    changed (or stayed unsatisfied) in Image 2. Never assume a constraint is satisfied just
    because a previous action mentioned it --- verify it against the CURRENT (Image 2)
    screenshot. CRITICAL: you must distinguish TWO very different kinds of evidence: (1)
    VALID evidence --- the filter/search control itself is visibly selected, checked,
    highlighted, or shown as an active applied filter chip, AND this selected state is new in
     Image 2 (or was already selected and remains so); (2) INVALID evidence --- some items in
     the result list happen to have an attribute/label that matches the constraint (e.g., a
    product card shows a 'brand' or 'promo' tag) even though no filter control shows as
    selected in Image 2. An unfiltered or partially filtered list almost always contains a
    mix of items, so a matching item appearing in the list does NOT mean the constraint has
    been applied --- only the control's own selected/applied state counts. When in doubt,
    prefer 'unsatisfied' or 'uncertain' over 'satisfied'. IMPORTANT: you have NO tools and NO
     functions available, and you must NOT call any tool or function to compare the two
    images --- YOU must do the comparison yourself by looking at them. Do NOT output a <
    tool_call> block under any circumstances (e.g., never invent or call something like '
    compare_images' or any other tool). You must directly output your final answer as raw
    JSON text --- nothing else.
```

Listing 7: Checklist-auditor user prompt for before/after state verification.

```
### Overall User Task
{instruction}

### Full Constraint Checklist (independent constraints, any order)
{constraint_list}

### Action History Before The Last Step (may be incomplete or unreliable; verify against the
     screenshots)
{history_str}

### Last Action Just Executed (between Image 1 and Image 2)
{last_action_str}

### Images
Image 1: the screen BEFORE the last action above.
Image 2: the screen AFTER the last action above (this is the CURRENT screen).

### Your Job
Compare Image 1 -> Image 2 to identify what actually changed on screen as a result of the last
      action. Then, for EACH constraint above, decide its status on the CURRENT screen (Image
     2):
  - "satisfied": there is clear visible evidence in Image 2 that this constraint has been
       applied/selected/entered.
  - "unsatisfied": there is clear visible evidence in Image 2 that this constraint has NOT been
        applied yet.
  - "uncertain": Image 2 does not provide enough evidence either way.

Be conservative: only mark "satisfied" when you can point to concrete visual evidence in Image
      2 (e.g., a filter tag showing the value, a checkbox/radio visibly selected, text visibly
      entered in a field). If the last action targeted a constraint but the before/after
     comparison shows no actual change on screen, treat it as still "unsatisfied" or "
     uncertain" rather than trusting the action's intent.

You have NO tools/functions available. Do NOT output a <tool_call> block or try to call any
     function (e.g., do not invent a tool like 'compare_images') --- YOU must compare the two
     images yourself and answer directly.

### Output Format (STRICT JSON ONLY --- no <tool_call>, no markdown fences, no extra text
     before or after)
{
  "checklist": [
    {"constraint": "<constraint text exactly as given above>", "status": "satisfied|unsatisfied
         |uncertain", "evidence": "<one short sentence describing what changed or didn't, or
        empty string if uncertain>"}
    ... one entry per constraint, same order as given ...
  ]
}
```

J.3 CHECKLIST INJECTION INTO THE ACTING AGENT

The parsed auditor output is converted to a compact natural-language block. The status mapping is `satisfied`→`DONE`, `unsatisfied`→`TODO`, and `uncertain`→`UNSURE`; available evidence is appended to the corresponding constraint. An empty auditor result produces no injected block.

Listing 8: Checklist block injected into the acting agent's context.

```
### Constraint Checklist (verified against the CURRENT screenshot by an independent auditor
    --- trust this over your own memory of history)
{checklist_lines}

Guidance:
- Constraints marked [DONE] are already satisfied on the current page. Do NOT repeat actions
    for them.
- Constraints marked [TODO] are NOT satisfied yet. Prioritize these.
- Constraints marked [UNSURE] could not be verified from the screenshot; re-check carefully
    before deciding they are done.
```