Same Tasks, Different Apps: Why Mobile GUI Agents Fail to Generalize?

Anonymous EMNLP submission

Abstract

Mobile GUI agents deployed in real settings must work across different applications that support the same functionality. However, existing mobile GUI benchmarks typically evaluate each task in a single application, leaving it unclear whether strong performance reflects transferable functional understanding or adaptation to one interface. We introduce **CATBench**, a category-controlled live Android benchmark designed to evaluate cross-application generalization while keeping the user goal fixed. CATBench spans 10 functional categories, 100 task templates, and 520 task–application combinations across 11 canonical applications and 41 alternative applications selected for interface diversity. Beyond measuring success, CATBench is designed to diagnose why agents fail to generalize. We evaluate agents both from raw user instructions and with externally provided application-agnostic sub-goals, then use a human-validated LLM-as-judge pipeline to attribute failed trajectories to planning, grounding/execution, mixed, or other failures. Our experiments show that agent performance varies substantially across applications within the same category: success on canonical applications does not reliably transfer to alternatives supporting the same goal, and the dominant failure mode often changes with the target interface. These findings highlight the need for category-controlled cross-application evaluation when measuring deployment-ready mobile GUI agents.

1 Introduction

Mobile GUI agents have improved rapidly, with recent systems exceeding 90% task success on live Android benchmarks (Kong et al., 2025). Yet deployment requires more than success on one interface: an agent should send a message, add a calendar event, or complete a to-do item across any application that supports the same user goal.

Existing benchmarks do not isolate this capability. Offline datasets such as Android-in-the-Wild (Rawles et al., 2023), AndroidControl (Li et al., 2024), and AMEX (Chai et al., 2025a) evaluate fixed states or recorded trajectories. Online benchmarks such as AndroidWorld (Rawles et al., 2025), AndroidLab (Xu et al., 2025b), and SPA-Bench (Chen et al., 2025a) support live interaction but bind each task to a single canonical application, conflating task difficulty, agent strategy, and interface-specific adaptation. Transfer-oriented benchmarks vary different axes: GUI-Odyssey (Lu et al., 2025) studies multi-application workflows, TransBench (Lu et al., 2025) focuses on static GUI grounding, and FedMABench (Wang et al., 2025) uses app heterogeneity for federated training rather than controlled same-task evaluation. Thus, existing benchmarks do not hold the functional task fixed while varying the application, and aggregate success rates provide limited diagnostic signal about planning, grounding, or execution failures.

To address this gap, we introduce **CATBench**, a category-controlled live Android benchmark for within-category cross-application generalization. CATBench defines reusable task templates with fixed parameters, initialization logic, and deterministic verifiers across 10 functional categories: Tasks, Notes, Finance, Music, Calendar, SMS, File Manager, Maps, Contacts, and Clock. Each template is instantiated across structurally diverse applications that support the same user-facing goal, yielding 100 task templates, 52 runnable apps, and 520 task–application combinations. For diagnosis, CATBench evaluates each instance under direct execution and execution with externally generated, application-independent sub-goals, then attributes failed rollouts using a human-validated LLM-as-judge pipeline.

Across 13 baselines, CATBench reveals three findings. First, cross-application transfer is weak: agents achieve **35.3%** success on AndroidWorld-

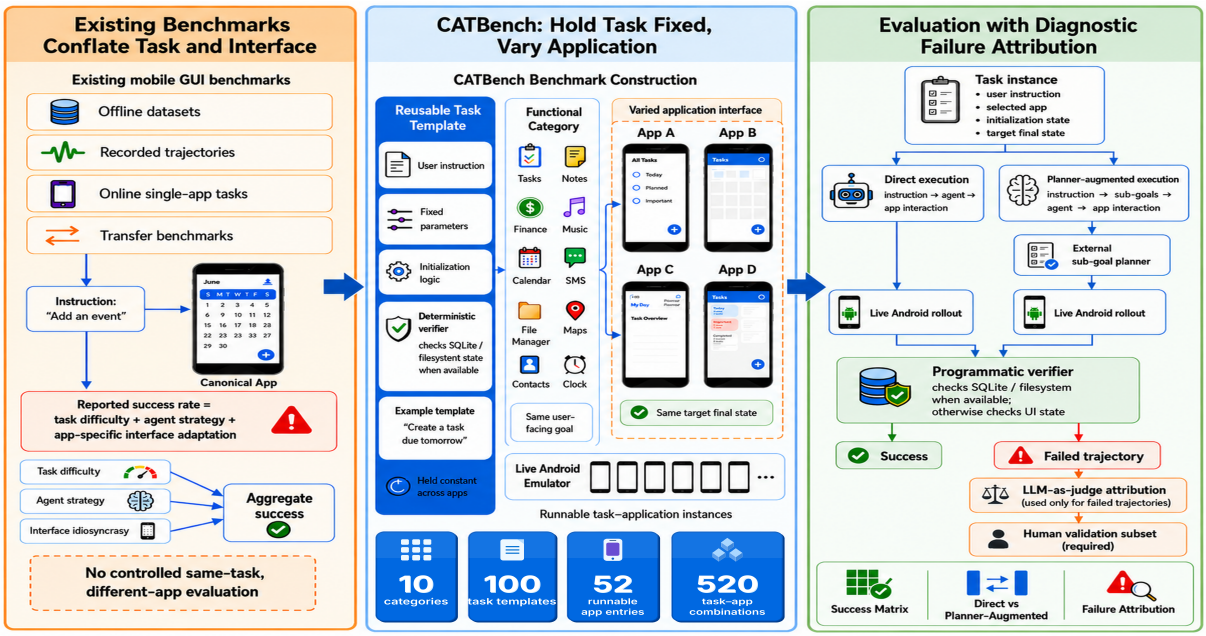

Figure 1: **Overview of CATBench.** Existing mobile GUI benchmarks bind each task to one application, conflating task difficulty with interface-specific adaptation. CATBench fixes the user instruction and target final state while varying the application within a functional category. It evaluates both direct and planner-augmented execution, attributes trajectory-level failures, and spans 520 task–application combinations across 10 categories.

original apps but only **22.3%** on newly installed same-category apps, a **13.1**-point drop and **63.2%** retention. Second, performance is strongly app-dependent: the same semantic goal can become easier or harder depending on layout, labels, control placement, and workflow conventions, with strong agents often showing **20–35**-point app-level standard deviations. Third, application-independent sub-goals provide only modest gains, improving success from **21.6%** to **22.1%** and failing to close the transfer gap. Failure analysis shows that agents differ in their bottlenecks: some are planning-limited, others are grounding-limited, and many fail through both. These results show that deployment-ready mobile agents require robust grounding, execution recovery, and task decomposition under UI variation.

Our contributions are:

- We introduce **CATBench**, a category-controlled benchmark that isolates within-category cross-application generalization by fixing the task and target final state while varying the application.
- We build a live Android suite with 10 categories, 100 task templates, 52 apps, and 520 task–application combinations selected for interface diversity.
- We establish a diagnostic protocol that compares direct execution with sub-goal-augmented execution and attributes failed trajectories using a human-validated LLM-as-judge pipeline.
- We evaluate 13 baselines and show that mobile GUI agents remain highly application-dependent, with failure modes shifting by target application rather than task alone.

2 Related Work

Mobile GUI agent benchmarks. Mobile GUI agents have motivated a growing set of datasets and benchmarks for screen understanding, action prediction, and closed-loop task execution. Offline resources such as Android-in-the-Wild (Rawles et al., 2023), AndroidControl (Li et al., 2024), AMEX (Chai et al., 2025a), Mobile-Bench (Deng et al., 2024), Mobile-Bench-v2 (Xu et al., 2025a), MobileVLM/Mobile3M (Wu et al., 2024a), MONDAY (Jang et al., 2025), ColorBench (Song et al., 2025), and MobiBench (Im et al., 2025) provide screenshots, UI states, demonstrations, or multi-path trajectory annotations. Online benchmarks such as AndroidWorld (Rawles et al., 2025), AndroidLab (Xu et al., 2025b), A3 (Chai et al., 2025b), SPA-Bench (Chen et al., 2025a), AndroidArena (Xing et al., 2024), and B-MoCA (Lee

et al., 2024) evaluate agents in live or emulated environments. CATBench is complementary to these efforts: rather than adding more task types or longer workflows, it holds the functional task fixed and varies the application within the same category.

Cross-app and transfer-oriented evaluation. Several benchmarks study transfer or multi-app behavior, but they vary a different experimental axis from CATBench. GUI-Odyssey (Lu et al., 2025) collects trajectories that may involve multiple applications, making it useful for studying cross-app composition. TransBench (Lu et al., 2025) studies transfer across version, platform, and application axes, but its cross-application setting is primarily framed around static GUI grounding. FedMABench (Wang et al., 2025) constructs heterogeneous mobile-agent datasets for federated training and shows that app identity can be an important source of heterogeneity. CATBench instead treats app variation as the controlled evaluation variable: the user instruction and intended final state are fixed, while the application interface changes.

Grounding and diagnostic evaluation. CATBench also connects to work on GUI grounding and diagnostic agent evaluation. Grounding benchmarks and models such as SeeClick-/ScreenSpot (Cheng et al., 2024), OS-Atlas (Wu et al., 2024b), UGround (Gou et al., 2025), Set-of-Mark prompting (Yang et al., 2023), and UI-Ins (Chen et al., 2025b) study how models map language or visual targets to UI elements. Evaluation work such as LlamaTouch (Zhang et al., 2024), GUIDE (Zhai et al., 2026), AgentRewardBench (Lù et al., 2025), and Auto-Eval Judge (Bhonsle et al., 2025) highlights the need to move beyond scalar success rates. CATBench combines these threads: it evaluates the same category-level task across interchangeable apps, and then uses trajectory-level diagnosis to separate planning, grounding, mixed, and other failures.

3 CATBench

CATBench is a category-controlled benchmark for measuring within-category cross-application generalization in mobile GUI agents. The benchmark spans 10 functional categories, 100 task templates, 52 runnable applications, and 520 task–application combinations on a live Android emulator. Each task template is instantiated across multiple structurally diverse applications that support the same user-facing goal, which enables controlled measurement of how agent performance varies when the task is held fixed and only the application changes.

CATBench is built on top of the AndroidWorld emulator infrastructure (Rawles et al., 2025), which provides screenshots, accessibility trees, persistent application state, and a touch-and-keystroke action interface. Section 3.1 formalizes the task setting and verifier annotation. Section 3.2 describes the bottom-up data construction pipeline. Section 3.3 introduces the diagnostic evaluation protocol. Section 3.4 reports the resulting benchmark statistics.

3.1 Task Design

We formulate each CATBench episode as a partially observed Markov decision process instantiated by a task template τ and an application ρ. The template τ specifies the user instruction, task parameters, initialization logic, target final state, and success verifier. At step t, the underlying Android system has state s_t, the agent receives observation o_t, and the agent executes action a_t. The context variable ρ specifies the application on which the task is instantiated. Each episode has a maximum horizon H.

Observation space. At each step, the harness exposes both a screenshot and a filtered accessibility tree. This dual representation accommodates vision-only, text-only, and multimodal agents under a unified evaluation protocol, and all observations derive from the same underlying emulator state.

Action space. The agent controls the device through six primitive actions, namely `tap(x, y)`, `long_press(x, y)`, `swipe(x1, y1, x2, y2)`, `type(text)`, `key(keycode)`, and `wait(ms)`. These primitives are sufficient to operate Android applications and match the action surface used by existing mobile GUI benchmarks (Rawles et al., 2025). Agent designers may compose these primitives into higher-level macros, but final execution always passes through the same low-level interface.

Success verification. Each task template τ is paired with a deterministic programmatic verifier $v_{\tau,\rho}$ for each supported application ρ. The verifier returns a binary success label after the episode terminates and is not exposed to the agent during execution. The verification methodology follows AndroidWorld (Rawles et al., 2025), which inspects

persistent application state through SQLite tables, content providers, on-disk files, and shared preferences whenever the target final state is preserved in persistent storage, and falls back to UI-state heuristics over the accessibility tree, the active screen, or transient runtime state when the outcome is display-bound or runtime-bound.

CATBench does not introduce a new verification mechanism. To make the source of every success label auditable, we annotate each verifier with one of three evidence types according to the source it consults:

- **Durable-state (D).** Reads persistent application or system state; Used for tasks whose outcome is preserved in app databases, content providers, on-disk files, or shared preferences, such as created notes, calendar events, alarms, transactions, and contact records.
- **UI-state (U).** Reads the final visible interface, accessibility tree, or transient runtime state; Used when the outcome is inherently display-bound or runtime-bound, such as displaying filtered search results, showing a route, or leaving a stopwatch running.
- **Hybrid (H).** Combines durable and UI-state evidence; Used when neither source alone is sufficient, for example when a task requires a persistent artifact to exist and the application to also reach a particular screen.

The evidence type is stored as metadata on each verifier implementation, and the per-template assignment is reported in Table D.12. This annotation does not change how success is decided. It records which AndroidWorld-style check was consulted, so that the source of every success label is auditable.

3.2 Bottom-up Data Construction

The CATBench task–application suite is constructed through a three-stage pipeline shown in Figure 2, combining automated tooling with human verification. Stage 1 specifies categories and task templates. Stage 2 expands each category with structurally diverse applications. Stage 3 enforces the validation gates that determine which task–application pairs enter the runnable suite.

Stage 1: Category and template specification. We define 10 functional categories corresponding to common mobile goals, namely Tasks, Notes, Finance, Music, Calendar, SMS, File Manager,

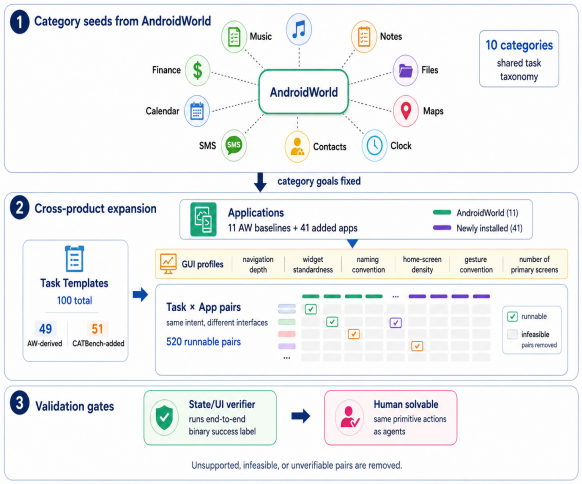

Figure 2: **Bottom-up construction of CATBench.** We seed CATBench from AndroidWorld-style functional categories, expand each category into task–application pairs, and retain only pairs that pass verifier, GUI-profile, and human-solvability validation gates.

Maps, Contacts, and Clock. Within each category, we specify 10 reusable task templates, yielding 100 templates in total. Each template is a parameterized natural-language instruction paired with deterministic initialization logic and a verifier of one of the three evidence types described in Section 3.1. To preserve compatibility with prior work, 49 templates inherit task intent from AndroidWorld (Rawles et al., 2025). The remaining 51 are newly introduced in CATBench to broaden coverage of operations such as recurring task creation, archive management, world clock manipulation, and route planning, so that the benchmark is not tied to the original AndroidWorld task distribution.

Stage 2: Cross-product application expansion. For each category, we combine AndroidWorld-original apps with 41 newly installed alternatives. The original apps serve as canonical baselines. The new apps are selected to expose meaningful interface variation rather than near-duplicate copies of the same interface. We first require that each app support the category's task templates using only the primitive actions defined in Section 3.1. We then profile each candidate app along six GUI properties: navigation depth, widget standardness, home-screen density, naming convention, gesture convention, and number of primary screens. These profiles are not exposed to the agent and are not used to determine task success. They are used to ensure interface diversity within each category and to support later analysis of app-level difficulty.

Stage 3: Validation gates. Each retained task–application pair must pass two validation gates. First, the task verifier must execute end-to-end and return a binary success label for any agent rollout. The same verifier is used for both execution conditions in Section 3.3, so execution and planner-augmented execution are compared under the same success criterion. Second, the pair must be solvable by a human operator using the same primitive action interface available to the agent. Unsupported, infeasible, or unverifiable pairs are removed from the runnable suite.

3.3 Diagnostic Evaluation Protocol

A central design goal of CATBench is to support diagnosis of failures rather than only counting them. The deterministic verifier of Section 3.1 produces a binary success label but cannot say whether a failure originated in task decomposition or in interface execution. To open this distinction up to measurement, every task–application instance in CATBench is executed under two complementary conditions.

C1: Direct execution. The agent receives only the user instruction and the current observation. It must infer the necessary sub-goals, ground the relevant UI elements, execute actions through the six-primitive action space, and decide when to stop. This condition measures end-to-end mobile GUI competence under the standard mobile-agent assumption that the agent is given a goal and an interface.

C2: Execution with generated sub-goals. A separate general-purpose frontier language model receives the same user instruction and produces an application-independent task breakdown that describes what the agent should accomplish step by step. The breakdown is prepended to the agent's prompt for the whole episode. The language model that produces the breakdown does not observe the screen, does not know the application-specific UI layout, and does not provide coordinates, accessibility-node identifiers, or button labels. The evaluated agent must still ground the breakdown in the live interface and execute the task through the same action space as in C1.

The contrast between C1 and C2 separates two distinct meanings of failure under an identical verifier. If a task fails under C1 but succeeds under C2, the original failure was at least partly caused by missing or incorrect task decomposition, which we call a *planning* failure. If a task fails under both conditions, the limiting factor is not the availability of a high-level plan, and the failure must originate at the level of UI grounding, action execution, or recovery from intermediate states, which we call *grounding* failure.

3.4 Benchmark Statistics

Verifier Coverage. Every runnable task–application pair has exactly one deterministic verifier, and every task template is annotated with one of the three AndroidWorld-style evidence types described in Section 3.1. The complete per-template assignment is reported in Table D.12. Because the annotation records which check was consulted, a reader can audit whether each reported success rate was derived from persistent state, final UI state, or both.

Task Difficulty Range. CATBench deliberately includes task–application pairs spanning a range of transfer difficulty. Some templates have nearly identical realizations across applications and serve as sanity checks for current agents. Others require navigating application-specific menus, custom widgets, nonstandard terminology, or multi-step state changes, and provide informative signal for future systems.

4 Experiments and Results

We conduct a comprehensive evaluation of CATBench to assess whether mobile GUI agents preserve task success when the application changes within the same functional category. CATBench evaluates 13 baselines across 10 categories, 52 applications, and 520 task–application combinations. Each instance is evaluated under two execution conditions. C1 is direct execution. C2 prepends an application-independent sub-goal breakdown to the agent prompt.

Our analysis is driven by four research questions:

- **RQ1.** How well does success on AndroidWorld-original applications transfer to newly installed same-category applications?
- **RQ2.** Is the cross-application gap driven only by AndroidWorld-inherited task templates?
- **RQ3.** Does app-independent sub-goal augmentation close the cross-application gap?

- **RQ4.** Which failure modes grow when the application implementation changes?

4.1 Setup

Models. We evaluate 13 baselines across five families (Table 1): *GUI-Trained* models post-trained for GUI control, *General* vision–language models, *Closed*-source proprietary models, *Hybrid* configurations pairing a frontier planner with the UI-Venus-72B grounder, and *Agent* systems wrapping a base model with planning, memory, and reflection.

Execution. All evaluations use the same emulator, six-primitive action space, and deterministic verifier. Each task–application instance is executed three times under each of C1 (direct execution) and C2 (planner-augmented execution, Section 3.3). Failed rollouts are categorized by a confidence-gated LLM-as-judge cascade into {PLANNING, GROUNDING, MIXED, OTHER}. The judge is used only for diagnosis. Success is always determined by the programmatic verifier. Architecture, prompts, decoding, and the 200-episode human-validation protocol are in Appendix C.

Metrics. For a category with AW-original application ρ_{orig} and newly installed set $\mathcal{R}_{\text{new}}$:

$$\begin{aligned}\text{Retention} &= \tfrac{1}{|\mathcal{R}_{\text{new}}|}\textstyle\sum_{\rho\in\mathcal{R}_{\text{new}}} \text{SR}_\rho \,/\, \text{SR}_{\rho_{\text{orig}}},\\ \Delta_{\text{app}} &= \tfrac{1}{|\mathcal{R}_{\text{new}}|}\textstyle\sum_{\rho\in\mathcal{R}_{\text{new}}} \text{SR}_\rho - \text{SR}_{\rho_{\text{orig}}},\\ \sigma_{\text{app}} &= \text{std}_{\rho\in\mathcal{R}_{\text{new}}}(\text{SR}_\rho)\,.\end{aligned}$$

Δ_{app} uses the convention new minus original throughout. Retention is undefined when $\text{SR}_{\rho_{\text{orig}}} = 0$ and is reported separately. For categories with multiple AW-original applications, the original-application success rates are averaged before the change is computed. Therefore, all app–task pairs would overestimate same-category generalization.

4.2 RQ1: Cross-Application Transfer

Table 1 shows a consistent cross-application transfer gap across model families. Averaging the two template-provenance partitions, AndroidWorld-original applications achieve **35.3%** success, while newly installed applications achieve **22.3%**, a signed change of −**13.1** percentage points and an aggregate retention of **63.2%**. The effect is not confined to one model class: GUI-trained models, general VLMs, closed-source systems, hybrid planner–grounder systems, and agent baselines all lose performance when the same task family is moved to unfamiliar apps.

The largest drops occur for strong models whose original-app performance is high. Gemini-3-Pro loses **32.5** points on AW-inherited templates, and the remaining SOTA baselines, including GPT-5.1, both hybrid configurations, and MAI-UI-8B, all lose between **20** and **28** points in the same partition. The pattern is even sharper on newly added CATBench templates, where the hybrid systems show the largest drops in the table: Gemini-3-Pro† falls by **40.4** points and GPT-5.1† by **25.7** points, substantially exceeding the drops of their closed-source counterparts (−19.7 and −4.2). This indicates that high AndroidWorld performance does not directly imply robust same-category transfer. Newly installed apps introduce different layouts, labels, control placements, and workflow conventions; models that succeed on the canonical app often fail when the semantic task remains fixed but the interface changes. The high app-level standard deviations in Table 1 (typically 20–35 points for the stronger systems) further show that transfer is uneven across target apps, not merely uniformly lower. Per-category and per-model breakdowns are in Appendix D.

4.3 RQ2: Performance on Original AndroidWorld Tasks vs. New Tasks

Splitting the benchmark by template provenance shows that the cross-app gap is not an artifact of the newly introduced CATBench tasks. On **AW-inherited templates**, models average **40.8%** success on AndroidWorld-original apps but only **25.1%** on newly installed apps, a −**15.7**-point drop. On **new CATBench templates**, models average **29.8%** on original apps and **19.4%** on newly installed apps, a smaller but still substantial −**10.4**-point drop.

This asymmetry suggests that AndroidWorld-style competence contains a strong task–interface familiarity component. The inherited templates benefit most from canonical app pairings: they start with a much higher original-app ceiling (**40.8%** vs. **29.8%**, an 11.0-point spread), but that advantage compresses to a **5.7**-point spread on new apps (**25.1%** vs. **19.4%**). Unfamiliar applications thus appear to push both task groups toward a similar cross-app floor, while AW-original pairings inflate the apparent ceiling. A benchmark containing only canonical app pairings would therefore overesti-

Table 1: **Signed cross-app generalization change, decomposed by template provenance (recomputed from the per-app appendix). (a)** AW-inherited templates from AndroidWorld (Rawles et al., 2025): same task, different app, isolating the cross-app effect. **(b)** Newly introduced CATBench templates. **AW app**: mean success on the AW-shaded baseline apps. **New**: mean success across newly installed apps in the same categories. **Std**: population std across new apps. **Δ**: New − AW app (negative = drop on new apps).

		(a) AW-inherited templates				(b) New CATBench templates			
Family	**Model**	**AW app**	**New**	**Std**	**Δ**	**AW app**	**New**	**Std**	**Δ**
GUI Trained	UI-Venus-Navi-72B	24.1	11.3	14.9	-12.8	13.6	13.3	23.0	-0.2
	UI-Venus-7B	37.7	21.4	21.7	-16.3	23.1	16.6	23.7	-6.5
	GUI-Owl-7B	20.3	17.7	26.3	-2.6	10.7	26.3	35.0	+15.6
	MAI-UI-8B	56.3	33.0	26.3	-23.4	37.9	25.7	31.2	-12.1
	UI Voyager-4B	42.6	30.6	25.7	-12.0	24.5	11.6	21.3	-12.9
General	Qwen3-VL-30B-A3B	13.5	4.6	7.8	-8.9	7.1	1.9	6.7	-5.2
	Qwen3-VL-8B	9.5	3.5	6.7	-6.0	11.2	3.3	7.7	-7.9
	Qwen2.5-VL-72B	19.0	13.2	16.7	-5.8	7.6	18.4	31.2	+10.8
Closed	Gemini-3-Pro	75.6	43.1	27.2	-32.5	61.0	41.2	32.8	-19.7
	GPT-5.1	61.5	41.2	25.6	-20.2	37.6	33.4	30.7	-4.2
Hybrid	Gemini-3-Pro†	66.4	43.1	25.9	-23.3	61.0	20.5	22.0	-40.4
	GPT-5.1†	66.4	39.9	28.8	-26.5	53.8	28.1	28.1	-25.7
Agt	Mobile-Agent-v3	38.0	23.6	26.2	-14.3	38.6	11.8	19.4	-26.8
	Average	**40.8**	**25.1**	**21.5**	**-15.7**	**29.8**	**19.4**	**24.1**	**-10.4**

Notes. Success rates in percent, Δ in percentage points. † *Hybrid*: closed-source planner + UI-Venus-Navi-72B grounder. Values recomputed from the per-app appendix tables, using equal weighting over apps (7 AW-shaded apps and 26 new apps across Tasks, Notes, Finance, Music, Calendar, SMS).

mate same-category generalization.

4.4 RQ3. Does App-Independent Sub-Goal Augmentation Close the Cross-App Gap?

We next ask whether explicit task decomposition helps agents transfer across apps. In C2, we prepend the original instruction with high-level sub-goals generated by Gemini-3-Pro. These sub-goals are app-independent. They describe what needs to be done, but do not include screen observations, coordinates, or app-specific UI hints. We run this comparison on the matched five-category subset used in C1, covering SMS, File Manager, Maps, Contacts, and Clock.

Table 2 shows that C2 gives a small average gain. Across all validated runs, success increases from **21.6%** to **22.1%**, a gain of **+0.5** points. The improvement is larger on newly installed apps, where success increases by **+1.3** points, while AndroidWorld-original apps slightly decrease by **-1.1** points. This suggests that app-independent sub-goals are most useful in the transfer setting, where the agent must apply the same intent to a less familiar interface.

The gains are clearer for several GUI-trained models. GUI-Owl-7B improves by **+4.8** points overall, UI-Venus-7B by **+3.0**, and MAI-UI-8B by **+1.7**. These gains are especially visible on newly installed apps, where UI-Venus-7B improves by **+5.2** points and MAI-UI-8B improves by **+2.6**. At the same time, the aggregate gain remains modest. This suggests that explicit sub-goals help some agents, but do not close the cross-app gap. Even with better task structure, agents still need to ground the next action in the current app interface. Full per-app deltas are reported in Appendix Table E.1.

4.5 RQ4: Application-Driven Shift in Failure Mode

Figure 3 reports the per-model distribution of failure modes under C1 after excluding environment and evaluator errors. Planning failures are the dominant mode for most agents, including GUI-Owl-7B (**81%**), Gemini-3-Pro-† (**66%**), GPT-5.1-† (**57%**), GPT-5.1 (**56%**), and the UI-Venus models (**51–53%**). These failures typically correspond to missing required sub-goals, stopping too early, or choosing an incorrect high-level strategy. Grounding failures are more concentrated in several models: Qwen3-VL-30B-A3B has the largest grounding share (**60%**), followed by Qwen3-VL-8B (**38%**) and Mobile-Agent-v3 (**34%**). This indicates that some models often know the broad task intent but

Table 2: **Effect of app-independent sub-goal augmentation.** C1 uses the original instruction. C2 prepends Gemini-3-Pro-generated app-independent sub-goals. The top rows report aggregate C1/C2 performance. The bottom rows show models with positive overall C2 gains.

Subset / model	All	AW	New
Aggregate success rate			
C1	21.6	27.6	18.6
C2	22.1	26.6	19.9
Δ	+0.5	-1.1	+1.3
Models with positive overall C2 gain			
UI-Venus-7B Δ	+3.0	-1.3	+5.2
GUI-Owl-7B Δ	+4.8	+7.9	+3.2
MAI-UI-8B Δ	+1.7	0.0	+2.6

Note. Values are percentages or percentage-point changes. AW denotes AndroidWorld-original apps. The matched subset contains 1,150 model–app–template trials. Full per-app deltas are reported in Appendix Table E.1.

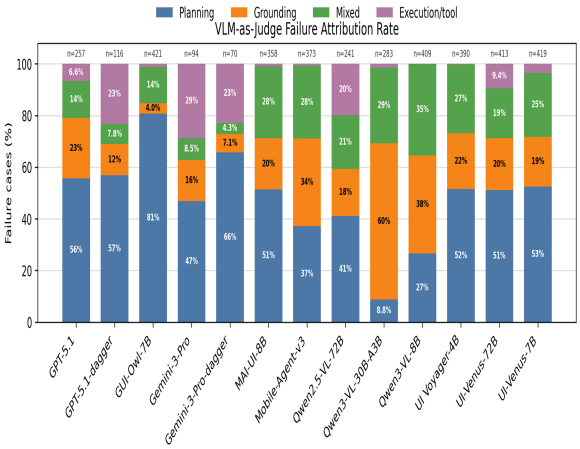

Figure 3: Per-category distribution of primary failure labels under direct execution.

fail to localize or operate the right UI element in the target app. Mixed failures are also common, especially for Qwen3-VL-8B (**35%**), Qwen3-VL-30B-A3B (**29%**), Mobile-Agent-v3 (**28%**), and MAI-UI-8B (**28%**), suggesting that planning and grounding errors frequently interact rather than appearing as clean separate categories. Overall, the results show that there is no single universal failure profile across mobile GUI agents. Some models are primarily planning-limited, others are grounding-limited, and many fail through a mixture of the two. This supports the need for CATBench's diagnostic evaluation: cross-app failure cannot be understood from final success rates alone, and improving deployment-ready mobile agents will require both better task decomposition and more robust interface grounding.

5 Conclusion

In this paper, we introduced CATBench, a category-controlled live Android benchmark that isolates within-category cross-application generalization in mobile GUI agents. By holding the user instruction and target final state fixed while varying only the application, CATBench exposes a variable that single-application benchmarks conflate with task difficulty. Across 13 baselines spanning five model families, performance is strongly application-dependent within a single functional category, success on a canonical benchmark application is not a reliable proxy for success on alternative applications supporting the same goal, externally generated sub-goals reduce planning failures but do not close the cross-application gap, and the dominant failure mode shifts between planning and grounding with the application rather than the task. These results indicate that progress on cross-application competence depends primarily on UI-level grounding and recovery, and that benchmark progress should be measured against application sets that can be replaced rather than fitted.

6 Limitations

CATBench isolates cross-application generalization by fixing the task goal while varying the app interface, but its scope remains bounded. The benchmark covers 10 Android categories, 52 apps, and 520 task–app combinations, which provides broad coverage but not an exhaustive sample of mobile applications, platforms, or real user contexts. Its deterministic verifiers make success labels auditable, yet they still depend on app-specific implementations and may miss ambiguous or partially correct outcomes. Finally, failure attribution relies on a human-validated LLM-based classifier, so the labels should be treated as diagnostic evidence rather than ground truth.

References

Roshita Bhonsle, Rishav Dutta, Sneha Vavilapalli, Harsh Seth, Abubakarr Jaye, Yapei Chang, Mukund Rungta, Emmanuel Aboah Boateng, Sadid Hasan, Ehi Nosakhare, and Soundar Srinivasan. 2025. Auto-eval judge: Towards a general agentic framework for task completion evaluation. arXiv preprint arXiv:2508.05508. *Preprint*, arXiv:2508.05508.

Yuxiang Chai, Siyuan Huang, Yazhe Niu, Han Xiao, Liang Liu, Guozhi Wang, Dingyu Zhang, Shuai Ren,

and Hongsheng Li. 2025a. AMEX: Android multi-annotation expo dataset for mobile GUI agents. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 2138–2156.

Yuxiang Chai, Shunye Tang, Han Xiao, Weifeng Lin, Hanhao Li, Jiayu Zhang, Liang Liu, Pengxiang Zhao, Guangyi Liu, Guozhi Wang, Shuai Ren, Rongduo Han, Haining Zhang, Siyuan Huang, and Hongsheng Li. 2025b. A3: Android agent arena for mobile GUI agents with essential-state procedural evaluation. arXiv preprint arXiv:2501.01149. *Preprint*, arXiv:2501.01149.

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

Liangyu Chen, Hanzhang Zhou, Chenglin Cai, Jianan Zhang, Panrong Tong, Quyu Kong, Xu Zhang, Chen Liu, Yuqi Liu, Wenxuan Wang, Yue Wang, Qin Jin, and Steven Hoi. 2025b. UI-Ins: Enhancing GUI grounding with multi-perspective instruction-as-reasoning. arXiv preprint arXiv:2510.20286. *Preprint*, arXiv:2510.20286.

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

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

Yichen Gong, Zhuohan Cai, Sunhao Dai, Yuqi Zhou, Zhangxuan Gu, Changhua Meng, and Shuheng Shen. 2026. VenusBench-Mobile: A challenging and user-centric benchmark for mobile GUI agents with capability diagnostics. arXiv preprint arXiv:2604.06182. *Preprint*, arXiv:2604.06182.

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

Youngmin Im, Byeongung Jo, Jaeyoung Wi, Seungwoo Baek, Tae Hoon Min, Joo Hyung Lee, Sangeun Oh, Insik Shin, and Sunjae Lee. 2025. MobiBench: Multi-branch, modular benchmark for mobile GUI agents. arXiv preprint arXiv:2512.12634. *Preprint*, arXiv:2512.12634.

Yunseok Jang, Yeda Song, Sungryull Sohn, Lajanugen Logeswaran, Tiange Luo, Dong-Ki Kim, Kyunghoon Bae, and Honglak Lee. 2025. Scalable video-to-dataset generation for cross-platform mobile agents. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8604–8614.

Quyu Kong, Xu Zhang, Zhenyu Yang, Nolan Gao, Chen Liu, Panrong Tong, Chenglin Cai, Hanzhang Zhou, Jianan Zhang, Liangyu Chen, Zhidan Liu, Steven Hoi, and Yue Wang. 2025. MobileWorld: Benchmarking autonomous mobile agents in agent-user interactive and MCP-augmented environments. arXiv preprint arXiv:2512.19432. *Preprint*, arXiv:2512.19432.

Juyong Lee, Taywon Min, Minyong An, Dongyoon Hahm, Haeone Lee, Changyeon Kim, and Kimin Lee. 2024. Benchmarking mobile device control agents across diverse configurations. arXiv preprint arXiv:2404.16660. *Preprint*, arXiv:2404.16660.

Kaixin Li, Ziyang Meng, Hongzhan Lin, Ziyang Luo, Yuchen Tian, Jing Ma, Zhiyong Huang, and Tat-Seng Chua. 2025. ScreenSpot-Pro: GUI grounding for professional high-resolution computer use. In *Proceedings of the 33rd ACM International Conference on Multimedia*, MM '25, pages 8778–8786, New York, NY, USA. Association for Computing Machinery.

Wei Li, William Bishop, Alice Li, Chris Rawles, Folawiyo Campbell-Ajala, Divya Tyamagundlu, and Oriana Riva. 2024. On the effects of data scale on UI control agents. arXiv preprint arXiv:2406.03679. *Preprint*, arXiv:2406.03679. NeurIPS 2024 Datasets and Benchmarks Track.

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

Xing Han Lù, Amirhossein Kazemnejad, Nicholas Meade, Arkil Patel, Dongchan Shin, Alejandra Zambrano, Karolina Stańczak, Peter Shaw, Christopher J. Pal, and Siva Reddy. 2025. AgentRewardBench: Evaluating automatic evaluations of web agent trajectories. arXiv preprint arXiv:2504.08942. *Preprint*, arXiv:2504.08942.

Yuheng Lu, Qian Yu, Hongru Wang, Zeming Liu, Wei Su, Yanping Liu, Yuhang Guo, Maocheng Liang, Yunhong Wang, and Haifeng Wang. 2025. TransBench: Breaking barriers for transferable graphical user interface agents in dynamic digital environments. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 12464–12478.

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

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

Yuanyi Song, Heyuan Huang, Qiqiang Lin, Yin Zhao, Xiangmou Qu, Jun Wang, Xingyu Lou, Weiwen Liu, Zhuosheng Zhang, Jun Wang, Yong Yu, Weinan Zhang, and Zhaoxiang Wang. 2025. ColorBench: Benchmarking mobile agents with graph-structured framework for complex long-horizon tasks. arXiv preprint arXiv:2510.14621. *Preprint*, arXiv:2510.14621.

WenHao Wang, Zijie Yu, Rui Ye, Jianqing Zhang, Guangyi Liu, Liang Liu, Siheng Chen, and Yanfeng Wang. 2025. FedMABench: Benchmarking mobile GUI agents on decentralized heterogeneous user data. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 26387–26408.

Qinzhuo Wu, Weikai Xu, Wei Liu, Tao Tan, Jianfeng Liu, Ang Li, Jian Luan, Bin Wang, and Shuo Shang. 2024a. MobileVLM: A vision-language model for better intra- and inter-UI understanding. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 10231–10251, Miami, Florida, USA. Association for Computational Linguistics.

Zhiyong Wu, Zhenyu Wu, Fangzhi Xu, Yian Wang, Qiushi Sun, Chengyou Jia, Kanzhi Cheng, Zichen Ding, Liheng Chen, Paul Pu Liang, and Yu Qiao. 2024b. OS-ATLAS: A foundation action model for generalist GUI agents. arXiv preprint arXiv:2410.23218. *Preprint*, arXiv:2410.23218.

Mingzhe Xing, Rongkai Zhang, Hui Xue, Qi Chen, Fan Yang, and Zhen Xiao. 2024. Understanding the weakness of large language model agents within a complex Android environment. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 6061–6072.

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

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

Jianwei Yang, Hao Zhang, Feng Li, Xueyan Zou, Chunyuan Li, and Jianfeng Gao. 2023. Set-of-mark prompting unleashes extraordinary visual grounding in GPT-4V. arXiv preprint arXiv:2310.11441. *Preprint*, arXiv:2310.11441.

Yuwen Zhai, Runze Li, Liang Wang, Nian Shi, Liwu Xu, Wei Zhang, Ran Lin, Bo Xu, and Benlei Cui. 2026. GUIDE: Interpretable GUI agent evaluation via hierarchical diagnosis. arXiv preprint arXiv:2604.04399. *Preprint*, arXiv:2604.04399.

Li Zhang, Shihe Wang, Xianqing Jia, Zhihan Zheng, Yunhe Yan, Longxi Gao, Yuanchun Li, and Mengwei Xu. 2024. LlamaTouch: A faithful and scalable testbed for mobile UI task automation. In *Proceedings of the 37th Annual ACM Symposium on User Interface Software and Technology*.

A Additional Related Work

Mobile GUI benchmarks. Mobile GUI benchmarks can be broadly divided into offline and online settings. Offline resources such as Android-in-the-Wild (Rawles et al., 2023), AndroidControl (Li et al., 2024), AMEX (Chai et al., 2025a), Mobile-Bench (Deng et al., 2024), Mobile-Bench-v2 (Xu et al., 2025a), MobileVLM/Mobile3M (Wu et al., 2024a), ColorBench (Song et al., 2025), MONDAY (Jang et al., 2025), and MobiBench (Im et al., 2025) provide screenshots, UI states, demonstrations, or multi-path trajectory annotations. These benchmarks are scalable and reproducible, but they primarily evaluate fixed states or pre-recorded trajectories. Online benchmarks such as AndroidWorld (Rawles et al., 2025), AndroidLab (Xu et al., 2025b), A3 (Chai et al., 2025b), SPA-Bench (Chen et al., 2025a), AndroidArena (Xing et al., 2024), B-MoCA (Lee et al., 2024), MobileWorld (Kong et al., 2025), and VenusBench-Mobile (Gong et al., 2026) evaluate agents in live or emulated Android environments. CATBench is complementary to both lines: rather than adding more task types or longer workflows, it holds the functional task fixed and varies the app within the same category.

Cross-app and transfer-oriented evaluation. Several works study transfer or multi-app behavior. GUI-Odyssey (Lu et al., 2025) collects trajectories that may move across multiple applications, making it useful for studying cross-app composition. TransBench (Lu et al., 2025) studies trans-

fer across version, platform, and application axes, but its cross-application setting is primarily framed around static GUI grounding. FedMABench (Wang et al., 2025) constructs heterogeneous mobile-agent datasets for federated training and shows that app identity can be an important source of heterogeneity. These works motivate CATBench, but their evaluation axes differ from ours. CATBench evaluates the same task template across interchangeable apps in the same functional category, so that performance variation can be attributed to app-specific interface differences rather than to different tasks or app-to-app workflows.

Grounding and diagnostic evaluation. Element grounding is central to GUI control: the agent must map a textual or visual intent to the correct actionable UI element. SeeClick/ScreenSpot (Cheng et al., 2024), OS-Atlas (Wu et al., 2024b), ScreenSpot-Pro (Li et al., 2025), UGround (Gou et al., 2025), Set-of-Mark prompting (Yang et al., 2023), and UI-Ins (Chen et al., 2025b) study GUI grounding across mobile, desktop, and web interfaces. CATBench connects to this line because its planner-augmented condition gives the acting agent an explicit task decomposition while still requiring it to locate and operate the relevant UI elements in each app. Thus, failures under C2 often expose grounding, interaction, or recovery limitations that are hidden by final success alone.

Evaluation methodology and LLM-based analysis. As agent benchmarks scale, evaluation design becomes increasingly important. AndroidWorld (Rawles et al., 2025) uses deterministic programmatic success checks, while A3 (Chai et al., 2025b), SPA-Bench (Chen et al., 2025a), and VenusBench-Mobile (Gong et al., 2026) incorporate more flexible automated evaluation for realistic mobile tasks. LlamaTouch (Zhang et al., 2024) evaluates mobile UI execution by matching essential UI states rather than exact action sequences. In broader agent evaluation, AgentRewardBench (Lù et al., 2025), GUIDE (Zhai et al., 2026), and Auto-Eval Judge (Bhonsle et al., 2025) study trajectory-level or judge-based diagnosis. CATBench follows this diagnostic direction, but applies it to within-category cross-app generalization: we compare unaided execution with planner-augmented execution and then use trajectory-level analysis to label planning, grounding, mixed, and other failures.

B Verifier Evidence Types

CATBench uses deterministic programmatic verifiers to decide task success. To make the source of each success label explicit, we annotate every task template with one of three verifier evidence types.

Durable-state checks. A durable-state verifier inspects persistent application or Android-system state after the episode terminates. Examples include app-specific SQLite tables, shared preferences, content-provider records, on-disk files, message stores, contact records, calendar events, alarms, notes, transactions, and playlist metadata. Durable-state checks are preferred whenever the target final state is represented as a persistent artifact.

UI-state checks. A UI-state verifier inspects the final visible interface, accessibility tree, active screen, or transient runtime state. This class is used when the task goal is inherently display- or runtime-dependent, such as showing a filtered list, displaying search results, viewing file or contact details, showing a route, playing a song, or leaving a stopwatch in a specified state.

Hybrid checks. A hybrid verifier combines durable-state and UI-state evidence. This class is used when durable state alone does not fully establish success, or when the task requires both a persistent artifact and an expected UI-level or external-intent state. Examples include attaching a receipt, importing or sharing content, exporting a map location, or recording a track.

Implementation consistency. The verifier evidence type is stored as metadata with each verifier implementation. During benchmark construction, we require every runnable task–application pair to have a verifier and every task template to have a non-null evidence type in $\{D, U, H\}$. The appendix table below is intended to be generated from this metadata, rather than hand-maintained, so that the paper and implementation cannot silently diverge.

C Failure-Mode Classification: Protocol and Validation

The success rate reported throughout this paper is computed entirely from the programmatic validators inherited from Android World (Rawles et al., 2025) or, for the CATBench-added app variants, from cross-app re-implementations that we verify preserve AW's success criterion on the AW-

baseline app, without any VLM as a judge or human annotator.

Separately and as a *diagnostic layer*, we classify each validator-failed episode into one of six failure modes, including PLANNING, GROUNDING, MIXED, EXECUTION_TOOLING, UNKNOWN and using a vision-language classifier. We then estimate the classifier's accuracy against human annotators who label the same field on the same sample, with the validator's verdict held fixed.

C.1 Classifier Architecture

Each *validator-failed* episode $e = (g, \tau, \mathcal{S}, y{=}0)$ with goal g, action–thought trace τ, screenshot sequence $\mathcal{S} = (s_1, \ldots, s_T)$, which was processed by a single VLM call. The classifier receives the goal, the validator's verdict (always 0 for cases routed to it, included as read-only context), a compact textual trace $\tilde{\tau}$, side artifacts, and up to $K = 6$ salience-selected key screenshots $\mathcal{S}_{\text{key}} \subseteq \mathcal{S}$ stitched in temporal order. The classifier returns a single JSON object with the failure mode, per-axis planning and grounding scores, a confidence label, and an evidence list that must cite at least one step or screenshot per non-trivial label.

There is no triage stage, no cascade, no relitigation of the verdict, and no second call.

C.2 Model and Decoding

Role	Model
Failure-mode classifier	Gemini 3 Pro

Table C.1: Single-stage classifier. Called via the `generateContent` REST endpoint of the Google AI Studio API.

Decoding settings. `temperature`$= 0$; provider-default `top_p` and `top_k`; no `max_output_tokens` cap; `responseMimeType="application/json"` so the model returns a single JSON object. Each call has a 120 s timeout and at most 3 retries with exponential backoff $(2, 4, 8)$ s. Images are passed inline as base64-encoded JPEGs at quality 80 and at most 896 px on the longest edge. At most $K = 6$ key screenshots are attached per call, selected by the salience-aware sampler in Section C.3. We never use sampling, chain-of-thought scratchpads, self-consistency, or majority voting; each classification is a single deterministic forward pass.

Cost telemetry. We persist the `promptTokenCount` and `candidatesTokenCount` fields of the `generateContent` response for every call. Median per-call spend is $\approx$ 11,000 tokens. Total cost for classifying all validator-failed episodes across the full $11{\times}10{\times}52$ matrix is under \$200.

C.3 Key-Frame Selection

The classifier sees a small number of screenshots rather than the full trajectory. The selector $\textsc{SmartKeyIndices}(e, K)$ always includes (i) the first and last step, then prioritises, in order: (ii) steps whose `error/exception_info` fields match the regex `traceback|error|exception|timeout|failed|cannot|impossible`, plus the step immediately preceding each such hit; (iii) the first index of any run of four or more consecutive identical actions, plus the step three before it; (iv) steps where the high-level action type (CLICK, TYPE, SWIPE, ...) changes from the previous step. If the selected set is still smaller than K we fill the remainder by uniform striding over the unselected indices. For an episode of T steps the resulting set has cardinality $\min(T, K)$ with $K = 6$ in all experiments.

C.4 Prompt

The prompt below is the literal system message sent to the API; the user turn contains the episode payload as JSON followed by the inline JPEG images. We do not modify the prompt between models, categories, or runs.

Failure-mode classifier system prompt

```
You are a CATBench failure-mode classifier.

You receive a CATBench episode whose
    programmatic validator already returned
is_successful = 0. Your job is NOT to second-
    guess that verdict. Your job is
to classify the dominant reason the agent
    failed.

You see: the goal, the recorded `
    is_successful` flag (always 0 for cases
routed to you), a compact step trace, side
    artifacts, and a STITCHED set of
key screenshots (first, last, error-adjacent,
     repeated-action, and any
transition steps).

Return one JSON object only, exactly:
{
```

```
  "primary_failure_mode": "planning|grounding
     |mixed_planning_grounding|
     execution_tooling|
     environment_or_evaluator|unknown",
  "planning_score": 0-3,
  "grounding_score": 0-3,
  "confidence": "low|medium|high",
  "rationale": "one concise paragraph",
  "evidence": ["short quoted/paraphrased
     evidence", "screenshot:step_4 shows ..."]
}

Rules:
- "planning" applies when the agent chose a
     wrong subgoal, missed a required
  step, stopped early, or committed to an
     infeasible strategy.
- "grounding" applies when the plan was right
      but the agent could not locate
  or interact with the correct UI element.
- "mixed_planning_grounding" applies when
     both meaningfully contributed.
- "execution_tooling" applies when the agent'
     s framework (parser, tool call,
  API timeout, malformed action) is the
     dominant failure source.
- "environment_or_evaluator" applies when an
     app crash, missing permission,
  network outage, or a likely validator false
     -negative is the dominant
  source. A true validator false-negative is
     reported separately by the
  validator audit in Sec. 6.7; flagging one
     here is a hypothesis, not a
  verdict.
- "unknown" when the evidence is genuinely
     insufficient.
- Scores: 0 absent / 1 possible / 2 likely /
     3 dominant for each axis.
- Cite at least one step number or screenshot
      index per non-trivial mode.
- Do NOT include markdown, prose, or extra
     fields.
```

The user message that follows the system prompt is constructed as "`Classify this CATBench episode.`" followed by a pretty-printed JSON object containing `goal`, `task_template`, `is_successful` (always 0 for cases routed here), `exception_info`, the compact `steps` array, `key_step_indices`, and `side_artifacts`. Each inline image is annotated with a trailing text part of the form "`^ above image is step` i `(field=name).`" to bind the image to the step index.

C.5 Human Annotation Protocol

To estimate the classifier's accuracy we draw a stratified sample of validator-failed episodes from the full CATBench matrix and have two authors annotate each one with the same six-class failure-mode taxonomy the classifier uses. **The annotators do not relabel the validator's verdict.** The validator's verdict is shown as read-only context; the human contributes only the failure-mode field, the same field the classifier produces, on the same sample. This keeps the VLM and the human in identical roles and makes the reported F_1 a clean measure of the classifier's ability to reproduce a human's failure-mode judgement.

Sample size and stratification. We sample $N = 200$ validator-failed episodes total, drawn from the union of all $M{=}11$ agents and $C{=}10$ categories. Each agent receives a floor of $\lceil N/M \rceil$ episodes ($\geq$ 8 per agent). Within each agent's quota we stratify across (category, prior classifier failure mode) so that every mode has nonzero support for per-class F_1 estimation. Low-confidence classifier cases receive a $3\times$ within-bucket weight relative to high-confidence cases because those are the most informative for the comparison (Table C.2). A separate randomly chosen 10% subset ($n = 20$) is double-annotated by both authors for the inter-rater agreement statistic.

Axis	Allocation
Agents covered	11
Min. episodes per agent	8
Source pool	validator-failed episodes
Confidence weight (low/med/high)	$3.0/1.5/1.0$
Double-annotated subset	10%
Total annotations	200 (primary) + 20 (double)

Table C.2: Stratified-sampling configuration used to draw the validation set for failure-mode labelling.

Annotator training. Two of the authors annotated the sample independently after a calibration round of 20 pilot episodes drawn outside the validation set. We resolved label disagreements by reference to (i) the written failure-mode definitions and (ii) the screenshots, and froze the protocol before any validation-set episode was labelled.

Annotation interface. For every sampled episode the tool renders an HTML report with the goal, the compact text trace, the validator's verdict shown as a read-only banner, and the same salience-selected key frames the classifier sees (saved as `step_NNN_field.jpg` and embedded inline). The report is opened in the system browser. The annotator then answers, on the command line, only:

1. `primary_failure_mode` $\in$ the same six modes the classifier uses (or `skip`);
2. `planning_score` $\in \{0, 1, 2, 3\}$;
3. `grounding_score` $\in \{0, 1, 2, 3\}$;
4. `confidence` $\in$ {LOW, MEDIUM, HIGH};
5. `rationale`: one free-text line referencing a step or screenshot;
6. OPTIONAL: a single yes/no flag indicating the screenshots appear to *contradict* the validator's verdict. This flag is written to a separate `audit_flags.jsonl` file and contributes only to the one-shot validator audit in Section C.6.

The schema mirrors the classifier's output schema so the two can be compared label-by-label. Median annotation time was 1 min 48 s per episode.

Inter-rater agreement. We report raw agreement and Cohen's κ on the multi-class failure-mode label across the double-annotated subset. Disagreements are reconciled by discussion; the reconciled labels are used only for the agreement statistic, never as training signal for the classifier.

Comparison metrics. For each classifier configuration we report, on the same fixed annotated set: (i) per-class precision, recall, and F_1 for the six failure modes; (ii) macro-F_1 averaged over classes with non-zero human support; (iii) the full confusion matrix; (iv) total token spend and wall-clock time per episode. The same annotated set is re-used for every classifier configuration so absolute differences in F_1 are not confounded by sample drift. We deliberately do not report a "binary verdict F_1" because the verdict is owned by the validator and is not in scope for the classifier.

C.6 Validator Audit (separate, one-shot)

A reviewer could reasonably ask whether the programmatic validator itself ever disagrees with how a human would read the screenshots — i.e. whether a validator-zero episode actually *looks* successful, or vice versa. This is a property of the validator, not of the classifier, so we measure it in a separate audit pass that is not part of the per-episode pipeline.

The audit samples 50 validator-successes and 50 validator-failures stratified across categories, asks a human to verify whether the screenshots agree with the validator's verdict, and reports the per-direction disagreement rate. The audit is reported once in the appendix; the SR numbers in the main table remain those produced by the validator. The annotation tool's "`suspect_validator`" flag (Sec. C.5) contributes to this audit when the failure-mode-annotation pool intersects the audit pool.

D Per-Category, Per-App Success Rates

In this appendix we report success rates broken down by individual app, for each of the ten functional categories in CATBench. The structure is identical across all ten tables. Each app has two sub-columns: **AW** reports the success rate on AW-inherited templates evaluated on that app, and **New** reports the success rate on the newly added CATBench templates. This decomposition follows the same template-provenance split used in Table 1 of the main paper, applied here at the per-app granularity. The leftmost shaded sub-columns belong to the AndroidWorld-original baseline app(s) for each category; the remaining sub-columns belong to the newly installed apps. The bottom row of each table reports the column-wise mean across model rows with completed results, exposing per-app difficulty independent of model choice. Aggregate summaries (New Avg., Std., Δ) are reported in the main paper and are not duplicated here.

Table D.1: **App-level success rates for Tasks / To-Do.** Each app has two sub-columns: AW-inherited templates (6) and newly added CATBench templates (4).

		Tasks.org		**Cfait**		**Todo List (PFA)**		**ntodotxt**		**TaskMate**		**Grit**	
Family	**Model**	**AW**	**New**	**AW**	**New**	**AW**	**New**	**AW**	**New**	**AW**	**New**	**AW**	**New**
GUI Trained	UI-Venus-Navi-72B	0/6	1/4	1/6	1/4	0/6	1/4	0/6	0/4	0/6	0/4	0/6	0/4
	UI-Venus-7B	3/6	2/4	0/6	0/4	0/6	1/4	1/6	0/4	2/6	2/4	0/6	1/4
	GUI-Owl-7B	2/6	2/4	0/6	0/4	0/6	1/4	0/6	0/4	2/6	2/4	0/6	1/4
	MAI-UI-8B	5/6	4/4	3/6	0/4	0/6	1/4	0/6	1/4	2/6	1/4	4/6	2/4
	UI Voyager-4B	3/6	0/4	1/6	1/4	2/6	1/4	4/6	0/4	3/6	0/4	4/6	1/4
General	Qwen3-VL-30B-A3B	1/6	0/4	0/6	0/4	0/6	0/4	1/6	0/4	0/6	0/4	0/6	0/4
	Qwen3-VL-8B	0/6	0/4	0/6	0/4	0/6	0/4	1/6	0/4	0/6	0/4	0/6	1/4
	Qwen2.5-VL-72B	0/6	0/4	0/6	0/4	0/6	1/4	0/6	0/4	0/6	0/4	0/6	0/4
Closed	Gemini-3-Pro	3/6	3/4	3/6	2/4	0/6	2/4	1/6	1/4	3/6	0/4	1/6	3/4
	GPT-5.1	0/6	1/4	1/6	0/4	3/6	2/4	1/6	1/4	1/6	1/4	2/6	2/4
Hybrid	Gemini-3-Pro†	0/6	3/4	2/6	1/4	1/6	2/4	2/6	1/4	1/6	0/4	1/6	3/4
	GPT-5.1†	0/6	1/4	5/6	2/4	0/6	2/4	4/6	1/4	0/6	0/4	1/6	3/4
Agent	Mobile-Agent-v3	0/6	1/4	0/6	0/4	0/6	0/4	0/6	0/4	0/6	0/4	1/6	1/4
Mean success rate		21.8%	34.6%	20.5%	13.5%	7.7%	26.9%	19.2%	9.6%	17.9%	11.5%	17.9%	34.6%

Cells show successful tasks out of scheduled templates. **AW**: AW-inherited templates. **New**: newly added CATBench templates. † Planner–executor setup: the closed-source model plans and UI-Venus-72B grounds. The bottom row reports the column-wise mean success rate over rows with completed results.

Table D.2: **App-level success rates for Notes.** Each app has two sub-columns: AW-inherited templates (9) and newly added CATBench templates (1).

		Joplin		Markor		NotallyX		neutriNote CE		Notesnook		Orgzly Revived	
Family	**Model**	**AW**	**New**	**AW**	**New**	**AW**	**New**	**AW**	**New**	**AW**	**New**	**AW**	**New**
GUI Trained	UI-Venus-Navi-72B	3/9	0/1	0/9	0/1	3/9	0/1	1/9	1/1	2/9	0/1	2/9	0/1
	UI-Venus-7B	4/9	0/1	1/9	0/1	5/9	0/1	4/9	0/1	6/9	1/1	6/9	0/1
	GUI-Owl-7B	4/9	0/1	0/9	0/1	8/9	1/1	6/9	1/1	4/9	1/1	6/9	1/1
	MAI-UI-8B	7/9	1/1	5/9	0/1	7/9	1/1	1/9	1/1	7/9	0/1	–	–
	UI Voyager-4B	2/9	0/1	0/9	0/1	8/9	0/1	7/9	0/1	3/9	0/1	5/9	1/1
General	Qwen3-VL-30B-A3B	1/9	0/1	0/9	0/1	2/9	0/1	1/9	0/1	0/9	0/1	0/9	0/1
	Qwen3-VL-8B	0/9	0/1	0/9	0/1	1/9	0/1	0/9	0/1	1/9	0/1	0/9	0/1
	Qwen2.5-VL-72B	4/9	0/1	0/9	0/1	4/9	1/1	2/9	1/1	3/9	0/1	4/9	1/1
Closed	Gemini-3-Pro	8/9	1/1	6/9	1/1	8/9	1/1	2/9	1/1	6/9	1/1	6/9	1/1
	GPT-5.1	8/9	1/1	6/9	0/1	6/9	1/1	4/9	1/1	5/9	1/1	6/9	0/1
Hybrid	Gemini-3-Pro†	8/9	1/1	6/9	1/1	5/9	0/1	2/9	0/1	7/9	0/1	7/9	0/1
	GPT-5.1†	8/9	1/1	6/9	1/1	7/9	1/1	5/9	0/1	5/9	0/1	3/9	0/1
Agent	Mobile-Agent-v3	3/9	1/1	0/9	0/1	6/9	0/1	5/9	0/1	3/9	0/1	3/9	0/1
Mean success rate		51.3%	46.2%	25.6%	23.1%	59.8%	46.2%	34.2%	46.2%	44.4%	30.8%	44.4%	33.3%

Cells show successful tasks out of scheduled templates. **AW**: AW-inherited templates. **New**: newly added CATBench templates. † Planner–executor setup: the closed-source model plans and UI-Venus-72B grounds. The bottom row reports the column-wise mean success rate over rows with completed results.

Table D.3: **App-level success rates for Finance.** Each app has two sub-columns: AW-inherited templates (5) and newly added CATBench templates (5).

		Pro Expense		Oinkoin		OpenMoneyBox		My Expenses		Finance Manager		Sushi	
Family	**Model**	**AW**	**New**	**AW**	**New**	**AW**	**New**	**AW**	**New**	**AW**	**New**	**AW**	**New**
GUI Trained	UI-Venus-Navi-72B	1/5	0/5	1/5	2/5	0/5	0/5	1/5	0/5	0/5	1/5	0/5	0/5
	UI-Venus-7B	1/5	0/5	1/5	1/5	1/5	0/5	1/5	3/5	0/5	1/5	2/5	1/5
	GUI-Owl-7B	0/5	0/5	1/5	2/5	1/5	1/5	0/5	1/5	1/5	2/5	1/5	2/5
	MAI-UI-8B	4/5	2/5	3/5	2/5	2/5	1/5	2/5	1/5	3/5	0/5	3/5	2/5
	UI Voyager-4B	5/5	3/5	0/5	1/5	2/5	0/5	2/5	0/5	2/5	1/5	1/5	0/5
General	Qwen3-VL-30B-A3B	0/5	0/5	0/5	0/5	0/5	0/5	0/5	0/5	0/5	0/5	0/5	0/5
	Qwen3-VL-8B	0/5	1/5	1/5	1/5	0/5	0/5	0/5	0/5	0/5	0/5	0/5	1/5
	Qwen2.5-VL-72B	1/5	1/5	1/5	1/5	0/5	0/5	1/5	1/5	1/5	1/5	0/5	1/5
Closed	Gemini-3-Pro	4/5	1/5	1/5	0/5	1/5	1/5	4/5	2/5	2/5	3/5	2/5	2/5
	GPT-5.1	5/5	2/5	3/5	1/5	3/5	0/5	4/5	1/5	3/5	1/5	3/5	0/5
Hybrid	Gemini-3-Pro†	3/5	1/5	1/5	0/5	0/5	1/5	4/5	2/5	3/5	1/5	1/5	2/5
	GPT-5.1†	3/5	1/5	2/5	2/5	0/5	1/5	2/5	2/5	3/5	2/5	3/5	2/5
Agent	Mobile-Agent-v3	3/5	1/5	2/5	1/5	1/5	0/5	1/5	1/5	2/5	3/5	2/5	1/5
Mean success rate		46.2%	20.0%	26.2%	21.5%	16.9%	7.7%	33.8%	21.5%	30.8%	24.6%	27.7%	21.5%

Cells show successful tasks out of scheduled templates. **AW**: AW-inherited templates. **New**: newly added CATBench templates. † Planner–executor setup: the closed-source model plans and UI-Venus-72B grounds. The bottom row reports the column-wise mean success rate over rows with completed results.

Table D.4: **App-level success rates for Music.** Each app has two sub-columns: AW-inherited templates (5) and newly added CATBench templates (5).

Family	Model	Retro Music		Fossify Music Player		Apollo		SicMu Neo		Phonograph Plus		MonsterMusic	
		AW	**New**	**AW**	**New**	**AW**	**New**	**AW**	**New**	**AW**	**New**	**AW**	**New**
GUI Trained	UI-Venus-Navi-72B	1/5	1/5	0/5	1/5	0/5	0/5	0/5	0/5	0/5	0/5	1/5	0/5
	UI-Venus-7B	0/5	1/5	0/5	1/5	0/5	0/5	0/5	0/5	0/5	0/5	1/5	0/5
	GUI-Owl-7B	0/5	0/5	0/5	0/5	0/5	0/5	0/5	0/5	0/5	0/5	0/5	0/5
	MAI-UI-8B	0/5	0/5	1/5	0/5	0/5	0/5	0/5	1/5	0/5	0/5	1/5	1/5
	UI Voyager-4B	0/5	1/5	0/5	0/5	0/5	0/5	0/5	1/5	0/5	0/5	0/5	0/5
General	Qwen3-VL-30B-A3B	0/5	0/5	1/5	0/5	0/5	0/5	0/5	0/5	0/5	0/5	0/5	0/5
	Qwen3-VL-8B	0/5	0/5	0/5	0/5	0/5	0/5	0/5	0/5	0/5	0/5	0/5	1/5
	Qwen2.5-VL-72B	2/5	0/5	0/5	1/5	0/5	1/5	0/5	0/5	0/5	0/5	2/5	0/5
Closed	Gemini-3-Pro	3/5	2/5	3/5	3/5	0/5	1/5	0/5	2/5	3/5	2/5	4/5	3/5
	GPT-5.1	1/5	2/5	2/5	2/5	0/5	1/5	0/5	2/5	2/5	2/5	1/5	3/5
Hybrid	Gemini-3-Pro†	4/5	2/5	4/5	3/5	1/5	1/5	0/5	1/5	3/5	1/5	4/5	3/5
	GPT-5.1†	4/5	2/5	4/5	3/5	1/5	1/5	0/5	1/5	3/5	0/5	3/5	3/5
Agent	Mobile-Agent-v3	1/5	0/5	0/5	1/5	0/5	0/5	0/5	0/5	0/5	0/5	1/5	0/5
Mean success rate		24.6%	16.9%	23.1%	23.1%	3.1%	7.7%	0.0%	12.3%	16.9%	7.7%	27.7%	21.5%

Cells show successful tasks out of scheduled templates. **AW**: AW-inherited templates. **New**: newly added CATBench templates. † Planner–executor setup: the closed-source model plans and UI-Venus-72B grounds. The bottom row reports the column-wise mean success rate over rows with completed results.

Table D.5: **App-level success rates for Calendar.** Each app has two sub-columns: AW-inherited templates (7) and newly added CATBench templates (3).

Family	Model	Simple Calendar Pro		Etar		Fossify Calendar		Calendar		KashCal	
		AW	**New**	**AW**	**New**	**AW**	**New**	**AW**	**New**	**AW**	**New**
GUI Trained	UI-Venus-Navi-72B	2/7	0/3	0/7	0/3	1/7	1/3	0/7	0/3	1/7	1/3
	UI-Venus-7B	5/7	2/3	1/7	1/3	3/7	1/3	0/7	0/3	2/7	0/3
	GUI-Owl-7B	1/7	0/3	0/7	0/3	1/7	0/3	0/7	0/3	0/7	0/3
	MAI-UI-8B	1/7	0/3	1/7	0/3	5/7	3/3	2/7	0/3	3/7	1/3
	UI Voyager-4B	3/7	2/3	1/7	0/3	2/7	1/3	2/7	0/3	2/7	1/3
General	Qwen3-VL-30B-A3B	0/7	0/3	0/7	0/3	0/7	0/3	0/7	0/3	0/7	0/3
	Qwen3-VL-8B	0/7	1/3	0/7	0/3	0/7	0/3	0/7	0/3	0/7	0/3
	Qwen2.5-VL-72B	2/7	1/3	2/7	0/3	3/7	0/3	0/7	0/3	2/7	1/3
Closed	Gemini-3-Pro	7/7	2/3	4/7	0/3	6/7	1/3	3/7	0/3	4/7	1/3
	GPT-5.1	5/7	1/3	1/7	1/3	6/7	2/3	3/7	0/3	3/7	1/3
Hybrid	Gemini-3-Pro†	6/7	2/3	4/7	0/3	4/7	1/3	3/7	0/3	3/7	0/3
	GPT-5.1†	6/7	2/3	2/7	0/3	5/7	2/3	1/7	0/3	1/7	0/3
Agent	Mobile-Agent-v3	6/7	3/3	0/7	0/3	7/7	2/3	0/7	0/3	2/7	0/3
Mean success rate		48.4%	41.0%	17.6%	5.1%	47.3%	35.9%	15.4%	0.0%	25.3%	15.4%

Cells show successful tasks out of scheduled templates. **AW**: AW-inherited templates. **New**: newly added CATBench templates. † Planner–executor setup: the closed-source model plans and UI-Venus-72B grounds. The bottom row reports the column-wise mean success rate over rows with completed results.

Table D.6: **App-level success rates for SMS.** Each app has two sub-columns: AW-inherited templates (6) and newly added CATBench templates (4).

Family	Model	Simple SMS Messenger		Fossify Messages		QUIK SMS		Messages	
		AW	**New**	**AW**	**New**	**AW**	**New**	**AW**	**New**
GUI Trained	UI-Venus-72B	4/6	2/4	4/6	2/4	1/6	0/4	1/6	0/4
	UI-Venus-7B	4/6	1/4	3/6	1/4	1/6	0/4	0/6	0/4
	GUI-Owl-7B	3/6	1/4	4/6	1/4	0/6	0/4	0/6	0/4
	MAI-UI-8B	5/6	1/4	0/6	1/4	2/6	0/4	1/6	0/4
	UI Voyager-4B	5/6	1/4	3/6	0/4	1/6	0/4	0/6	0/4
General	Qwen3-VL-30B-A3B	4/6	2/4	1/6	1/4	1/6	1/4	1/6	0/4
	Qwen3-VL-8B	4/6	1/4	1/6	0/4	0/6	0/4	1/6	0/4
	Qwen2.5-VL-72B	0/6	0/4	0/6	0/4	0/6	0/4	0/6	0/4
Closed	Gemini-3-Pro	5/6	1/4	4/6	1/4	1/6	0/4	1/6	0/4
	GPT-5.1	5/6	1/4	5/6	1/4	0/6	0/4	1/6	0/4
Hybrid	Gemini-3-Pro†	5/6	1/4	5/6	1/4	2/6	0/4	2/6	0/4
	GPT-5.1†	5/6	1/4	5/6	1/4	0/6	0/4	1/6	0/4
Agent	Mobile-Agent-v3	4/6	1/4	4/6	2/4	0/6	1/4	2/6	0/4
Mean success rate		67.9%	26.9%	50.0%	23.1%	11.5%	3.8%	14.1%	0.0%

Cells show successful tasks out of scheduled templates. **AW**: AW-inherited templates. **New**: newly added CATBench templates. † Planner–executor setup: the closed-source model plans and UI-Venus-72B grounds. The bottom row reports the column-wise mean success rate over rows with completed results.

Table D.7: **App-level success rates for File Manager.** Each app has two sub-columns: AW-inherited templates (3) and newly added CATBench templates (7).

		Material Files		Amaze File Manager		Fossify File Manager		Total Commander		X-plore File Manager	
Family	**Model**	**AW**	**New**	**AW**	**New**	**AW**	**New**	**AW**	**New**	**AW**	**New**
GUI Trained	UI-Venus-72B	1/3	3/7	2/3	2/7	1/3	4/7	0/3	1/7	1/3	1/7
	UI-Venus-7B	1/3	3/7	1/3	2/7	1/3	1/7	0/3	1/7	0/3	0/7
	GUI-Owl-7B	1/3	3/7	1/3	3/7	0/3	3/7	0/3	0/7	0/3	2/7
	MAI-UI-8B	2/3	3/7	2/3	3/7	0/3	1/7	2/3	5/7	1/3	4/7
	UI Voyager-4B	1/3	2/7	1/3	1/7	2/3	1/7	0/3	2/7	1/3	1/7
General	Qwen3-VL-30B-A3B	2/3	3/7	0/3	1/7	0/3	1/7	0/3	0/7	0/3	0/7
	Qwen3-VL-8B	3/3	5/7	1/3	3/7	0/3	0/7	0/3	1/7	0/3	3/7
	Qwen2.5-VL-72B	0/3	0/7	0/3	0/7	0/3	0/7	0/3	0/7	0/3	0/7
Closed	Gemini-3-Pro	3/3	5/7	3/3	5/7	3/3	7/7	3/3	7/7	2/3	7/7
	GPT-5.1	1/3	3/7	0/3	2/7	3/3	6/7	1/3	3/7	0/3	0/7
Hybrid	Gemini-3-Pro†	3/3	5/7	3/3	7/7	3/3	7/7	3/3	6/7	0/3	0/7
	GPT-5.1†	3/3	4/7	3/3	3/7	3/3	5/7	1/3	4/7	1/3	2/7
Agent	Mobile-Agent-v3	0/3	3/7	2/3	3/7	3/3	4/7	1/3	3/7	1/3	3/7
Mean success rate		53.8%	46.2%	48.7%	38.5%	48.7%	44.0%	28.2%	36.3%	17.9%	25.3%

Cells show successful tasks out of scheduled templates. **AW**: AW-inherited templates. **New**: newly added CATBench templates. † Planner–executor setup: the closed-source model plans and UI-Venus-72B grounds. The bottom row reports the column-wise mean success rate over rows with completed results.

Table D.8: **App-level success rates for Maps.** Each app has two sub-columns: AW-inherited templates (3) and newly added CATBench templates (7).

Family	Model	OsmAnd~		Organic Maps		CoMaps	
		AW	**New**	**AW**	**New**	**AW**	**New**
GUI Trained	UI-Venus-72B	0/3	3/7	1/3	0/7	1/3	1/7
	UI-Venus-7B	1/3	1/7	0/3	1/7	0/3	1/7
	GUI-Owl-7B	1/3	1/7	0/3	0/7	1/3	0/7
	MAI-UI-8B	1/3	0/7	0/3	1/7	1/3	0/7
	UI Voyager-4B	2/3	3/7	0/3	0/7	0/3	0/7
General	Qwen3-VL-30B-A3B	1/3	2/7	1/3	1/7	2/3	1/7
	Qwen3-VL-8B	2/3	2/7	1/3	3/7	1/3	2/7
	Qwen2.5-VL-72B	0/3	0/7	0/3	0/7	0/3	0/7
Closed	Gemini-3-Pro	1/3	2/7	0/3	0/7	1/3	1/7
	GPT-5.1	1/3	4/7	0/3	0/7	1/3	1/7
Hybrid	Gemini-3-Pro†	1/3	2/7	0/3	0/7	2/3	0/7
	GPT-5.1†	1/3	3/7	0/3	1/7	1/3	1/7
Agent	Mobile-Agent-v3	2/3	1/7	0/3	0/7	0/3	0/7
Mean success rate		35.9%	26.4%	7.7%	7.7%	28.2%	8.8%

Cells show successful tasks out of scheduled templates. **AW**: AW-inherited templates. **New**: newly added CATBench templates. † Planner–executor setup: the closed-source model plans and UI-Venus-72B grounds. The bottom row reports the column-wise mean success rate over rows with completed results.

Table D.9: **App-level success rates for Contacts.** Each app has two sub-columns: AW-inherited templates (2) and newly added CATBench templates (8).

Family	Model	Google Contacts		Fossify Contacts		Connect You		Simple Contacts Pro SE		Right Contact	
		AW	New	AW	New	AW	New	AW	New	AW	New
GUI Trained	UI-Venus-72B	2/2	2/8	1/2	0/8	0/2	0/8	0/2	2/8	0/2	2/8
	UI-Venus-7B	0/2	1/8	1/2	0/8	0/2	0/8	0/2	0/8	0/2	0/8
	GUI-Owl-7B	2/2	2/8	0/2	1/8	0/2	1/8	1/2	0/8	0/2	1/8
	MAI-UI-8B	1/2	0/8	0/2	0/8	0/2	0/8	1/2	0/8	0/2	0/8
	UI Voyager-4B	2/2	0/8	1/2	0/8	0/2	0/8	1/2	1/8	0/2	0/8
General	Qwen3-VL-30B-A3B	0/2	0/8	0/2	0/8	0/2	0/8	0/2	0/8	0/2	0/8
	Qwen3-VL-8B	2/2	1/8	0/2	0/8	0/2	2/8	0/2	2/8	0/2	0/8
	Qwen2.5-VL-72B	0/2	0/8	0/2	0/8	0/2	0/8	0/2	0/8	0/2	0/8
Closed	Gemini-3-Pro	1/2	1/8	1/2	3/8	0/2	2/8	1/2	2/8	1/2	3/8
	GPT-5.1	2/2	2/8	0/2	2/8	0/2	3/8	1/2	0/8	1/2	1/8
Hybrid	Gemini-3-Pro†	2/2	2/8	1/2	4/8	0/2	3/8	1/2	2/8	1/2	3/8
	GPT-5.1†	2/2	2/8	1/2	0/8	0/2	3/8	1/2	2/8	0/2	2/8
Agent	Mobile-Agent-v3	2/2	2/8	1/2	0/8	0/2	3/8	0/2	1/8	0/2	0/8
Mean success rate		69.2%	14.4%	26.9%	9.6%	0.0%	16.3%	26.9%	11.5%	11.5%	11.5%

Cells show successful tasks out of scheduled templates. **AW**: AW-inherited templates. **New**: newly added CATBench templates. † Planner–executor setup: the closed-source model plans and UI-Venus-72B grounds. The bottom row reports the column-wise mean success rate over rows with completed results.

Table D.10: **App-level success rates for Clock.** Each app has two sub-columns: AW-inherited templates (3) and newly added CATBench templates (7).

Family	Model	Google Clock		Clock		Simple Clock		Clock You		Chrono		Fossify Clock	
		AW	New	AW	New	AW	New	AW	New	AW	New	AW	New
GUI Trained	UI-Venus-72B	0/3	0/7	0/3	0/7	0/3	2/7	1/3	0/7	1/3	2/7	0/3	4/7
	UI-Venus-7B	0/3	1/7	0/3	0/7	0/3	1/7	0/3	0/7	0/3	0/7	0/3	3/7
	GUI-Owl-7B	2/3	5/7	0/3	1/7	0/3	1/7	0/3	1/7	0/3	1/7	0/3	4/7
	MAI-UI-8B	3/3	4/7	0/3	2/7	0/3	2/7	0/3	0/7	0/3	2/7	0/3	5/7
	UI Voyager-4B	3/3	3/7	1/3	0/7	0/3	1/7	1/3	1/7	0/3	2/7	0/3	2/7
General	Qwen3-VL-30B-A3B	0/3	0/7	0/3	0/7	0/3	0/7	0/3	0/7	0/3	0/7	0/3	0/7
	Qwen3-VL-8B	2/3	3/7	2/3	4/7	0/3	3/7	1/3	1/7	1/3	1/7	0/3	4/7
	Qwen2.5-VL-72B	0/3	0/7	0/3	0/7	0/3	0/7	0/3	0/7	0/3	0/7	0/3	0/7
Closed	Gemini-3-Pro	1/3	0/7	1/3	1/7	0/3	3/7	1/3	0/7	0/3	1/7	0/3	4/7
	GPT-5.1	2/3	4/7	0/3	0/7	0/3	2/7	0/3	1/7	1/3	1/7	0/3	4/7
Hybrid	Gemini-3-Pro†	3/3	4/7	1/3	3/7	0/3	3/7	0/3	0/7	1/3	2/7	1/3	5/7
	GPT-5.1†	0/3	0/7	0/3	4/7	0/3	2/7	0/3	0/7	0/3	1/7	0/3	1/7
Agent	Mobile-Agent-v3	3/3	3/7	0/3	3/7	0/3	3/7	1/3	0/7	1/3	1/7	0/3	4/7
Mean success rate		48.7%	29.7%	12.8%	19.8%	0.0%	25.3%	12.8%	4.4%	12.8%	15.4%	2.6%	44.0%

Cells show successful tasks out of scheduled templates. **AW**: AW-inherited templates. **New**: newly added CATBench templates. † Planner–executor setup: the closed-source model plans and UI-Venus-72B grounds. The bottom row reports the column-wise mean success rate over rows with completed results.

Table D.11: **CATBench overview.** CATBench evaluates cross-app generalization within 10 functional categories. Each category contains 10 task templates: AndroidWorld-derived task intents first, followed by newly added CATBench templates. Templates marked with ^{AW} preserve AndroidWorld task intent, even when renamed to category-level template names. Bold app names denote AndroidWorld-original app baselines. Total: 100 task templates, 52 apps, and 520 nominal task–app combinations.

Category	Task Templates	AW Orig. App(s)	Newly Installed Apps	# Apps AW+New
Tasks / To-Do	**CompletedTasksForDate**^{AW}, **DueNextWeek**^{AW}, **DueOnDate**^{AW}, **HighPriorityTasks**^{AW}, **HighPriorityDueOnDate**^{AW}, **IncompleteTasksOnDate**^{AW}, DueWithTime, Recurring, EditTask, CompleteTask	**Tasks.org**	Cfait, Todo List (PFA), ntodotxt, TaskMate, Grit	**1+5**
Notes	**CreateNote**^{AW}, **EditNote**^{AW}, **MergeNotes**^{AW}, **DeleteNote**^{AW}, **SearchNote**^{AW}, **ShareOrImportNote**^{AW}, **FolderOrMoveNote**^{AW}, **AttachOrTranscribeContent**^{AW}, **CountTodoItems**^{AW}, CreateChecklist	**Joplin**, **Markor**	NotallyX, neutriNote CE, Notesnook, Orgzly Revived	**2+4**
Finance	**AddExpense**^{AW}, **AddMultipleExpenses**^{AW}, **DeleteTransaction**^{AW}, **DeleteDuplicateTransactions**^{AW}, **AttachReceipt**^{AW}, AddIncome, EditTransaction, CategorySummary, DateRangeTotal, TransferBetweenWallets	**Pro Expense**	Oinkoin, OpenMoneyBox, My Expenses, Finance Manager, Sushi	**1+5**
Music	**CreatePlaylist**^{AW}, **AddToPlaylist**^{AW}, **AddToQueue**^{AW}, **SaveOrExportPlaylist**^{AW}, **PlaylistDuration**^{AW}, RenamePlaylist, RemoveFromPlaylist, ReorderQueue, SleepTimer, SearchAndPlay	**Retro Music**	Fossify Music Player, Apollo, SicMu Neo, Phonograph Plus, MonsterMusic	**1+5**
Calendar	**AddOneEvent**^{AW}, **AddTimedEvent**^{AW}, **AddRepeatingEvent**^{AW}, **DeleteEvent**^{AW}, **EventsOnDate**^{AW}, **NextEvent**^{AW}, **EventsInRange**^{AW}, EditEvent, AddReminder, MoveEvent	**Simple Calendar Pro**	Etar, Fossify Calendar, Calendar, KashCal	**1+4**
SMS	**Send**^{AW}, **Reply**^{AW}, **ReplyMostRecent**^{AW}, **Resend**^{AW}, **SendClipboard**^{AW}, **SendReceivedAddress**^{AW}, CreateDraftMessage, EditDraftMessage, DeleteConversation, ArchiveConversation	**Simple SMS Messenger**	Fossify Messages, QUIK SMS, Messages	**1+3**
File Manager	**DeleteFile**^{AW}, **MoveFile**^{AW}, **SaveCopyOfFile**^{AW}, CreateFolder, RenameFile, CopyFile, SearchFile, CompressFiles, ExtractArchive, ViewFileInfo	**Material Files**	Amaze File Manager, Fossify File Manager, Total Commander, X-plore File Manager	**1+4**
Maps	**AddFavorite**^{AW}, **AddMarker**^{AW}, **RecordTrack**^{AW}, SearchPlace, ViewPlaceInfo, RemoveFavorite, DeleteMarker, GetDirections, SearchNearbyPlace, MapsExportLocation	**OsmAnd~**	Organic Maps, CoMaps	**1+2**
Contacts	**AddContact**^{AW}, **NewContactDraft**^{AW}, EditContact, SearchContact, ViewContactDetails, AddFavoriteContact, RemoveFavoriteContact, DeleteContact, CallContact, MessageContact	**Google Contacts**	Fossify Contacts, Connect You, Simple Contacts Pro SE, Right Contact	**1+4**
Clock	**CreateTimer**^{AW}, **StopwatchRunning**^{AW}, **PauseStopwatch**^{AW}, CreateAlarm, EditAlarm, EnableAlarm, DeleteAlarm, StartTimer, StopwatchReset, AddWorldClock	**Clock**	Simple Clock, Google Clock, Clock You, Chrono, Fossify Clock	**1+5**
Total	**49 AW-derived templates + 51 CATBench-added templates**	**11 AW apps**	**41 newly installed apps**	**11+41**

Templates marked with ^{AW} preserve AndroidWorld task intent, not necessarily the exact original AndroidWorld class name. Unmarked templates are newly added CATBench templates used to extend each category to 10 task templates.
 # Apps AW+New reports AndroidWorld-original apps plus newly installed apps; the number after "+" is the newly installed app count.

Table D.12: **Verifier evidence type by task template.** D: durable-state verifier. U: UI-state verifier. H: hybrid verifier combining durable and UI-state evidence. Templates marked with ^{AW} preserve AndroidWorld task intent.

Category	**Task templates and verifier evidence type**
Tasks / To-Do	CompletedTasksForDate^{AW} (U); DueNextWeek^{AW} (U); DueOnDate^{AW} (U); HighPriorityTasks^{AW} (U); HighPriorityDueOnDate^{AW} (U); IncompleteTasksOnDate^{AW} (U); DueWithTime (D); Recurring (D); EditTask (D); CompleteTask (D)
Notes	CreateNote^{AW} (D); EditNote^{AW} (D); MergeNotes^{AW} (D); DeleteNote^{AW} (D); SearchNote^{AW} (U); ShareOrImportNote^{AW} (H); FolderOrMoveNote^{AW} (D); AttachOrTranscribeContent^{AW} (H); CountTodoItems^{AW} (U); CreateChecklist (D)
Finance	AddExpense^{AW} (D); AddMultipleExpenses^{AW} (D); DeleteTransaction^{AW} (D); DeleteDuplicateTransactions^{AW} (D); AttachReceipt^{AW} (H); AddIncome (D); EditTransaction (D); CategorySummary (U); DateRangeTotal (U); TransferBetweenWallets (D)
Music	CreatePlaylist^{AW} (D); AddToPlaylist^{AW} (D); AddToQueue^{AW} (U); SaveOrExportPlaylist^{AW} (H); PlaylistDuration^{AW} (U); RenamePlaylist (D); RemoveFromPlaylist (D); ReorderQueue (U); SleepTimer (U); SearchAndPlay (U)
Calendar	AddOneEvent^{AW} (D); AddTimedEvent^{AW} (D); AddRepeatingEvent^{AW} (D); DeleteEvent^{AW} (D); EventsOnDate^{AW} (U); NextEvent^{AW} (U); EventsInRange^{AW} (U); EditEvent (D); AddReminder (D); MoveEvent (D)
SMS	Send^{AW} (D); Reply^{AW} (D); ReplyMostRecent^{AW} (D); Resend^{AW} (D); SendClipboard^{AW} (D); SendReceivedAddress^{AW} (D); CreateDraftMessage (D); EditDraftMessage (D); DeleteConversation (D); ArchiveConversation (D)
File Manager	DeleteFile^{AW} (D); MoveFile^{AW} (D); SaveCopyOfFile^{AW} (D); CreateFolder (D); RenameFile (D); CopyFile (D); SearchFile (U); CompressFiles (D); ExtractArchive (D); ViewFileInfo (U)
Maps	AddFavorite^{AW} (D); AddMarker^{AW} (D); RecordTrack^{AW} (H); SearchPlace (U); ViewPlaceInfo (U); RemoveFavorite (D); DeleteMarker (D); GetDirections (U); SearchNearbyPlace (U); MapsExportLocation (H)
Contacts	AddContact^{AW} (D); NewContactDraft^{AW} (U); EditContact (D); SearchContact (U); ViewContactDetails (U); AddFavoriteContact (D); RemoveFavoriteContact (D); DeleteContact (D); CallContact (U); MessageContact (U)
Clock	CreateTimer^{AW} (U); StopwatchRunning^{AW} (U); PauseStopwatch^{AW} (U); CreateAlarm (D); EditAlarm (D); EnableAlarm (D); DeleteAlarm (D); StartTimer (U); StopwatchReset (U); AddWorldClock (D)

D = durable-state check; U = UI-state check; H = hybrid check. The concrete implementation may use app-specific database paths, content-provider APIs, file paths, accessibility-tree queries, or screenshot-derived UI checks, but the evidence type above records the semantic source of the success label at the task-template level.

If an individual app requires an app-specific fallback whose evidence type differs from the template default, the implementation should expose that override and the appendix should report it in an additional app-level exception table.

E Additional RQ3 Results

Table E.1: **Per-app effect of C2 sub-goal augmentation.** Each cell reports C2 minus C1 success in percentage points for the same model, app, and task templates. Positive values mean that app-independent sub-goals improve success. Google Maps and Maps.me are excluded because they are not part of the final CATBench app set.

Category	App	UI-Venus-7B	GUI-Owl-7B	MAI-UI-8B	UI Voyager-4B	Qwen3-VL-8B
SMS	Simple SMS Messenger	0	+30	+20	+10	+10
SMS	Fossify Messages	0	0	0	0	-10
SMS	QUIK SMS	0	+20	+10	+10	+10
SMS	Messages	+10	0	0	0	-10
File Manager	Material Files	0	0	+20	0	-30
File Manager	Amaze File Manager	+10	+30	+10	0	+10
File Manager	Fossify File Manager	-10	+20	-10	+10	+10
File Manager	Total Commander	-10	0	-20	-20	0
File Manager	X-plore File Manager	0	0	-10	0	-20
Maps	OsmAnd~	+20	+20	+20	-30	-20
Maps	Organic Maps	-10	0	-10	0	-40
Maps	CoMaps	-10	-10	-10	0	-30
Contacts	Google Contacts	+20	-10	0	+10	0
Contacts	Fossify Contacts	-10	-10	0	0	+10
Contacts	Connect You	0	+10	0	+10	-10
Contacts	Simple Contacts Pro SE	+20	+10	0	0	-20
Contacts	Right Contact	0	0	0	+10	0
Clock	Google Clock	+20	0	0	+10	+20
Clock	Clock	0	-10	+20	-10	-10
Clock	Simple Clock	+20	+10	+10	-10	0
Clock	Clock You	0	0	+10	-10	-10
Clock	Chrono	+20	+10	+10	0	0
Clock	Fossify Clock	-10	-10	0	-10	0

Note. Values are percentage-point changes from C1 to C2 on matched app-template cells. Each app contributes 10 task-template trials per model, so one additional successful task corresponds to a +10 point change.

F Disclosure of AI Assistance

In accordance with the ACL Policy on Publication Ethics regarding the use of generative AI, we disclose the AI tools used in the preparation of this work.

Writing assistance. We used ChatGPT (OpenAI) as a language aid to improve the grammar, fluency, and readability of text originally drafted by the authors. It was not used to generate scientific content, formulate research ideas, or produce claims, analyses, or conclusions.

Coding assistance. We used AI coding assistants such as ChatGPT, OpenAI Codex, and Claude Code (Anthropic) to support implementation tasks such as boilerplate generation, code refactoring, debugging, writing unit tests, and producing visualization scripts. The overall experimental design, model architecture, algorithmic choices, and evaluation protocols were conceived by the authors. All AI-generated or AI-assisted code was reviewed, tested, and verified by the authors before being incorporated into the experiments reported in this paper.

Responsibility. The authors take full responsibility for the entire content of this paper, including any text or code produced with AI assistance. AI tools were not listed as authors and did not contribute to research ideation, literature review, data analysis, interpretation of results, or scientific argumentation.