LIVEGUI-BENCH: A BEHAVIORAL BENCHMARK FOR AGENT-AS-GUI WORKFLOW SURFACES

Anonymous authors
Paper under double-blind review

ABSTRACT

Recurring personal work is stateful: a user should be able to return tomorrow to visible, editable task state an assistant created today, yet chat answers and static generated interfaces cannot support this, and no existing benchmark evaluates whether an agent can generate the surface through which that future user–agent work will occur. We introduce LIVEGUI-BENCH, a behavioral benchmark for *agent-native apps* — interactive applications whose interface, state, and behavior are generated and maintained by an agent — evaluated as Agent-as-GUI workflow surfaces that must actually drive the real services they name, grounded in the executable application APIs of AppWorld. Given an end-user intention and context, a system must generate a surface that satisfies five executable contracts across an open-submit-revise-reopen episode sequence: visible task state, action routing into an app-scoped agent workflow, real backend-API invocation, grounded write-back of returned entities into user-visible state, and persistence across reopening. LIVEGUI-BENCH contains 191 cases over eleven AppWorld app domains, with 764 episodes and 2,172 deterministic assertions checked against execution traces, API-call records, and application-database diffs — every score is read from executed behavior. The primary metric, Full Workflow Resolution (FWR), credits a case only when every contract holds in every episode, and a scripted control suite quantifies how assertion-level scores overstate workflow competence (up to an Assertion Pass Rate (APR) of 0.82 with FWR $= 0$). Frontier agentic systems resolve 5–15$\%$ of public workflows (`gpt-5.4` 0.05, `claude-opus-4.7` 0.15): the failure localizes sharply — routing (0.92–0.99) and persistence (0.72–0.75) are largely solved, but only 0.16–0.23 of required API invocations actually happen: real-API grounding is the dominant bottleneck.

1 INTRODUCTION

On Monday, a user asks an assistant to plan outfits for a rainy week. On Tuesday, the forecast changes and the user revises the plan. On Wednesday, they reopen the plan before leaving home. Each of today's dominant interaction forms fails this three-day workflow in a different way. A chat answer is one-shot: Tuesday's revision requires restating Monday's context. A generated mockup looks like a planner, but its buttons dispatch to nothing. Hidden agent memory may retain the facts, yet gives the user no visible object to inspect, edit, or trust. A GUI agent can operate an existing weather app, but it cannot create the missing workspace itself. What the user needs is an *agent-native app*: an interactive application whose interface, state, and behavior are generated and maintained by an agent (Figure 1). Its evaluable form is the *Agent-as-GUI workflow surface*: a generated interface the agent maintains as the user's visible, operable, recoverable task state.

Evaluating such a surface requires a shift in what benchmarks measure. The evaluated artifact is not a patch that makes tests pass (Jimenez et al., 2024), a rebuilt program that satisfies behavioral probes (Yang et al., 2026), an action trace inside existing software (Zhou et al., 2024; Xie et al., 2024; Trivedi et al., 2024), a website graded test-by-test by a navigation agent (Lu et al., 2025), or a final work product graded once (Mialon et al., 2023; Sun et al., 2026). It is a closed interaction loop: a generated surface, controls that route into an app-scoped agent workflow, agent output that becomes visible state, and persistence that lets the user reopen and continue. The serialized package a generator emits is only the carrier; correctness is defined behaviorally, by executing the surface through reserved episodes. This stance follows ProgramBench, which treats implementation style as

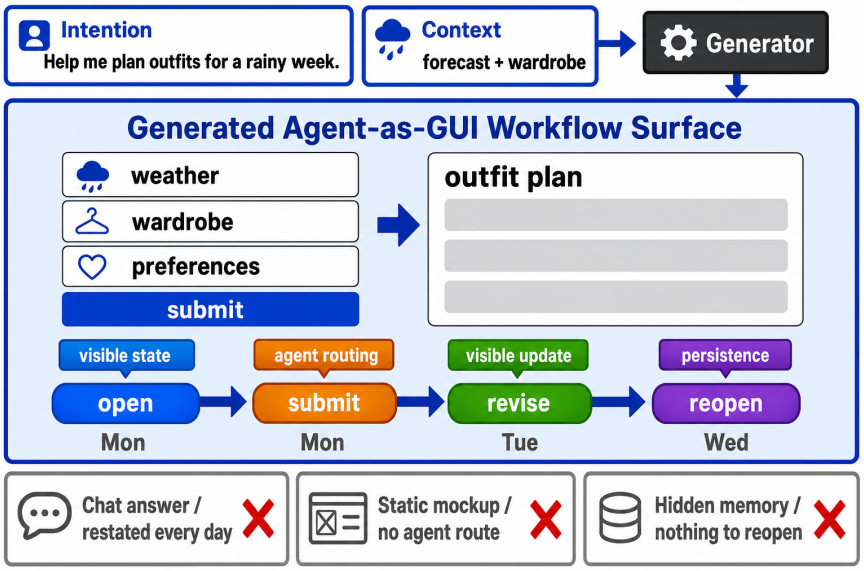

Figure 1: LIVEGUI-BENCH overview. Given an end-user intention and context, a generator must produce an Agent-as-GUI workflow surface — a generated interface that an agent maintains as the user's visible, operable, recoverable task state — and the surface must survive a reserved open-submit-revise-reopen episode sequence under layer checks for visible state, agent routing, visible update, and persistence. Each existing interaction form fails a different contract (bottom row): a chat answer must be restated every day, a static mockup routes no control into an agent, and hidden memory leaves the user nothing to reopen.

secondary to behavior under reserved tests (Yang et al., 2026), and AppWorld, which checks database state rather than agent narration (Trivedi et al., 2024).

We introduce LIVEGUI-BENCH, a behavioral benchmark that operationalizes this loop with five executable contracts. *Visible task state*: opening the surface must render task-relevant state and controls. *Action routing*: submitting or revising through the surface must dispatch into the app-scoped agent workflow it declares, not a dead handler. *Real-API grounding*: the workflow must invoke the named backend API and mutate the corresponding application database — naming an API is not calling it. *Grounded visible update*: the concrete returned entity (a playlist title, an amount, a file path) must be written back into user-visible state. *Reopen persistence*: after closing and reopening, that entity must still be visible and editable. Together the contracts make the agent-native definition executable: interface is checked by visible task state; behavior by action routing and real-API grounding; state by grounded visible update and reopen persistence. A case counts as resolved only if all contracts hold across all four episodes; we deliberately award no partial credit at the case level, following the full-completion convention of recent agent exams (Sun et al., 2026).

Our contributions are:

- **Task.** We formalize Agent-as-GUI workflow surface generation — *intention* + *context* → *surface* — as five executable contracts checked across an open-submit-revise-reopen episode sequence (Section 2).
- **Benchmark.** We construct LIVEGUI-BENCH: 191 recurring-workflow cases grounded in eleven AppWorld app domains, with 764 episodes and 2,172 deterministic assertions read from execution artifacts (Section 3).
- **Harness.** We implement an execution harness that drives generated surfaces only through user-level episodes, verifies real backend effects through database snapshots, and never repairs a broken package (Section 4).
- **Measurement.** Frontier agentic systems resolve only 5–15% of workflows even though routing and persistence are largely solved — the dominant bottleneck is invoking the real APIs the surface

Contract	Episodes	Checked behavior	Typical failure
Visible task state	open	task state and controls render	blank shell, transcript answer
Action routing	submit, revise	events dispatch into app-scoped agent workflow	static controls, dead handlers
Real-API grounding	submit, revise	the named AppWorld API is invoked and the application database changes	API named but never called
Grounded visible update	submit, revise	the concrete returned entity becomes user-visible state	hidden backend memory, generic echo
Reopen persistence	reopen	the entity remains visible and editable after teardown	state reset, route loss

Table 1: The five executable contracts. A case is resolved only if every contract holds in every episode that tests it.

names (0.16–0.23). A no-grounding ablation isolates this contract, and a deterministic control suite quantifies how assertion-level scores overstate competence (Section 5).

2 THE LIVEGUI-BENCH TASK

Task definition. Given an end-user intention x and optional context c, a system must produce an Agent-as-GUI workflow surface:

$$\texttt{generate}(x, c) \rightarrow \textit{surface} \mid \textit{failure}.$$

The surface is serialized as a package containing a manifest, UI description, routes, an orchestrator, a state schema, persistence bindings, and assets. The evaluator never grades this serialization directly. Instead, the evaluator renders the surface and executes a reserved episode sequence against it; the package is merely the carrier of behavior.

Episodes. Every case runs four episodes in order, mirroring how a real user returns to a workspace over days: *open* the surface for the first time; *submit* initial task information through its controls; *revise* constraints or continue the workflow; and *reopen* the surface after teardown. Episodes are executed through user-level actions only — fill, click, observe — never through privileged APIs into the package.

Contracts and assertions. Table 1 maps the five contracts to the episodes that test them and the assertion types that encode them. Each case carries on average 11 deterministic assertions whose expected values are concrete case entities (e.g., "the visible result names the *Monday Commute* playlist"), so a generator must ground the workflow in real objects rather than echo memorized phrasings.

Metrics. **Full Workflow Resolution** (FWR) is the primary metric: the fraction of cases whose assertions all pass across all four episodes. **Assertion Pass Rate** (APR) is reported only as a diagnostic, because a package can pass many layers while never touching a real API — the analogue of a program that passes unit tests but cannot be run against production. We additionally report per-contract pass rates (routing, API invocation, database change, grounded visibility, persistence) so failures localize to a contract. Verifying resulting environment state through database diffs follows AppWorld's evaluation philosophy (Trivedi et al., 2024; Yao et al., 2024); LIVEGUI-BENCH applies it to surfaces the agent itself generated.

3 THE LIVEGUI-BENCH DATASET

LIVEGUI-BENCH targets recurring personal workflows in which a persistent visible surface is more appropriate than a one-off answer, and grounds every case in the executable application APIs of AppWorld (Trivedi et al., 2024). The tracked case file is `benchmarks/intentions/livegui-bench.json`; each case exposes only intention and

Statistic	Value
Cases	191
AppWorld app domains	11
Episodes	764
Deterministic assertions	2172
Dev / public / sealed	37 / 75 / 79

Table 2: LIVEGUI-BENCH statistics — case, app-domain, episode, and deterministic-assertion counts with the dev/public/sealed split sizes — generated from the tracked case file `benchmarks/intentions/livegui-bench.json`.

context to the generator, while the grounding metadata (target APIs) and assertions remain evaluator-side.

App domains. The 191 cases span eleven app domains: nine single-app domains over the AppWorld services (`spotify`, `amazon`, `gmail`, `phone`, `venmo`, `splitwise`, `todoist`, `simple_note`, `file_system`) and two cross-app domains (`amazon+venmo` shopping-with-payments, `gmail+todoist` inbox-to-tasks). Every intention is a first-person request for a recurring, stateful agent-native app — a weekly commute-playlist builder, a shared-expense settle-up tracker, a download-folder sorter — whose workflow necessarily drives named AppWorld APIs (Table 2; the per-domain distribution is in Appendix A).

Scale and splits. Each case carries the four open-submit-revise-reopen episodes and on average 11 deterministic assertions — 2,172 in total, of which 376 assert a named API invocation, 268 assert an application-database change, and 382 assert a concrete entity in visible state. Splits are stratified within every app domain: 37 development, 75 public-test, and 79 sealed-test cases. The sealed split is withheld for hosted evaluation, following the contamination-control practice of live and sealed benchmarks (Jain et al., 2024; Sun et al., 2026).

Example case. In the weekly commute-playlist case, the intention asks for an app that builds a fresh commute playlist each week and remembers last week's choices. The open episode checks that input controls and an executable route exist. The submit episode ("vibe upbeat indie, seed artists Phoenix… for *Monday Commute*") asserts that the event enters the orchestrator, that `create_playlist` is invoked, that the `spotify` database changes, and that the visible result names *Monday Commute*. The revise episode adds "no songs over 4 minutes" and asserts a further database change and an updated visible result. The reopen episode asserts *Monday Commute* is still visible after teardown.

Construction controls. Intentions were authored by a multi-agent pipeline (one author per app domain, grounded in the live AppWorld capability catalog) and adversarially reviewed for recurring-workflow shape and concrete seed entities. Assertions were authored by a second pipeline and adversarially repaired so that every grounding value is a concrete entity from the case context — never a generic word — and every named API exists in the domain's catalog. Case ids, split counts, episode shape, and assertion types are validated mechanically. Domain labels are never exposed to the generator.

4 EVALUATION HARNESS

Adapter boundary. All systems are evaluated through a single adapter boundary: the harness supplies intention and context, and receives a workflow-surface package or an explicit failure. Any generator can participate — an end-to-end agent, a code-first pipeline behind an adapter, or a scripted control. During evaluation the harness never repairs missing routes, missing state, invalid assets, or unrenderable surfaces; a broken package fails exactly as it would for a user.

Deterministic episode scoring. The benchmark runner executes each case's episodes against the generated surface and checks every assertion against observable execution artifacts. Routing is

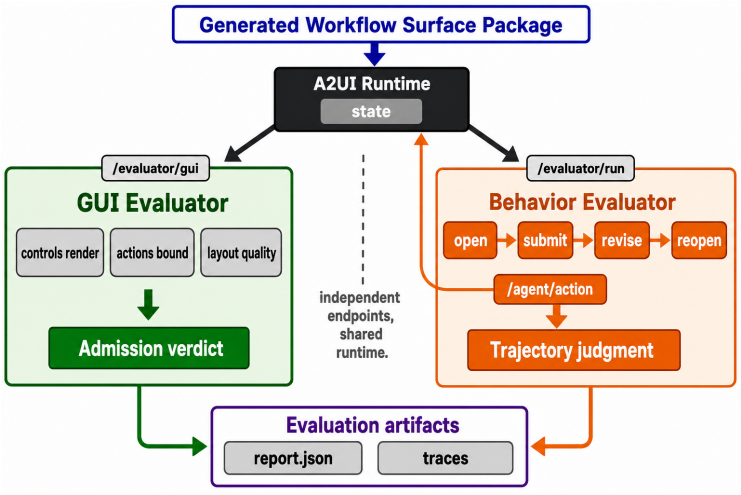

Figure 2: Evaluator architecture. The GUI evaluator (`/evaluator/gui`) performs admission-quality checks on the package and its rendered surface; the behavior evaluator (`/evaluator/run`) drives online behavioral episodes one user-level action at a time, ending in a trajectory judgment that cross-checks the deterministic scoring. The two are independent endpoints over the same generated surface, sharing one runtime and one action path; both write artifact-backed reports and neither mutates the package.

read from the runtime's agent trace; API invocation from the per-action tool-call record; database change from AppWorld database snapshots diffed before and after each call; grounded visibility from the surface's user-visible result state; persistence from a fresh state query after teardown. Because AppWorld apps are real executable services with real databases, "the workflow created the *Monday Commute* playlist" is a checkable fact, not a rubric judgment.

Two evaluators, one action path. Beyond the scoring runner, the harness exposes two evaluator endpoints over the same generated surface (Figure 2). The *GUI evaluator* (`/evaluator/gui`) checks admission quality with deterministic rules over the package and its rendered A2UI model: semantic input slots, visible result blocks, action bindings, readable text, and cross-platform readiness; blocking findings prevent admission. The *behavior evaluator* (`/evaluator/run`) drives long-horizon exploratory sessions online, one user-level fill/click/observe action at a time through the runtime action path. Each session ends in an LLM judgment over the full trajectory, and the evaluator archives step-by-step snapshots and rendered frames (Appendix D). Both write artifact-backed reports; neither mutates the package. All reported scores come from the deterministic runner; the behavior evaluator reaches its own verdict from the user's side of the screen. On a stratified 15-case public probe the two agreed on 14 of 15 cases. The single disagreement was a case whose deterministic run had flaked on an orchestrator parse error; the freshly generated behavior session passed that case, so the behavior evaluator caught a spurious deterministic failure rather than contradicting a real one. This agreement indicates the deterministic assertions capture the same workflow failures a human-style online tester would observe.

Orchestration loop. Figure 3 shows how a single loop coordinates the two evaluation stages. A surface passes GUI checks before save and executes reserved behavioral episodes after render; every finding is attributed to a specific layer and handed back to the generator as an optimization signal. The harness itself never edits the surface — it does not patch renderers or shortcut actions — so the benchmark distinguishes "could not render a plausible surface" from "rendered a surface that does not behave," a distinction the layered metrics then quantify.

Benchmark pipeline. Figure 4 summarizes the end-to-end pipeline from case sampling through layered reporting. Every reported number is generated from archived `report.json` and `run-summary.json` artifacts; the paper's tables are compiled from those artifacts rather than transcribed by hand.

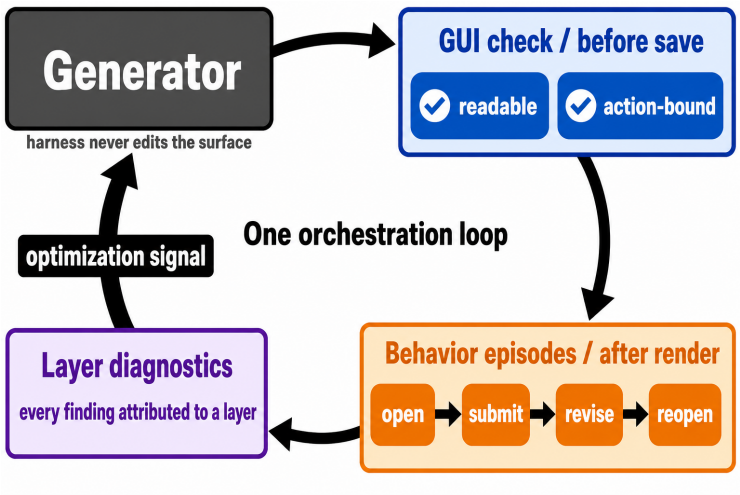

Figure 3: Evaluation orchestration. One loop coordinates the two evaluation stages — GUI checks before save, behavioral episodes after render — and layer-attributed diagnostics flow back to the generator as an optimization signal; the harness never edits the surface.

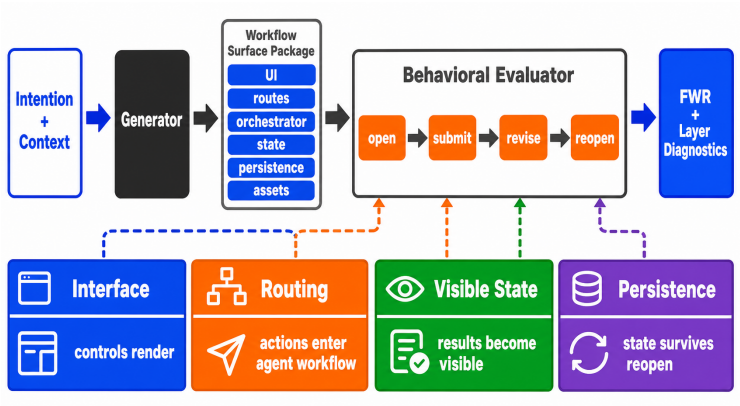

Figure 4: Benchmark pipeline. A generator receives only intention and context, emits a workflow-surface package, and is evaluated through reserved open-submit-revise-reopen episodes with per-layer assertions feeding the primary metric — Full Workflow Resolution (FWR), which credits a case only when every contract holds in every episode — and layer-attributed diagnostics.

5 Experiments

This section describes the evaluated systems and the diagnostic control suite, then presents the results. The experiments test the two claims of the Measurement contribution: that real-API grounding, not surface behavior, is the dominant unsolved capability of frontier agentic systems, and that assertion-level scores overstate workflow competence. Every number is compiled from archived run artifacts (Appendix C) by checked-in scripts, over the 75-case public-test split. No number appears without a corresponding artifact.

Control suite. Six scripted controls isolate the false positives that assertion-level scoring invites: *direct answer* (text, no surface), *static shell* (plausible controls bound to nothing), *route-only* (declared routes that never reach an API), *hidden state* (performs real API work but never shows the user a result), *schema-first* (well-formed persistent state schema with dead handlers), and *multi-agent* (coordinated read-only sub-agents that lose state on reopen). Each control passes some contracts and fails others by design (Table 5, Appendix A), so the control rows of Table 3 calibrate what APR alone would overcount.

System	Cases	FWR	APR	Route	API	DB	Grounded	Persist
Direct answer	75	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Static shell	75	0.00	0.18	0.00	0.00	0.00	0.00	0.00
Route-only	75	0.00	0.44	1.00	0.00	0.00	0.00	0.00
Hidden state	75	0.00	0.82	1.00	1.00	1.00	0.00	1.00
Schema-first	75	0.00	0.35	0.00	0.00	0.00	0.00	1.00
Multi-agent control	75	0.00	0.79	1.00	1.00	0.00	1.00	0.00
livegui-agent (gpt-5.4)	75	0.05	0.67	0.92	0.16	0.18	0.73	0.72
livegui-agent (opus-4.7)	75	0.15	0.72	0.99	0.23	0.27	0.80	0.75
no-grounding ablation (gpt-5.4)	25	0.00	0.67	1.00	0.00	0.00	0.84	0.76

Table 3: LIVEGUI-BENCH public-split results, compiled from archived `report.json` artifacts by `scripts/workflowgen/write-appworld-bench-table.cjs`. All metric columns are pass rates in $[0, 1]$; higher is better. FWR = fraction of cases whose assertions all pass in every episode (primary metric); APR = fraction of individual assertions passing (diagnostic); Route = GUI events entering the app-scoped agent workflow; API = named AppWorld API actually invoked; DB = application database mutated; Grounded = concrete returned entity visible; Persist = entity survives reopen. Controls (upper block) are each constructed to fail a targeted contract; their rows calibrate the metrics rather than compete.

Baseline systems and ablation. We evaluate the end-to-end livegui-agent baseline: a generator plus an agentic backend executor that, on each GUI action, selects and invokes real AppWorld APIs (authentication handled by the harness bridge) and writes the returned entities into visible state. We run this baseline on two backing models, `gpt-5.4` and `claude-opus-4.7`, behind one OpenAI-compatible endpoint, with temperature 0.2, a bounded orchestrator timeout, retry on transient transport errors only, and a pristine world reset before every run. A *no-grounding ablation* disables the API-invoking executor while keeping the identical generator, isolating the real-API contract; the ablation is run with `gpt-5.4` on a subset of the public split (case counts in Table 3). Model identity, API base, command, split, and case counts are logged per run (Appendix C); sealed-split evaluation remains reserved for hosted leaderboard runs.

Analyses. Four analyses follow from the construction of the suite. *(1)* APR–FWR *gap:* the hidden-state control row of Table 3 reaches APR 0.82 with FWR $= 0$ — it performs real API work, changes real databases, and persists state, yet never shows the user a result. The multi-agent control reaches APR 0.79 with FWR $= 0$. The gap persists for real systems (APR 0.67–0.72 against FWR 0.05–0.15): a single assertion-level score overstates workflow competence by construction. *(2) Contract localization:* in the per-contract columns of Table 3, both frontier systems route almost perfectly (0.92–0.99), persist state well (0.72–0.75), and echo the requested entities into visible results (0.73–0.80) — but only 0.16 (`gpt-5.4`) to 0.23 (`claude-opus-4.7`) of required API invocations actually happen, and database-change rates are similarly low (0.18–0.27). Grounding GUI actions in authenticated, correctly-parameterized API execution is the dominant unsolved capability; the no-grounding ablation passes zero API and database assertions by construction and therefore bounds exactly how much of the remaining score is surface behavior. *(3) Model gap:* `claude-opus-4.7` resolves nearly three times more workflows than `gpt-5.4` (0.15 vs 0.05), with the advantage concentrated in the API-invocation contract — evidence the benchmark discriminates tool-grounding ability rather than surface fluency. *(4) Per-domain difficulty:* single-user CRUD domains are tractable (`spotify` $5/8$ resolved by `claude-opus-4.7`, $2/8$ by `gpt-5.4`; `file_system` $3/6$ and $1/6$), while multi-party ledger domains (`venmo`, `splitwise`, `todoist` collaboration) resolve 0 for both models — workflows whose APIs require coordinating other principals' state remain fully open.

6 RELATED WORK

Operating existing software. GUI, OS, and app-operation benchmarks evaluate agents acting in environments that already exist: web navigation and grounding (Shi et al., 2017; Deng et al., 2023; Zhou et al., 2024; Koh et al., 2024; Yao et al., 2022; He et al., 2024), desktop and mobile control (Xie et al., 2024; Rawles et al., 2024; 2023; Kapoor et al., 2024), office and enterprise work (Drouin

et al., 2024; Wang et al., 2024; 2025), safety and live-web robustness (Levy et al., 2024; Garg et al., 2025; Anupam et al., 2025), and app ecosystems with database-level state checks (Trivedi et al., 2024). AppWorld in particular grades tasks by verifying resulting environment state rather than agent output, a philosophy LIVEGUI-BENCH adopts — but LIVEGUI-BENCH moves the evaluated object one step earlier: the agent must *create* the environment that later interactions operate. GUI-agent systems and data pipelines such as Aguvis and OS-Genesis (Xu et al., 2024; Sun et al., 2025) are complementary consumers of such environments.

Generating and rebuilding software. Code-generation benchmarks evaluate code as the artifact (Chen et al., 2021; Austin et al., 2021; Hendrycks et al., 2021; Lai et al., 2023; Lu et al., 2021; Cassano et al., 2023; Jain et al., 2024; Zhuo et al., 2024; Ren et al., 2020); repair benchmarks evaluate patches against issue checks (Just et al., 2014; Lin et al., 2017; Widyasari et al., 2020; Madeiral et al., 2019; Karampatsis & Sutton, 2020; Le Goues et al., 2015; Tan et al., 2017; Jimenez et al., 2024; Aleithan et al., 2024; Tian et al., 2026). ProgramBench evaluates rebuilding a program from an executable and documentation, scored by behavioral tests and fuzzed probes (Yang et al., 2026); LIVEGUI-BENCH shares its stance that implementation is secondary to behavior under reserved tests, but changes the artifact from a program to a user-operable workflow surface with an interaction loop.

Tool use, long-horizon work, and exams. Tool and workflow-agent benchmarks stress tool calling, multi-turn interaction, and professional outcomes (Liu et al., 2023; Qin et al., 2023; Li et al., 2023; Yao et al., 2024; Mialon et al., 2023; Yoran et al., 2024; Shridhar et al., 2021; Wang et al., 2022); all grade the work an agent performs, not the persistent surface it leaves behind for the user to continue that work. Agents' Last Exam grades complete work products with full-completion scoring (Sun et al., 2026); LIVEGUI-BENCH applies the same all-or-nothing convention at the case level, but to a recurring interaction loop rather than a single deliverable.

Generated interfaces. Work on generated and malleable interfaces splits along two evaluation axes. Generative-interface research synthesizes UIs from queries or evolving user data (Chen et al., 2025; Cao et al., 2025; Vaithilingam et al., 2024; Park et al., 2023) and evaluates authoring experience or human preference — *GUI evaluation without behavior evaluation*. Conversely, WebGen-Bench scores generated websites by dispatching an LLM navigation agent over per-functionality test cases (Lu et al., 2025), and AUI-Gym positions a computer-use agent as judge of task solvability on generated, agent-oriented UIs (Lin et al., 2025) — *behavior evaluation without GUI admission standards*, scored per test by model-based judges within a single session. LIVEGUI-BENCH requires both axes over one shared runtime: a GUI evaluator admits the surface a user must be able to read and operate, behavior is evaluated by a deterministic assertion runner over reserved open-submit-revise-reopen episodes, and credit is granted only to reopen-persistent, fully resolved workflows.

Benchmark methodology. LIVEGUI-BENCH follows current methodology on contamination control, dynamic collection, documentation, and shortcut analysis (Kiela et al., 2021; Srivastava et al., 2023; Liang et al., 2023; Ribeiro et al., 2020; Gebru et al., 2021; Mitchell et al., 2019; Bender & Friedman, 2018; Lee et al., 2022; Geirhos et al., 2020), with public and sealed splits, artifact-backed reports, and layer-attributed failures instead of a single score. Table 4 summarizes the positioning.

7 LIMITATIONS

First, asset validity and renderability checks are proxy diagnostics; LIVEGUI-BENCH measures verifiable interaction outcomes, not subjective GUI preference or visual design quality. Second, workflows are synthetic: they do not capture real personal data, long-term habit drift, or third-party integrations. This trade is deliberate: synthetic AppWorld services are what make every assertion deterministically checkable, and the grounding bottleneck they expose would, if anything, be harder against live third-party services. Third, public-split results are reported from archived artifacts, while sealed-split evaluation is reserved for hosted execution; until sealed-run artifacts are archived, no leaderboard claim is made. Fourth, code-first systems require adapters that preserve surface/action/state semantics, and adapter quality is a confound we report alongside results.

Family	Evaluated object	GUI eval	Behavior eval	Success
Code repair	Patch / code	–	Tests	Resolved issue
ProgramBench	Rebuilt program	–	Behavioral probes	Full program pass
GUI agents	Existing software	–	Action episodes	Task completion
AppWorld	Existing app world	–	API/GUI episodes	State-verified
ALE / exams	Work product	–	Outcome checks	Full pass
Generative interfaces	Generated UI	Preference	–	Human preference
WebGen-Bench	Generated website	–	LLM agent tests	Per-test accuracy
AUI-Gym	Generated agent UI	–	CUA judge	Task solvability
LIVEGUI-BENCH	Workflow surface	Admission gate	4-episode sequence	FWR

Table 4: Positioning. LIVEGUI-BENCH evaluates generated Agent-as-GUI workflow surfaces on both axes — GUI evaluation of the admitted surface and behavior evaluation of reserved episodes — with deterministic assertions and reopen persistence: the agent creates the environment that future user interactions will operate.

8 CONCLUSION

LIVEGUI-BENCH evaluates whether an agent can turn an end-user intention into an agent-native app — evaluated as an Agent-as-GUI workflow surface that future interactions can operate, revise, and recover — and whether that app actually drives the real services it names. The benchmark provides 191 cases grounded in eleven AppWorld app domains, scored by 2,172 deterministic assertions over execution traces, API-call records, and application-database diffs. The benchmark's standard is deliberately strict: an agent has not generated a usable workflow until every contract holds in every episode. Frontier agentic systems resolve 5–15% of public workflows: the surface contracts are largely solved, real-API grounding is the open capability, and multi-party ledger domains remain entirely unsolved.

ETHICS STATEMENT

LIVEGUI-BENCH is fully synthetic: intentions, contexts, and assertions are authored against the simulated AppWorld application environment and contain no real user data, personal identifiers, or scraped content. No human subjects were involved, and no real third-party service is called during evaluation — all APIs are AppWorld's executable simulations. The benchmark targets everyday personal workflows (planning, record keeping, shared expenses), and the evaluator checks behavioral contracts rather than content preferences. The sealed split is withheld to limit contamination, and leaderboard claims require hosted execution with archived reports.

REPRODUCIBILITY STATEMENT

All empirical numbers are generated from archived run artifacts (`report.json` and `run-summary.json`) by checked-in scripts rather than transcribed by hand. Appendix C defines the run-manifest and claim-traceability format that every reported run satisfies. The tracked case file, split construction rules, and evaluator endpoints are described in Sections 3 and 4.

USE OF LARGE LANGUAGE MODELS

Large language models were used to assist manuscript editing and to draft prompts for the four conceptual figures, which were rendered with an image-generation model. LLMs also appear inside the benchmark, as described in the main text: multi-agent pipelines authored and adversarially reviewed intentions and assertions (Section 3), and the behavior evaluator ends in an LLM judgment over the trajectory (Section 4). The reported metrics (FWR, APR, and every per-contract rate) derive from deterministic assertions only; the only LLM-derived statistic is the behavior evaluator's agreement probe, a check on the deterministic runner rather than a benchmark result.

REFERENCES

Reem Aleithan, Haoran Xue, Mohammad Mahdi Mohajer, Elijah Nnorom, Gias Uddin, and Song Wang. SWE-Bench+: Enhanced coding benchmark for llms, 2024. URL https://arxiv.org/abs/2410.06992.

Sagnik Anupam, Davis Brown, Shuo Li, Eric Wong, Hamed Hassani, and Osbert Bastani. BrowserArena: Evaluating LLM agents on real-world web navigation tasks, 2025. URL https://arxiv.org/abs/2510.02418.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. Program synthesis with large language models. In *arXiv preprint arXiv:2108.07732*, 2021. URL https://arxiv.org/abs/2108.07732.

Emily M. Bender and Batya Friedman. Data statements for natural language processing: Toward mitigating system bias and enabling better science. *Transactions of the Association for Computational Linguistics*, 6:587–604, 2018. URL https://aclanthology.org/Q18-1041/.

Yining Cao, Peiling Jiang, and Haijun Xia. Generative and malleable user interfaces with generative and evolving task-driven data model, 2025. URL https://arxiv.org/abs/2503.04084.

Federico Cassano, John Gouwar, Daniel Nguyen, Sydney Nguyen, Luna Phipps-Costin, Donald Pinckney, Ming-Ho Yee, Yangtian Zi, Carolyn Jane Anderson, Molly Q. Feldman, Arjun Guha, Michael Greenberg, and Abhinav Jangda. MultiPL-E: A scalable and polyglot approach to benchmarking neural code generation. *IEEE Transactions on Software Engineering*, 2023. URL https://arxiv.org/abs/2208.08227.

Jiaqi Chen, Yanzhe Zhang, Yutong Zhang, Yijia Shao, and Diyi Yang. Generative interfaces for language models, 2025. URL https://arxiv.org/abs/2508.19227.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code, 2021. URL https://arxiv.org/abs/2107.03374.

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. In *Advances in Neural Information Processing Systems*, 2023. URL https://arxiv.org/abs/2306.06070.

Alexandre Drouin, Maxime Gasse, Massimo Caccia, Issam H. Laradji, Manuel Del Verme, Tom Marty, Leo Boisvert, Megh Thakkar, Quentin Cappart, David Vazquez, Nicolas Chapados, and Alexandre Lacoste. WorkArena: How capable are web agents at solving common knowledge work tasks?, 2024. URL https://arxiv.org/abs/2403.07718.

Divyansh Garg, Shaun VanWeelden, Diego Caples, Andis Draguns, Nikil Ravi, Pranav Putta, Naman Garg, Tomas Abraham, Michael Lara, Federico Lopez, James Liu, Atharva Gundawar, Prannay Hebbar, Youngchul Joo, Jindong Gu, Charles London, Christian Schroeder de Witt, and Sumeet Motwani. REAL: Benchmarking autonomous agents on deterministic simulations of real websites, 2025. URL https://arxiv.org/abs/2504.11543.

Timnit Gebru, Jamie Morgenstern, Briana Vecchione, Jennifer Wortman Vaughan, Hanna Wallach, Hal Daumé III, and Kate Crawford. Datasheets for datasets. *Communications of the ACM*, 64(12): 86–92, 2021. doi: 10.1145/3458723. URL https://doi.org/10.1145/3458723.

Robert Geirhos, Jörn-Henrik Jacobsen, Claudio Michaelis, Richard Zemel, Wieland Brendel, Matthias Bethge, and Felix A. Wichmann. Shortcut learning in deep neural networks. *Nature Machine Intelligence*, 2:665–673, 2020. doi: 10.1038/s42256-020-00257-z. URL https://doi.org/10.1038/s42256-020-00257-z.

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

Dan Hendrycks, Steven Basart, Saurav Kadavath, Mantas Mazeika, Akul Arora, Ethan Guo, Collin Burns, Samir Puranik, Horace He, Dawn Song, and Jacob Steinhardt. Measuring coding challenge competence with APPS. In *Advances in Neural Information Processing Systems Datasets and Benchmarks Track*, 2021. URL https://arxiv.org/abs/2105.09938.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida I. Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. LiveCodeBench: Holistic and contamination free evaluation of large language models for code, 2024. URL https://arxiv.org/abs/2403.07974.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. SWE-bench: Can language models resolve real-world GitHub issues? In *International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2310.06770.

René Just, Darioush Jalali, and Michael D. Ernst. Defects4J: A database of existing faults to enable controlled testing studies for Java programs. In *International Symposium on Software Testing and Analysis*, 2014. doi: 10.1145/2610384.2628055. URL https://doi.org/10.1145/2610384.2628055.

Raghav Kapoor, Yash Parag Butala, Melisa Russak, Jing Yu Koh, Kiran Kamble, Waseem Alshikh, and Ruslan Salakhutdinov. OmniACT: A dataset and benchmark for enabling multimodal generalist autonomous agents for desktop and web, 2024. URL https://arxiv.org/abs/2402.17553.

Rafael-Michael Karampatsis and Charles Sutton. How often do single-statement bugs occur? the ManySStuBs4J dataset of single-statement bugs. In *Proceedings of the International Conference on Mining Software Repositories*, 2020. URL https://arxiv.org/abs/1905.13334.

Douwe Kiela, Max Bartolo, Yixin Nie, Divyansh Kaushik, Atticus Geiger, Zhengxuan Wu, Bertie Vidgen, Grusha Prasad, Amanpreet Singh, Pratik Ringshia, Zhiyi Ma, Tristan Thrush, Sebastian Riedel, Zeerak Waseem, Pontus Stenetorp, Robin Jia, Mohit Bansal, Christopher Potts, and Adina Williams. Dynabench: Rethinking benchmarking in NLP. *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2021. doi: 10.18653/v1/2021.naacl-main.324. URL https://aclanthology.org/2021.naacl-main.324/.

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

Yuhang Lai, Chengxi Li, Yiming Wang, Tianyi Zhang, Ruiqi Zhong, Luke Zettlemoyer, Wen-tau Yih, Daniel Fried, Sida I. Wang, and Tao Yu. DS-1000: A natural and reliable benchmark for data science code generation. In *International Conference on Machine Learning*, 2023. URL https://arxiv.org/abs/2211.11501.

Claire Le Goues, Neal Holtschulte, Edward K. Smith, Yuriy Brun, Premkumar Devanbu, Stephanie Forrest, and Westley Weimer. The ManyBugs and IntroClass benchmarks for automated repair of C programs. *IEEE Transactions on Software Engineering*, 41(12):1236–1256, 2015. doi: 10.1109/TSE.2015.2454513. URL https://doi.org/10.1109/TSE.2015.2454513.

Katherine Lee, Daphne Ippolito, Andrew Nystrom, Chiyuan Zhang, Douglas Eck, Chris Callison-Burch, and Nicholas Carlini. Deduplicating training data makes language models better. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, 2022. doi: 10.18653/v1/2022.acl-long.577. URL https://aclanthology.org/2022.acl-long.577/.

Ido Levy, Ben Wiesel, Sami Marreed, Alon Oved, Avi Yaeli, Nir Mashkif, and Segev Shlomov. ST-WebAgentBench: A benchmark for evaluating safety and trustworthiness in web agents, 2024. URL https://arxiv.org/abs/2410.06703.

Minghao Li, Yingxiu Zhao, Bowen Yu, Feifan Song, Hangyu Li, Haiyang Yu, Zhoujun Li, Fei Huang, and Yongbin Li. API-Bank: A comprehensive benchmark for tool-augmented LLMs, 2023. URL https://arxiv.org/abs/2304.08244.

Percy Liang, Rishi Bommasani, Tony Lee, Dimitris Tsipras, Dilara Soylu, Michihiro Yasunaga, Yian Zhang, Deepak Narayanan, Yuhuai Wu, Ananya Kumar, et al. Holistic evaluation of language models. *Transactions on Machine Learning Research*, 2023. URL https://arxiv.org/abs/2211.09110.

Derrick Lin, James Koppel, Angela Chen, and Armando Solar-Lezama. QuixBugs: A multi-lingual program repair benchmark set based on the Quixey challenge. In *Proceedings Companion of the ACM SIGPLAN International Conference on Systems, Programming, Languages, and Applications: Software for Humanity*, 2017. doi: 10.1145/3135932.3135941. URL https://doi.org/10.1145/3135932.3135941.

Kevin Qinghong Lin, Siyuan Hu, Linjie Li, Zhengyuan Yang, Lijuan Wang, Philip Torr, and Mike Zheng Shou. Computer-use agents as judges for generative user interface, 2025. URL https://arxiv.org/abs/2511.15567.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, Shudan Zhang, Xiang Deng, Aohan Zeng, Zhengxiao Du, Chenhui Zhang, Sheng Shen, Tianjun Zhang, Yu Su, Huan Sun, Minlie Huang, Yuxiao Dong, and Jie Tang. AgentBench: Evaluating LLMs as agents, 2023. URL https://arxiv.org/abs/2308.03688.

Shuai Lu, Daya Guo, Shuo Ren, Junjie Huang, Alexey Svyatkovskiy, Ambrosio Blanco, Colin Clement, Dawn Drain, Daxin Jiang, Duyu Tang, Ge Li, Lidong Zhou, Linjun Shou, Long Zhou, Michele Tufano, Ming Gong, Ming Zhou, Nan Duan, Neel Sundaresan, Shao Kun Deng, Shengyu Fu, and Shujie Liu. CodeXGLUE: A machine learning benchmark dataset for code understanding and generation. In *Advances in Neural Information Processing Systems Datasets and Benchmarks Track*, 2021. URL https://arxiv.org/abs/2102.04664.

Zimu Lu, Yunqiao Yang, Houxing Ren, Haotian Hou, Han Xiao, Ke Wang, Weikang Shi, Aojun Zhou, Mingjie Zhan, and Hongsheng Li. WebGen-Bench: Evaluating LLMs on generating interactive and functional websites from scratch, 2025. URL https://arxiv.org/abs/2505.03733.

Fernanda Madeiral, Simon Urli, Marcelo Maia, and Martin Monperrus. BEARS: An extensible Java bug benchmark for automatic program repair studies. In *Proceedings of the International Conference on Software Analysis, Evolution and Reengineering*, 2019. doi: 10.1109/SANER.2019.8667991. URL https://doi.org/10.1109/SANER.2019.8667991.

Grégoire Mialon, Clémentine Fourrier, Craig Swift, Thomas Wolf, Yann LeCun, and Thomas Scialom. GAIA: A benchmark for general AI assistants, 2023. URL https://arxiv.org/abs/2311.12983.

Margaret Mitchell, Simone Wu, Andrew Zaldivar, Parker Barnes, Lucy Vasserman, Ben Hutchinson, Elena Spitzer, Inioluwa Deborah Raji, and Timnit Gebru. Model cards for model reporting. In *Proceedings of the Conference on Fairness, Accountability, and Transparency*, 2019. doi: 10.1145/3287560.3287596. URL https://doi.org/10.1145/3287560.3287596.

Joon Sung Park, Joseph C. O'Brien, Carrie J. Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*, 2023. doi: 10.1145/3586183.3606763. URL https://doi.org/10.1145/3586183.3606763.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. ToolLLM: Facilitating large language models to master 16000+ real-world APIs, 2023. URL https://arxiv.org/abs/2307.16789.

Christopher Rawles, Alice Li, Daniel Rodriguez, Oriana Riva, and Timothy Lillicrap. Android in the wild: A large-scale dataset for Android device control. In *Advances in Neural Information Processing Systems*, 2023. URL https://arxiv.org/abs/2307.10088.

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

Shuo Ren, Daya Guo, Shuai Lu, Long Zhou, Shujie Liu, Duyu Tang, Neel Sundaresan, Ming Zhou, Ambrosio Blanco, and Shuai Ma. CodeBLEU: A method for automatic evaluation of code synthesis, 2020. URL https://arxiv.org/abs/2009.10297.

Marco Tulio Ribeiro, Tongshuang Wu, Carlos Guestrin, and Sameer Singh. Beyond accuracy: Behavioral testing of NLP models with CheckList. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, 2020. doi: 10.18653/v1/2020.acl-main.442. URL https://aclanthology.org/2020.acl-main.442/.

Tianlin Shi, Andrej Karpathy, Linxi Fan, Jonathan Hernandez, and Percy Liang. World of bits: An open-domain platform for web-based agents. In *International Conference on Machine Learning*, 2017. URL https://proceedings.mlr.press/v70/shi17a.html.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Cote, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. ALFWorld: Aligning text and embodied environments for interactive learning. In *International Conference on Learning Representations*, 2021. URL https://arxiv.org/abs/2010.03768.

Aarohi Srivastava, Abhinav Rastogi, Abhishek Rao, Abu Awal Md Shoeb, Abubakar Abid, Adam Fisch, Adam R. Brown, Adam Santoro, Aditya Gupta, Adrià Garriga-Alonso, et al. Beyond the imitation game: Quantifying and extrapolating the capabilities of language models. *Transactions on Machine Learning Research*, 2023. URL https://arxiv.org/abs/2206.04615.

Qiushi Sun, Kanzhi Cheng, Zichen Ding, Chuanyang Jin, Yian Wang, Fangzhi Xu, Zhenyu Wu, Chengyou Jia, Liheng Chen, Zhoumianze Liu, Ben Kao, Guohao Li, Junxian He, Yu Qiao, and Zhiyong Wu. OS-Genesis: Automating GUI agent trajectory construction via reverse task synthesis. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, pp. 5555–5579, 2025. doi: 10.18653/v1/2025.acl-long.277. URL https://aclanthology.org/2025.acl-long.277/.

Yiyou Sun, Xinyang Han, Weichen Zhang, Yuanbo Pang, Tianyu Wang, Yuhan Cao, Yixiao Huang, Chris Duroiu, Haoyun Zhang, Jeffrey Lin, Weishu Zhang, Tyler Zeng, Ying Yan, Bo Liu, Hanson Wen, Mingyang Xu, Xiaoyuan Liu, Zimeng Chen, Weiyan Shi, Amanda Dsouza, Vincent Sunn Chen, Dawn Song, et al. Agents' last exam, 2026. URL https://arxiv.org/abs/2606.05405.

Shin Hwei Tan, Jooyong Yi, Yulis, Sergey Mechtaev, and Abhik Roychoudhury. Codeflaws: A programming competition benchmark for evaluating automated program repair tools. In *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*, pp. 180–182, 2017. doi: 10.1109/ICSE-C.2017.76. URL https://doi.org/10.1109/ICSE-C.2017.76.

Muxin Tian, Zhe Wang, Blair Yang, Zhenwei Tang, Kunlun Zhu, Honghua Dong, Hanchen Li, Xinni Xie, Guangjing Wang, and Jiaxuan You. SWE-Bench Mobile: Can large language model agents develop industry-level mobile applications?, 2026. URL https://arxiv.org/abs/2602.09540.

Harsh Trivedi, Tushar Khot, Mareike Hartmann, Ruskin Manku, Vinty Dong, Edward Li, Shashank Gupta, Ashish Sabharwal, and Niranjan Balasubramanian. AppWorld: A controllable world of apps and people for benchmarking interactive coding agents. In *Annual Meeting of the Association for Computational Linguistics*, 2024. URL https://arxiv.org/abs/2407.18901.

Priyan Vaithilingam, Elena L. Glassman, Jeevana Priya Inala, and Chenglong Wang. DynaVis: Dynamically synthesized ui widgets for visualization editing. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*, 2024. doi: 10.1145/3613904.3642639. URL https://doi.org/10.1145/3613904.3642639.

Ruoyao Wang, Peter Jansen, Marc-Alexandre Cote, and Prithviraj Ammanabrolu. ScienceWorld: Is your agent smarter than a 5th grader? In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2022. doi: 10.18653/v1/2022.emnlp-main.775. URL https://aclanthology.org/2022.emnlp-main.775/.

Weixuan Wang, Dongge Han, Daniel Madrigal Diaz, Jin Xu, Victor Rühle, and Saravan Rajmohan. OdysseyBench: Evaluating LLM agents on long-horizon complex office application workflows, 2025. URL https://arxiv.org/abs/2508.09124.

Zilong Wang, Yuedong Cui, Li Zhong, Zimin Zhang, Da Yin, Bill Yuchen Lin, and Jingbo Shang. OfficeBench: Benchmarking language agents across multiple applications for office automation, 2024. URL https://arxiv.org/abs/2407.19056.

Ratnadira Widyasari, Sheng Qin Sim, Camellia Lok, Haodi Qi, Jack Phan, Qijin Tay, Constance Tan, Fiona Wee, Jodie Ethelda Tan, Yuheng Yieh, Brian Goh, Ferdian Thung, Hong Jin Kang, Thong Hoang, David Lo, and Eng Lieh Ouh. BugsInPy: A database of existing bugs in Python programs to enable controlled testing and debugging studies. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2020. doi: 10.1145/3368089.3417943. URL https://doi.org/10.1145/3368089.3417943.

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

Yiheng Xu, Zekun Wang, Junli Wang, Dunjie Lu, Tianbao Xie, Amrita Saha, Doyen Sahoo, Tao Yu, and Caiming Xiong. Aguvis: Unified pure vision agents for autonomous GUI interaction, 2024. URL https://arxiv.org/abs/2412.04454.

John Yang, Kilian Lieret, Jeffrey Ma, Parth Thakkar, Dmitrii Pedchenko, Sten Sootla, Emily McMilin, Pengcheng Yin, Rui Hou, Gabriel Synnaeve, Diyi Yang, and Ofir Press. ProgramBench: Can language models rebuild programs from scratch?, 2026. URL https://arxiv.org/abs/2605.03546.

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. WebShop: Towards scalable real-world web interaction with grounded language agents. In *Advances in Neural Information Processing Systems*, 2022. URL https://arxiv.org/abs/2207.01206.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ-bench: A benchmark for tool-agent-user interaction in real-world domains, 2024. URL https://arxiv.org/abs/2406.12045.

Ori Yoran, Samuel Joseph Amouyal, Chaitanya Malaviya, Ben Bogin, Ofir Press, and Jonathan Berant. AssistantBench: Can web agents solve realistic and time-consuming tasks? In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2024. URL https://aclanthology.org/2024.emnlp-main.505/.

Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. WebArena: A realistic web environment for building autonomous agents. In *International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2307.13854.

Terry Yue Zhuo, Minh Chien Vu, Jenny Chim, Han Hu, Wenhao Yu, Ratnadira Widyasari, Imam Nur Bani Yusuf, Haolan Zhan, Junda He, Indraneil Paul, Simon Brunner, Chen Gong, Thong Hoang, Yifan Liu, Muhammad Asaduzzaman, Zhenchang Xing, Xin Xia, and David Lo. BigCodeBench: Benchmarking code generation with diverse function calls and complex instructions, 2024. URL https://arxiv.org/abs/2406.15877.

A Dataset Details

This appendix records the dataset statistics summarized in the main text. The generator-visible case fields are intention and context; the app-domain and assertion fields below are evaluator-side metadata.

Control	Construction	Contract failed by design
Direct answer	Returns text with no surface	Visible task state
Static shell	Renders plausible controls bound to nothing	Action routing
Route-only	Declares agent routes but never writes visible state	Grounded visible update
Hidden state	Persists agent results in backend memory the user cannot see	Grounded visible update, reopen persistence
Schema-first	Emits a well-formed state schema without visible workflow output	Visible task state, grounded visible update
Multi-agent	Coordinates sub-agents but surfaces results partially and loses revision state on reopen	Grounded visible update, reopen persistence

Table 5: The six scripted controls. Each passes some assertion layers cleanly while failing a targeted contract.

Split	Cases
dev	37
public_test	75
sealed_test	79
total	191

Table 6: LiveGUI-Bench split sizes generated from the case file.

App domain	Cases
simple_note	22
spotify	21
splitwise	21
todoist	21
venmo	20
amazon	19
file_system	16
gmail	15
phone	13
gmail_todoist	12
amazon_venmo	11

Table 7: App-domain distribution. Domain labels support auditing and are not exposed to the generator.

Assertion type	Count
route_enters_orchestrator	382
visible_result_grounded	382
appworld_api_called	376
appworld_db_changed	268
input_available	191
action_available	191
route_exists	191
state_persists	191
total	2172

Table 8: Assertion-type counts. Every assertion is checked deterministically against execution artifacts.

B RELEASE GOVERNANCE

The benchmark release separates public development from sealed evaluation. Public artifacts support debugging and reproducibility; sealed-test claims require hosted execution and archived reports.

Split	Cases	Builder-visible	Held for evaluation	Claim policy
development	37	full cases and rubrics	none	engineering evidence only
public-test	75	public cases and rubrics	archived reports required	public claims require report JSON and run manifest
sealed-test	79	hashes, metadata, schema, adapter API, and server policy	sealed case bodies, episodes, and assertion values	sealed-test ranking requires hosted run, archived app package,...
sealed-refresh	0	hash commitments and metadata only	periodic paraphrase and new-profile additions	refresh results are separately versioned

Table 9: LIVEGUI-BENCH release governance.

Release gate	Checked condition
schema	required fields, split names, duplicate ids, episode shape, and allowed assertion types
leakage	family labels and analysis metadata are absent from generator-facing fields
reporting	reported values are recorded in report JSON, run summaries, and generated tables
runtime	runtime fails on missing package, route, state, or asset contracts
paper	paper sources, PDFs, and figure OCR pass the asset audit

Table 10: Release gates for schema, leakage, reporting, runtime, and paper artifact checks.

C RUN-ARTIFACT PROTOCOL

Every reported run must satisfy the following protocol before its numbers may appear in the main text. (1) The run writes `report.json` and `run-summary.json` under a unique artifact directory, logging model identity, API base, command, split, case count, latency, temperature, token budget, and cost. (2) Every table in the paper is generated from those artifacts by checked-in scripts, never edited by hand. (3) A claim-traceability ledger maps each main-text claim to its source artifact, and a run manifest lists every run with its command so controls and diagnostics can be rerun through the same adapter boundary. (4) Sealed-split numbers additionally require hosted execution with archived reports. The ledger and manifest formats are checked automatically by the repository's paper-readiness gate.

D EXAMPLE BEHAVIOR TRAJECTORIES

The online behavior evaluator archives a per-step snapshot of the rendered surface (fields with live values, action buttons, and the executed operation), which the harness renders into trajectory frames. Figures 5 and 6 show two archived trajectories. In the first, a fitness check-in surface completes a submit–revise sequence: each click routes into the app-scoped workflow and the visible plan updates. In the second, a purchase-review backlog surface accepts input and routes correctly, but the trajectory judgment fails the session because the visible state never reflects the real Amazon order data the intention requires — the same real-API grounding gap the deterministic assertions localize.

Figure 7 isolates the grounding bottleneck by contrasting two apps generated from unrelated cross-domain intentions. On the left, an Amazon review-backlog surface routes a submit into the app-scoped workflow, which selects and invokes the real `create_product_review` API; the returned entity (*"Created a 5-star review … with review id 15968"*) is written back into the visible result block, so the state the user sees is grounded in a real database write. On the right, a Venmo recurring-rent surface exposes equally well-structured semantic slots (per-tenant rooms, amounts, paid/unpaid status, late fee), but the executed action degenerates to echoing the app title rather than issuing the real per-tenant payment requests the intention names — no grounded entity ever reaches the surface. The two frames share the same generated-app machinery; the passing workflow differs from the failing one only in whether the backend agent completed real, visible API grounding — precisely the property LIVEGUI-BENCH's grounding contracts measure.

Figure 5: Trajectory frames from a passing episode on a fitness check-in surface (fill → first submit → revision submit): each submit routes into the app-scoped workflow and the visible plan updates. The banner records the executed operation and its outcome; the card shows the live field values the user sees at that step.

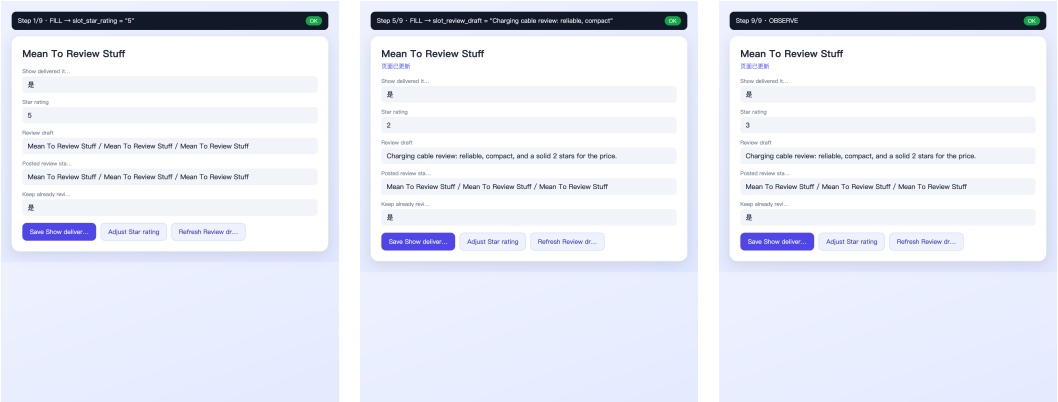

Figure 6: Trajectory frames from a purchase-review backlog surface. Controls fill and route correctly, but the visible state never surfaces the real order entities the intention names, so the episode fails — the behavior-level view of the API-grounding bottleneck.

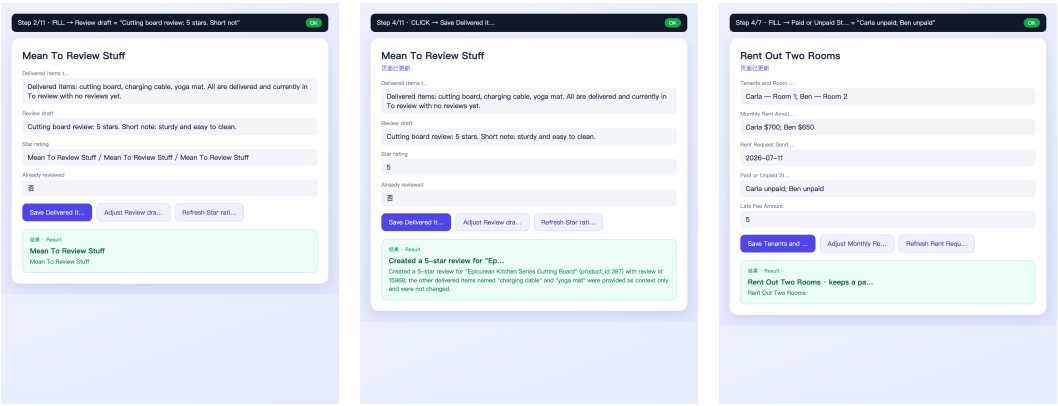

Figure 7: Cross-domain grounding contrast. *Left/center:* an Amazon review-backlog surface opens, then a submit routes into the app-scoped workflow, which invokes the real `create_product_review` API and writes the returned review entity (*review id 15968*) into the visible result block — a grounded, passing step. *Right:* a Venmo recurring-rent surface with equally structured semantic slots, whose action degenerates to echoing the app title instead of issuing real per-tenant payment requests, so no grounded entity reaches the surface and the episode fails. Same generated-app machinery; real-API grounding is the discriminator.

E Illustrative Episode Schema

This schema-only example illustrates the episode format; it is not a sealed-test case.

```
{
  "id": "spotify-weekly-commute-playlist__submit",
  "kind": "submit",
  "userMessage": "Vibe is upbeat indie, seed artists Phoenix and
      Two Door Cinema Club, about 20 songs for Monday Commute.",
  "assertions": [
    { "type": "route_enters_orchestrator", "value": true },
    { "type": "appworld_api_called", "value": "create_playlist" },
    { "type": "appworld_db_changed", "value": "spotify" },
    { "type": "visible_result_grounded", "value": "Monday Commute" }
  ]
}
```