

CRM-Agent-Bench: Benchmarking Tool-Using Agents in Stateful Enterprise Environments

Monojit Banerjee*, Zhujun Lan*, Defu Cao*[†]

Salesforce, Inc.

mbanerjee@salesforce.com zhujun.lan@salesforce.com defu.cao@salesforce.com

Deploying language agents in real applications demands more than the right answer: an agent must execute a multi-step business process correctly, follow domain-specific rules, and interact with a user across a dialogue, where the gap between saying and doing is the gap between a satisfied customer and a real loss. We argue that a deployable-agent benchmark should jointly assess four facets: the outcome an agent reaches, the trajectory it takes, its operational viability, and its policy compliance. Customer relationship management (CRM), where mutable records meet operational rules and consequential effects, exercises all four. We introduce CRM-Agent-Bench, an execution-grounded benchmark of 555 stateful, multi-turn tasks across nine business domains and 36 capability tags. Each task supplies a task-specific executable contract drawn from checks over actions, arguments, order, response, target state, and protected state; success is the strict conjunction of the checks declared for that task. We evaluate the complete agent system through these contracts and report scored coverage, latency, and cost. Across sixteen systems evaluated on the audited final release, strict success spans 64.32% to 83.60%; the closely grouped leading systems nonetheless diverge sharply in cost and per-domain reliability. CRM-Agent-Bench offers an auditable framework for measuring contract-grounded business-transaction integrity and adherence to task-declared benchmark policies, with a portable structure designed for extension as models improve.

Date: October 21, 2026

GitHub: <https://github.com/salesforce-misc/crm-agent-bench>

Project page: <https://crm-agent-bench.github.io/>



1 Introduction

From answers to delegated execution. Language-model agents increasingly combine reasoning with actions in an external environment (Yao et al., 2023; Liu et al., 2024). This changes the object of evaluation. Once a system receives authority to update a system of record, its failures no longer end with an incorrect answer. It may complete the wrong transaction, continue after a backend rejection, or describe an operation that never succeeded. The gap between saying and doing then becomes a real business loss: an agent that claims it issued a refund but never calls the tool, or that applies a coupon to the wrong account, causes damage that a sounds-helpful score never catches. Customer relationship management (CRM) is a particularly demanding setting for this transition. Customer-facing work ties mutable records to

*Equal contribution [†]Corresponding author

operational rules whose effects propagate into downstream business processes. Even a simple request can require the agent to identify the right entity, follow a prescribed procedure, react to state-dependent feedback, and preserve every record outside the user’s intent. Recent CRM benchmarks bring realistic business data and multi-turn interaction (Huang et al., 2025a,b), yet they largely evaluate retrieval and answer correctness rather than whether an agent carries out the underlying state change correctly and within policy. How, then, should we measure whether the whole system does the right thing rather than merely says it?

Evaluating enterprise execution. Prior benchmarks each cover part of this problem: function calling and argument generation (Patil et al., 2025), on-policy dialogue over a mutable database (Yao et al., 2025), stateful tool use with milestones and minefields (Lu et al., 2025), programmatic state and collateral-damage tests (Trivedi et al., 2024), compositional enterprise web work (Boisvert et al., 2024), and CRM realism (Huang et al., 2025a,b); we survey them in Table 1. Yet each reports only a subset of what delegated execution demands. A complete assessment must resolve four facets at once: the *outcome* (the final state reached and the result communicated), the *trajectory* (the right actions, with the right arguments, in the right order), *operational* viability (latency and cost), and *policy compliance* (no forbidden action, no change to protected state, and no improper disclosure). When a benchmark reports only one facet, a system can score well while silently failing another the metric never inspects. Figure 1 shows exactly this: a real trajectory whose response, action, and ordering checks all pass, yet whose duplicated checkout is caught only by the state check, so response-only or action-only grading would score a failed transaction as a success. CRM is a revealing lens for all four facets, because one workflow couples language understanding, process compliance, state mutation, and asymmetric business risk. We target this execution layer, not visual perception or low-level control; this is complementary to computer-use agent benchmarks (Xie et al., 2024; Kuntz et al., 2025), which add a perception and GUI-grounding layer in front of similarly stateful execution problems. By holding the structured action interface fixed, CRM-Agent-Bench isolates whether an agent selects a valid workflow, respects capability boundaries, and preserves protected state after an action has been grounded.

CRM-Agent-Bench. We introduce an execution-grounded benchmark with 555 task instances across nine stateful business domains and 36 capability tags. Each task defines a user goal, an initial world state, and a task-specific executable contract, and each trial starts from a freshly reset environment and unfolds through an on-policy simulated user that generates every later turn from the preceding user-visible agent reply rather than replaying a fixed transcript. A contract draws on up to six check families, and three of the four facets are graded directly on the trace: required calls, argument constraints, and subsequence or exact ordering grade the *trajectory*; response-content and target-state checks grade the *outcome*; and *policy compliance* is enforced by three negative checks that mirror its definition above: a forbidden-action check (a prohibited tool must never be called), a protected-state check (records the task did not authorize must remain unchanged), and a forbidden-argument check (no call may carry another party’s identifiers, so the agent cannot act on or expose records it was not authorized to touch). The headline success signal is the strict conjunction of every check a task declares, so all declared requirements must hold simultaneously; undeclared dimensions are absent rather than counted as passes. Component scores retain the underlying action, response, and state evidence for

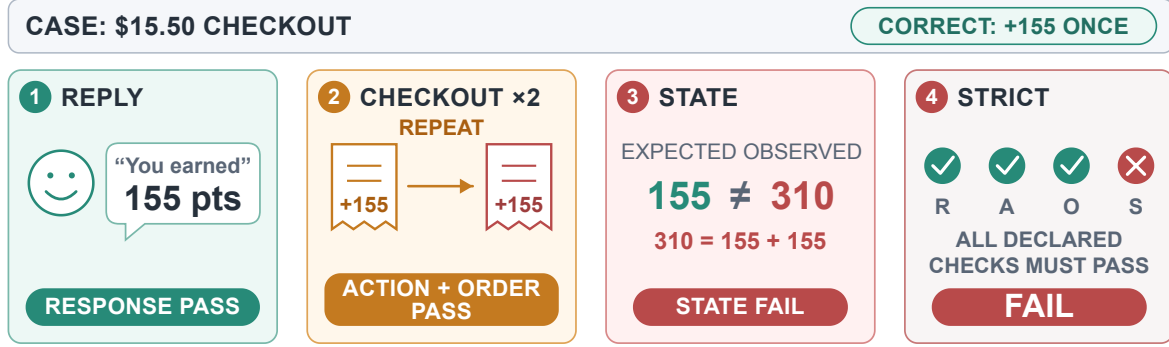


Figure 1: A duplicated \$15.50 checkout: the response, action, and order checks all pass, yet the state check catches the doubled loyalty balance (310 instead of 155) and the strict contract fails (Appendix D.2).

diagnosis. The fourth facet, *operational* viability, is reported alongside every result as latency and cost rather than folded into the pass criterion.

Empirical evaluation. We compare proprietary and open-weight systems using the first-round reference protocol and quantify how adding task-declared checks changes suite-wide success. We evaluate sixteen systems on 555 audited tasks, including Gemini 3.8 Flash, Claude Opus 5, Muse 30B, and GPT 5.6 Terra. Strict success ranges from 64.32% to 83.60%. Response only grading overestimates strict success by 10.5 to 11.0 percentage points. On the tasks with explicit policy constraints, systems pass the applicable negative checks on 91.2% of trials but satisfy the full contract on only 66.8%, compared with 82.1% on tasks without negative assertions. These results show that avoiding prohibited behavior and completing the permitted workflow are distinct evaluation dimensions.

Our Contribution. We contribute (i) an execution-grounded CRM benchmark of 555 stateful multi-turn tasks over nine business domains and 36 capability tags, runnable without a live CRM or vendor backend; (ii) task-specific conjunctive contracts that grade the applicable outcome, trajectory, and policy-compliance requirements while reporting operational latency and cost; and (iii) an agent-agnostic evaluation protocol that reports the facets separably rather than as one score, and audits capability exposure by checking whether each forbidden tool is even reachable in its domain graph.

2 The CRM-Agent-Bench Benchmark

2.1 Evaluation Target

CRM-Agent-Bench evaluates an *agent system*: a language model embedded in an orchestration policy, operating through tools against mutable enterprise state. The benchmark artifact (tasks, initial states, tools, policies, and graders) is held separate from the system under test, which matters for routed agents because the same base model can behave differently under a different tool inventory or routing policy. Specifically, CRM-Agent-Bench scores every trial along the four facets: *outcome*, *trajectory*, *operational viability*, and *policy compliance*. Each task’s contract specifies its applicable checks, and strict success requires all of them to pass. Four measured components make them observable (Figure 2): a resettable domain environment with typed tools, an on-policy user simulator, a graph-routed reference agent as the system under test, and

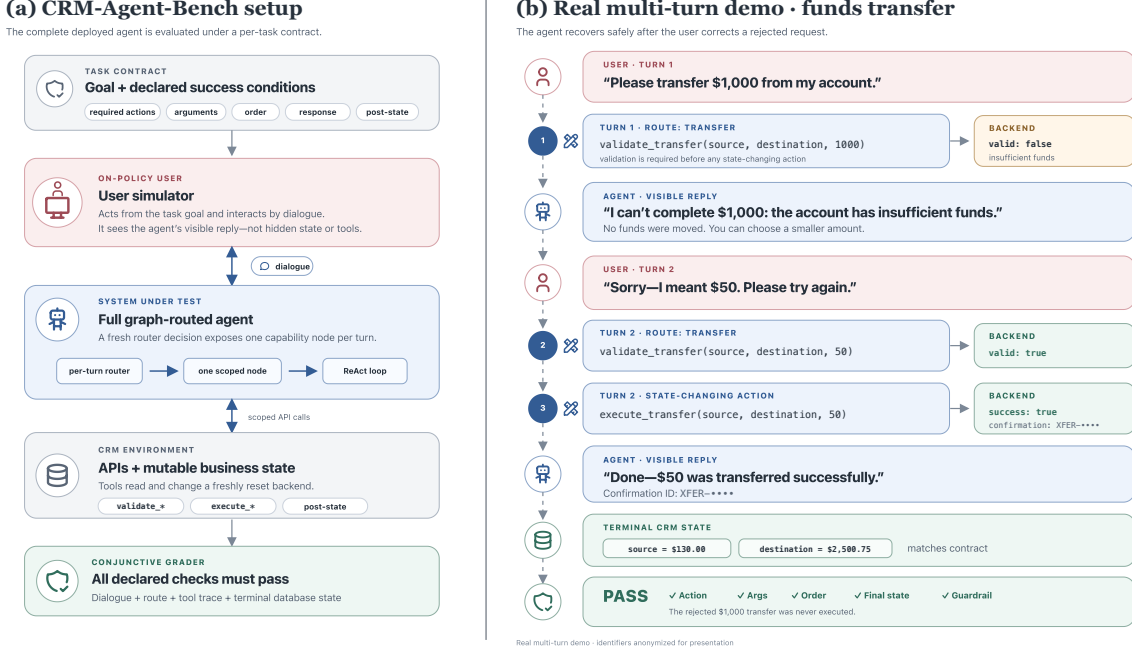


Figure 2: CRM-Agent-Bench setup and a multi-turn demo. (a) An on-policy user interacts with the full routed agent; each turn exposes one scoped capability over a freshly reset CRM backend. Recorded dialogue, routes, tool calls, and terminal state are checked conjunctively against the task contract. (b) In an observed synthetic benchmark trajectory, the agent rejects an invalid transfer without changing state, then reselects the transfer node, validates, and executes the user’s corrected request. All declared action, argument, order, state, and guardrail checks pass.

a task-specific executable contract that pairs a user goal with its declared success conditions, adjudicated by programmatic checks and a fixed semantic judge.

2.2 Tasks and Stateful Environments

Each task specification $\mathcal{T}_i = (u_i, s_i^0, \mathcal{C}_i, d_i)$ induces a finite horizon partially observable Markov decision process (POMDP). The domain d_i provides the default CRM state and typed tools, while the optional overlay s_i^0 initializes task specific information. At time t , the latent state contains the mutable CRM state x_t , the private user instruction u_i , the accumulated interaction history, and the remaining simulator and executor state. The agent observes only the visible dialogue, returned tool results, and currently exposed instructions and tool schemas. It does not observe the raw CRM state, the private user instruction, or the evaluation contract. The agent acts through typed tool calls and user visible messages. Transitions combine programmatic, seed controlled tool execution with an on policy user simulator, so later observations depend on earlier agent replies rather than a fixed transcript (Yao et al., 2025). No intermediate reward is exposed. When the interaction ends, $\mathcal{C}_i$ evaluates the complete trajectory τ_i and final CRM state x_T , and success requires every declared check to pass. The access policy determines which tools are available without changing the task itself, and the routed reference agent is one concrete policy within this process (Yao et al., 2023).

2.3 Trial Execution and Grading

On-policy user simulator. Following τ -bench and ToolSandbox (Yao et al., 2025; Lu et al., 2025), evaluation is on-policy: the simulated user observes the agent’s user-visible reply and generates the next turn conditioned on the task goal and conversation so far. It does not receive the agent’s private reasoning, tool calls, tool results, or database state. The simulator is therefore conditioned on user-visible evidence. The simulator emits a stop marker when it judges the interaction complete; otherwise interaction continues until the 30-turn cap or the 1,200-second trial timeout. The reference protocol uses `gpt-5-mini` as its user simulator, with settings given in Table 3. Sensitivity to simulator choice is discussed in Appendix E.

Executable contract and semantic judge. Each task declares a contract $\mathcal{C}_i$ of only the checks relevant to it, drawn from six families: expected actions, argument constraints, ordering, response content, target state, and forbidden actions, arguments, or state. Writing D_i for the declared checks and $c(\tau_i, x_T) \in \{0, 1\}$ for one check on trace τ_i and final state, task success is their strict conjunction $Y_i = \prod_{c \in D_i} c(\tau_i, x_T)$ (Figure 10), so a persuasive reply cannot offset a missing transaction and a correct endpoint cannot erase a forbidden intermediate action; undeclared checks are absent rather than passed, and per-check components are retained to localize failure. These six families instantiate three of the four facets of Section 1: *outcome* (target state and response), *trajectory* (actions, arguments, and ordering), and *policy compliance* (forbidden actions, forbidden or collateral state, and identity-gated disclosure); *operational viability* is reported alongside rather than folded into the conjunction. Only the response-content criteria are sent to a fixed LLM judge, which sees just the user instruction and the visible replies and cannot override the programmatic checks; the per-family check semantics and the full judge protocol are given in Appendix A. Expected-action checks match successfully executed calls; forbidden-action checks reject prohibited attempts, including calls rejected by the tool.

Reporting. The static capability-exposure audit measures graph-wide reachability of forbidden tools; turn-level exposure depends on routing. We report *scored coverage*, the fraction of scheduled tasks yielding a valid grade, alongside conditional success

$$\text{Pass}_{\text{scored}}^1 = \frac{\sum_{i \in G} Y_i}{|G|}, \quad (1)$$

where G excludes infrastructure failures but retains model-attributable failures such as hallucinated tools and human-verified non-convergence at timeout. Success counts are paired with $|G|/555$ and failure-category counts; latency and cost cover the tested agent model. We report along these four facets rather than collapsing them into a composite index: strict success and its per-check components for outcome and trajectory, latency and cost for operational viability, and the forbidden-action, forbidden-argument, and collateral-state checks for policy compliance. A system can look competitive on a single headline rate while differing sharply in its per-component pass rates or in cost, so tables and figures keep the component evidence and cost behind any headline rate.

3 Benchmark Construction

3.1 Task and Reference-Agent Construction

As shown in Figure 2, CRM-Agent-Bench is built around business workflows rather than isolated utterances or tool schemas. Each source workflow specifies an actor, a business goal, a stopping condition, the information needed to complete it, and both intended and prohibited effects, and is ported through a common pipeline into three separable artifacts: (i) a resettable domain environment of synthetic records, business policies, and deterministic tools, including structured business rejections; (ii) an architecture-neutral task definition binding a user goal and an optional initial-state overlay to an executable contract; and (iii) a graph-structured reference agent whose capability-specific nodes hold scoped instructions and tools. The reference graph is one fixed orchestration scaffold; the task and its contract do not require a tested agent to adopt it. Because all records and transactions are synthetic, the benchmark runs without a live vendor CRM system.

CRM-Agent-Bench contains 555 tasks drawn from two complementary sources. The first derives from five CRM applications developed and reviewed with practitioners and domain experts. The second adapts 22 tool using workflow families from a public, versioned collection of agent recipes (Salesforce Developers, 2026). The resulting tasks are organized into nine business domains and cover routine workflows, capability routing, multi step discovery, ambiguous requests, policy refusal, adversarial behavior, and collateral damage. Every task undergoes the same schema, tool reference, and fresh environment gold path validation. Here, schema refers to the benchmark task format rather than the agent’s tool interface. Validation confirms that the task is well formed, every referenced tool exists, and a reference solution can reach the contracted state from a fresh environment.

3.2 Workflow Taxonomy and Coverage

Within a domain, coverage is organized at four levels: *domain* (a shared mutable world and tool registry) $\rightarrow$ *capability* (a coverage tag, not an oracle route and not revealed to the agent) $\rightarrow$ *scenario* (a persona goal and stopping condition over seed records) $\rightarrow$ *executable contract*. Because a multi-turn task may traverse several nodes, this hierarchy describes coverage without predetermining the agent’s route, and grading is execution-first: every declared action resolves to a tool in the task’s domain, every state assertion addresses the runtime environment, and the grader consumes the recorded trace and final state rather than a post-hoc label.

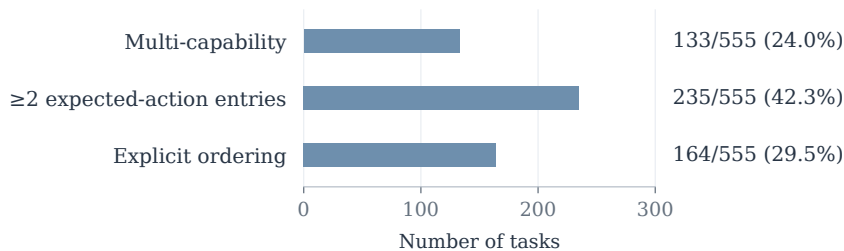


Figure 3: Multi-capability coverage and execution requirements in the 555-task evaluation set. Each bar counts tasks with the indicated property; percentages use all 555 tasks. The categories overlap and describe declared task requirements.

The nine domains carry 36 capability tags, and each task is labeled easy, medium, or hard by its authors as a descriptive stratum rather than an input to the agent or grader. The benchmark spans a per-domain distribution across difficulty together with a count of tasks carrying a negative (forbidden action, argument, or state) assertion. Coverage spans direct workflows, cross-capability routing, multi-hop discovery, ambiguity, policy refusal, adversarial requests, and collateral-damage cases, so a single contract may combine a valid update, an ordering constraint, a customer-facing explanation, and preservation of a protected record. We freeze task identifiers and configuration before model comparison so that all systems face the same workload and evaluator, following the matched-protocol principle of stateful interactive benchmarks (Yao et al., 2025; Barres et al., 2025). Figure 3 summarizes three overlapping properties of the task contracts: 133 tasks carry multiple capability tags, 235 declare at least two expected-action entries, and 164 impose explicit ordering constraints.

4 Results and Analysis

We evaluate sixteen systems spanning the Gemini, Claude, GPT, and open-weight families on the 555-task canonical set. Each run yields valid grades for all 555 tasks, with zero infrastructure exclusions, so strict accuracy (passed / 555) is directly comparable across models. The main leaderboard uses one execution per task, so we report descriptive comparisons rather than significance claims. In addition, we examine three-round reliability in Appendix C. We read the runs along the four facets of Section 1: the *outcome* an agent reaches, the *trajectory* it takes, its *operational viability*, and its *policy compliance*, and close by showing that these facets reorder systems that a single strict-accuracy number ranks alike.

4.1 Outcome: Task Success and Where It Concentrates

Figure 4 ranks all sixteen systems by strict accuracy; Figure 5 shows their domain profiles. Table 6 in Appendix B gives the numeric leaderboard results. Three runs (Inkling Small NVFP4, Muse Glimmer 30B, and Nemotron 3.5 Lightning 30B) use self-hosted open-weight checkpoints without metered per-token prices; their costs are reported as “n/a.”

Overall strict success. The sixteen strict-accuracy rates span a 19.28-point range, from 83.60% for Gemini 3.8 Flash down to 64.32% for Nemotron 3.5 Lightning 30B and GPT-4.1 Nano, which tie. The top group is tightly compressed: eight models (ranks 1 to 8) fall within a 2.88-point band (80.72% to 83.60%), and three of them (Claude Sonnet 5, GPT-5.5, and GPT-5.6 Sol) tie exactly at 81.26% (451/555), so a single headline number leaves the leading systems nearly indistinguishable. Strict success also declines monotonically from easy to hard tasks for every system, with a within-model easy-to-hard drop of 5.37 to 27.39 points (Table 7), the clearest headroom signal in the runs.

Three-run reliability. Across all 16 original leaderboard systems evaluated in all three rounds, Gemini 3.8 Flash ranks first in every round, with mean strict success of 85.53% (sample SD 1.66 percentage points). Claude Opus 5 averages 83.29% (SD 0.62 points) and remains second, including a tie with GPT-5.5 in round 2. Pairwise Spearman rank correlations range from 0.869 to 0.934, while nearby systems exchange positions. Task-level outcomes also vary: Gemini passes 480 tasks in both rounds 2 and 3, but 30 task outcomes switch between those executions. Across all three rounds, 434/555 tasks succeed every time ($\widehat{\text{pass}}^3 = 78.20\%$), compared with its 85.53%

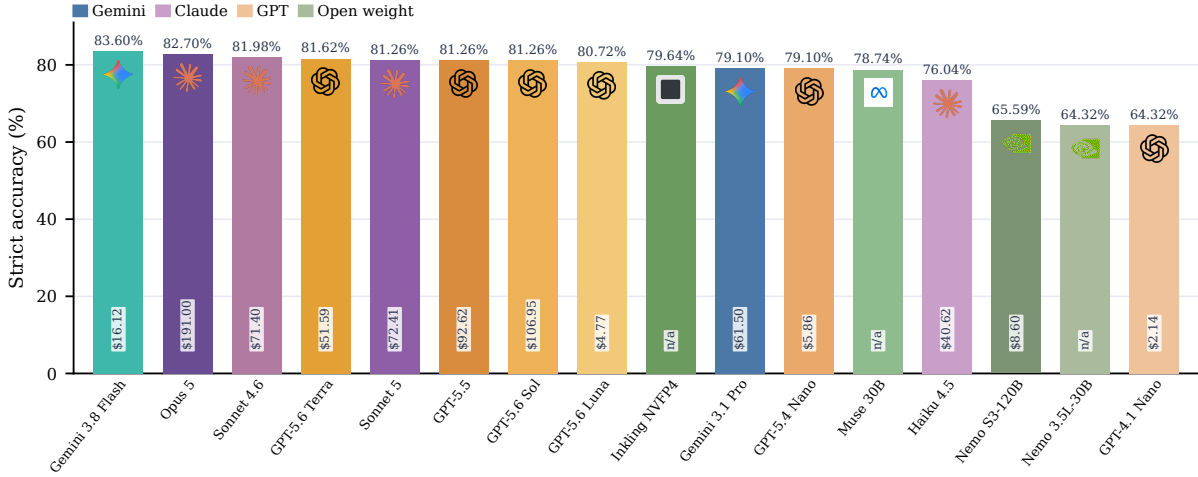


Figure 4: Sixteen full-suite runs on the audited 555-task canonical set, ranked by strict accuracy, colored by family and marked with their developer logos; strict-accuracy percentages and dollar-prefix estimated tested-agent costs are annotated per bar.

mean success rate. Appendix C gives the full three-round table, aggregation formulas, and task-level consistency results.

Domain variation. No model dominates every domain (Figure 5). Customer support is by far the hardest domain: its across-model mean strict success is 56.35%, 20.80 points below the next-hardest domain (travel, 77.15%), and every one of the sixteen models scores lower on customer support than on its own overall strict accuracy. Customer support is also the largest domain by task count (125/555). Sales and marketing have the smallest samples (18 and 21 tasks) and correspondingly the widest swings from a single flipped task; onboarding has the highest across-model mean strict success (90.24%).

Domain leadership does not track the overall ranking. Gemini 3.8 Flash, the overall leader, is the sole strongest model on retail (87/95) but leads no other domain: travel goes to Inkling Small NVFP4 (78/93) and ITSM to GPT-5.6 Luna (63/66). Muse Glimmer 30B is the sole strongest model on customer support (85/125, 68.00%) despite ranking 12th overall (78.74%), a domain leader that a coarser, aggregate-only comparison would miss. GPT-5.6 Sol is the only model to sweep sales (18/18) and ties for strongest on field service (23/23, with Claude Sonnet 4.6) and payments. The two lowest-ranked models, Nemotron 3.5 Lightning 30B and GPT-4.1 Nano, tie on overall strict accuracy (64.32%) but reach it differently: GPT-4.1 Nano is comparatively strong on retail (77/95, 81.05%) and weak on customer support (40/125, 32.00%, the single lowest domain score in the heatmap), while Nemotron 3.5 Lightning 30B is more evenly weak across domains, illustrating that the same overall score can hide different failure profiles.

4.2 Trajectory: A Valid Process, Not Just a Valid Answer

Grading only the final reply badly overstates task success. Conjoining the trajectory and state checks onto a response-only grade (which requires only the declared response check) removes a large and consistent slice of apparently successful tasks: for the four systems in Figure 6, a response-only score overestimates strict success by 10.5 to 11.0 percentage points, and the

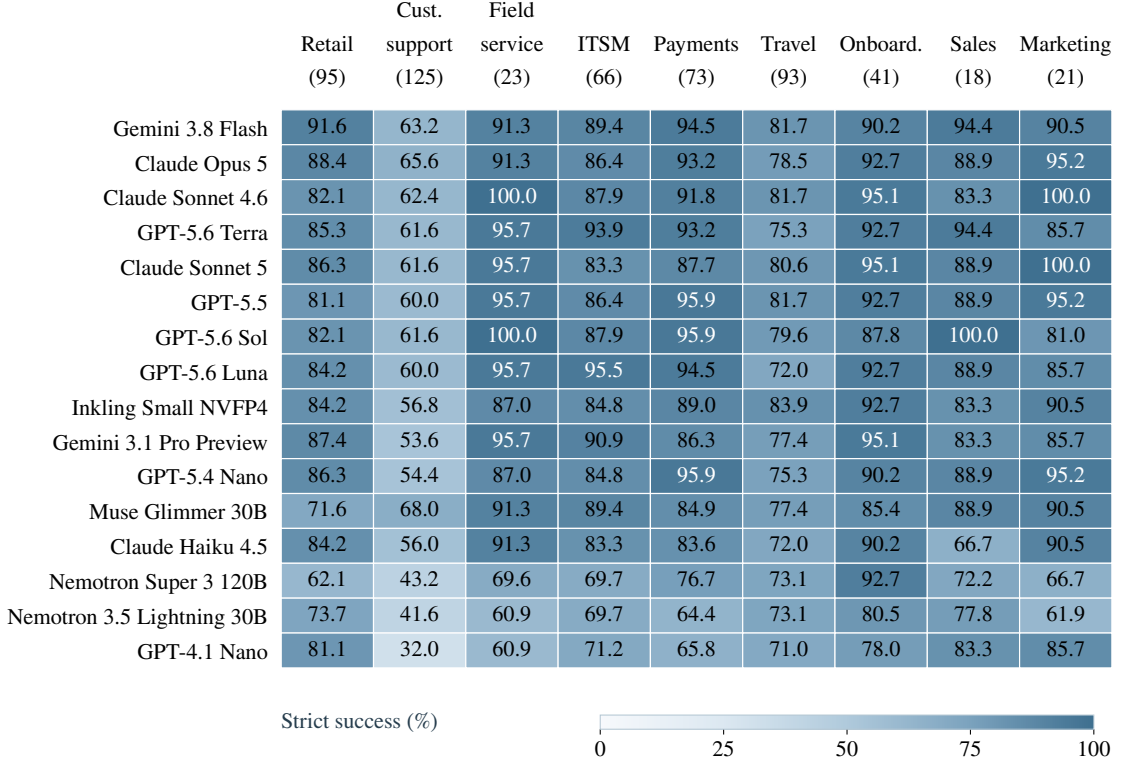


Figure 5: Domain leadership varies across the sixteen systems. Rows follow first-round overall strict-accuracy rank; each cell reports strict success (%) within a domain, whose task count is shown in the column header. All models use the same per-domain denominators.

action check alone removes 45 to 50 of the 555 tasks. The conjunctive contract is therefore not a cosmetic stricter threshold; it is what separates saying from doing.

The per-check grader view in the appendix (Figure 14) locates where trajectories break. Computed over each check’s recorded verdicts, semantic-response pass rates stay high, 92.3% for the strongest system and above 77% for every run, yet on the same runs the action check passes only 70.3% to 89.8% of applicable tasks and the argument check 70.2% to 90.7%: response-only success does not ensure correct execution. Under the primary-label precedence in Figure 13, action mismatch is the single largest non-success category for every one of the sixteen runs, from 54 tasks for the strongest system to a maximum of 157 for Nemotron Super 3 120B, while semantic-response, argument, and negative-check failures are individually much smaller. Trajectory validity remains a major constraint on these systems.

4.3 Operational Viability: Cost and Latency

The efficiency views in Figure 7 prevent accuracy differences from being read in isolation. Gemini 3.8 Flash has both the highest strict-success rate (83.60%) and the lowest mean tested-agent token count (24.0k per scored task), so it is the sole Pareto-optimal point in the token panel. The open-weight Nemotron 3.5 Lightning 30B has the lowest mean LLM latency (5.86 seconds) and Inkling Small NVFP4 the second-lowest (6.66 seconds), but they trail on accuracy (64.32% and 79.64%); Gemini 3.1 Pro Preview and Muse Glimmer 30B sit at the high-latency extreme (69.08 and 84.96 seconds). Tool-call budgets differ nearly four-fold: Nemotron Super 3 120B issues the

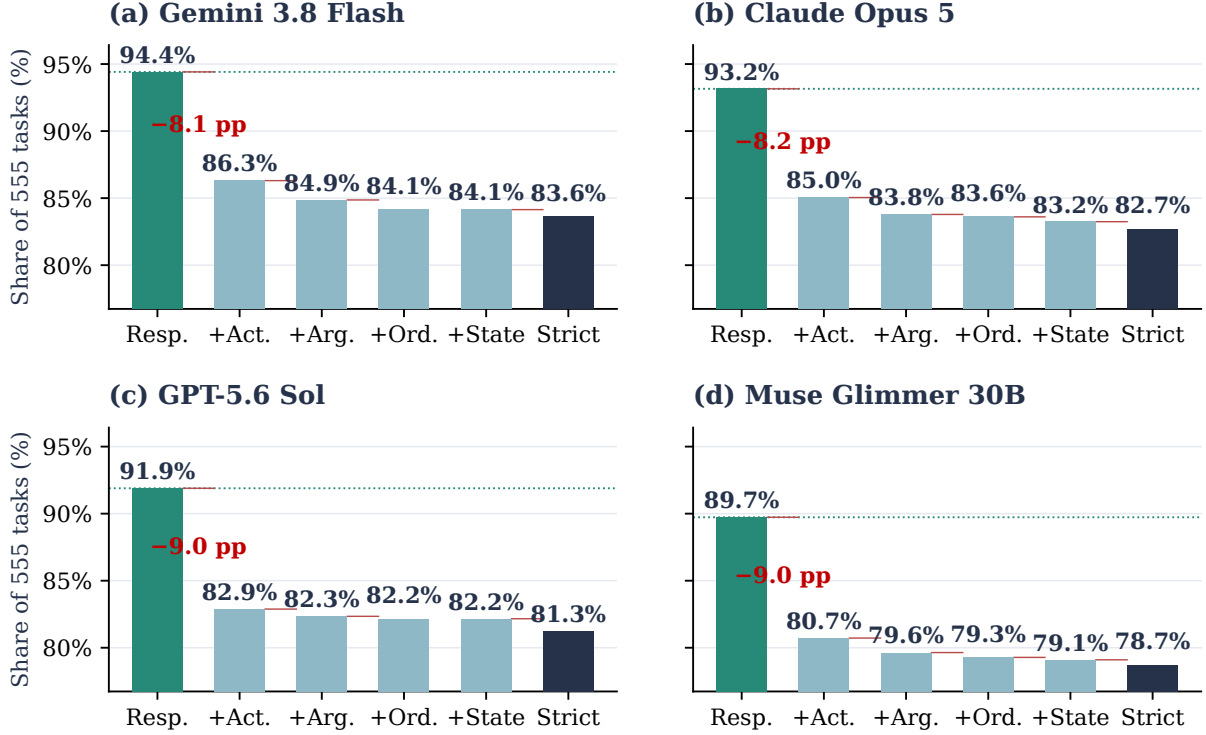


Figure 6: Success retained as execution checks are added to a response-only filter. Each panel uses all 555 tasks at every stage. Bars add action, argument, ordering, and state checks; the strict endpoint adds the remaining declared checks and model-attributable error handling. Undeclared or unavailable component verdicts are neutral at partial stages. Annotated drops are differences between the displayed percentages.

fewest calls per task (2.09) and Nemotron 3.5 Lightning the most (7.68). These are system-level descriptives; they do not isolate provider load, model architecture, or gateway implementation.

Cost. Estimated tested-agent cost varies far more than accuracy and does not track rank. Across the thirteen priced runs it spans 89.3 $\times$, from GPT-4.1 Nano at \$2.14 (tied for the lowest accuracy, 64.32%) to Claude Opus 5 at \$191.00 (only 2nd on accuracy). Gemini 3.8 Flash is the informative reference: it leads on accuracy (83.60%) at just \$16.12, the fifth-cheapest priced run, and no priced model costing as much or more scores higher; several pay more for less, e.g. Claude Haiku 4.5 costs \$24.50 above Gemini 3.8 Flash while scoring 7.56 points lower. The three unpriced runs are self-hosted open-weight checkpoints without a metered per-token price, so we do not rank them on cost. All thirteen priced entries are recomputed from published list pricing and observed token usage under one cache-neutral rule and exclude user-simulator and judge usage (Table 6).

4.4 Policy-Constrained Tasks: Compliance, Completion, and Exposure

The 555-task set splits into 390 tasks without a negative assertion and 165 *policy-constrained* tasks whose contract declares at least one forbidden-action, forbidden-argument, or protected-state check; this contract-defined subset is fixed across all sixteen runs. Figure 8 reports strict success on each subset and *negative-check compliance*: the share of policy-constrained tasks on which all applicable negative checks held, independent of the rest of the conjunctive contract. Across the 2,640 policy-constrained model-task trials, 2,408 (91.2%) pass all applicable negative

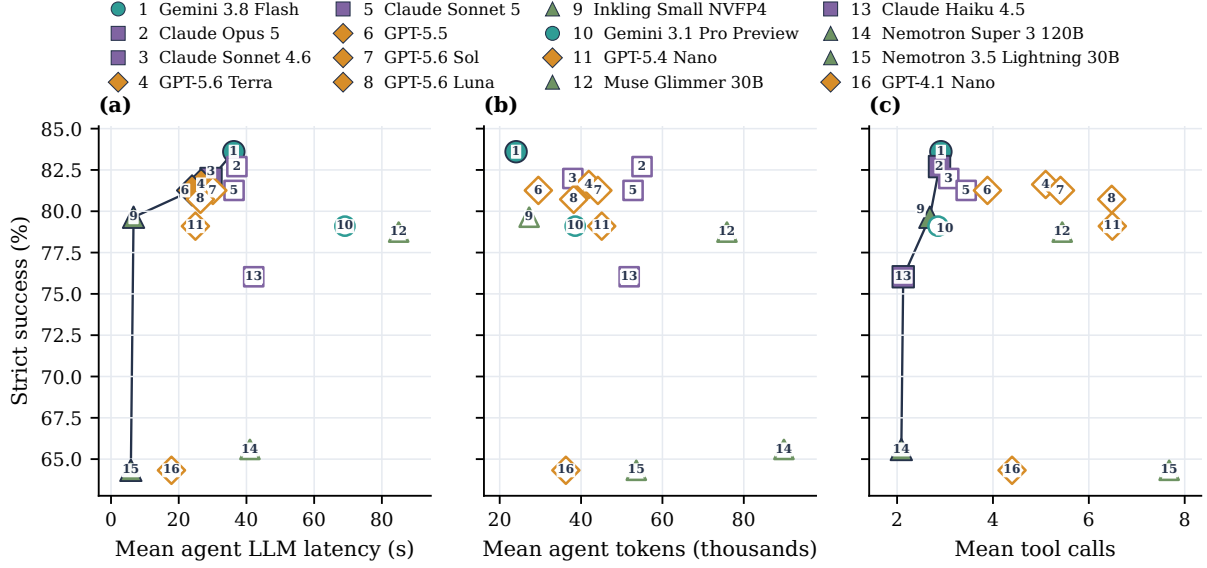


Figure 7: Efficiency and accuracy over the 555 scored tasks. Each panel plots strict success against a mean tested-agent efficiency measure: LLM latency, tokens, and tool calls (simulator and judge usage excluded). Dark segments trace each panel’s Pareto frontier (lower resource use with higher success preferred); filled markers lie on it, hollow markers are dominated. Marker IDs follow the Figure 4 rank order, and color and shape encode model family.

checks and 1,764 (66.8%) pass the full contract. Thus, 644 trials including 24.4% of the subset and 73.5% of its 876 strict failures pass the negative checks but fail another required component.

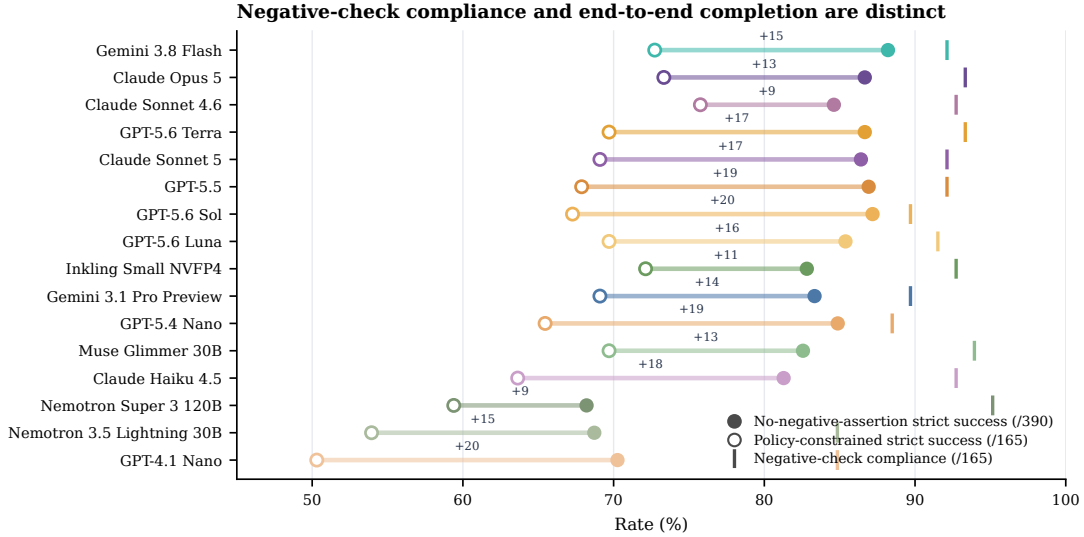


Figure 8: Compliance and completion on the task set, in Table 6 rank order. Filled circles: strict success on the 390 tasks without negative assertions; open circles: strict success on the 165 policy-constrained tasks; bars: their difference. Carets mark the share of policy-constrained tasks passing all applicable negative checks, and sit right of the open circles, so most failures on those tasks come from the accompanying required components rather than violations.

5 Related Work

Benchmark	On-policy user	Agent-mutated state	Required final effect	No-extra final effect	Forbidden trace event	Trace-order enforcement	Process-outcome hard gate	Self-contained environment
<i>Structured tool-use and enterprise agents</i>								
BFCL (Patil et al., 2025)	–	⦿	⦿	⦿	⦿	–	⦿	⦿
τ -bench (Yao et al., 2025)	✓	✓	✓	✓	–	–	–	✓
τ^2 -Bench (Barres et al., 2025)	✓	✓	✓	⦿	–	–	⦿	✓
ToolSandbox (Lu et al., 2025)	✓	✓	✓	✓	✓	✓	⦿	⦿
AppWorld (Trivedi et al., 2024)	–	✓	✓	✓	–	–	–	✓
CRMArena (Huang et al., 2025a)	–	–	–	–	–	–	–	–
CRMArena-Pro (Huang et al., 2025b)	✓	–	–	–	⦿	–	–	–
<i>Computer-use and GUI agents</i>								
WorkArena++ (Boisvert et al., 2024)	–	✓	✓	⦿	–	–	–	–
WebArena (Zhou et al., 2023)	–	✓	✓	⦿	–	–	–	✓
OSWorld (Xie et al., 2024)	–	✓	✓	⦿	–	–	–	⦿
AndroidWorld (Rawles et al., 2024)	–	✓	✓	⦿	–	–	–	✓
OS-Harm (Kuntz et al., 2025)	–	✓	✓	⦿	✓	–	–	⦿
CRM-Agent-Bench (ours)	✓	✓	✓	✓	✓	✓	✓	✓

Table 1: Comparison of representative agent benchmarks by evaluation contract and deployment setting. ✓ marks a native mechanism scored when applicable, ⦿ limited or indirect support, and – no primary support. The process-outcome gate distinguishes reaching the correct outcome from following the required procedure by requiring both declared checks to pass. *Self-contained* means that no live vendor backend is required.

Stateful, Enterprise, and Computer-Use Agent Benchmarks. Benchmarks for tool-using language models span increasingly complete execution settings, from function selection and argument generation in AgentBench and BFCL (Liu et al., 2024; Patil et al., 2025) to stateful, multi-turn interaction: τ -bench pairs an on-policy simulated user with domain policies and a mutable database, τ^2 -Bench extends this to dual control, and ToolSandbox and AppWorld add stateful environments with trajectory-aware or programmatic evaluation (Yao et al., 2025; Barres et al., 2025; Lu et al., 2025; Trivedi et al., 2024). In the enterprise setting, WorkArena++ studies compositional knowledge work, and CRMArena and CRMArena-Pro contribute connected enterprise data, professional CRM tasks, and confidentiality-sensitive scenarios, with CRMArena-Pro separately scoring confidentiality-aware refusal (Boisvert et al., 2024; Huang et al., 2025a,b); unlike these, which primarily grade retrieval and answer correctness, CRM-Agent-Bench grades the executed state change and its policy compliance. The concurrent EnterpriseOps-Gym likewise targets stateful agentic planning and tool use in enterprise settings (Malay et al., 2026), while CRM-Agent-Bench is distinguished by a per-task conjunctive executable contract that jointly gates the required actions, ordering, resulting state, and prohibited effects rather than a single success signal. A parallel line of computer-use agents adds perception and low-level GUI control, with WebArena, AndroidWorld, and OSWorld evaluating functional completion over web, mobile, and desktop applications, and SCUBA extending this line to computer use over Salesforce enterprise interfaces (Zhou et al., 2023; Rawles et al., 2024; Xie et al., 2024; Dai et al., 2026), and safety benchmarks such as OS-Harm, BLIND-ACT, and OSGuard showing that task completion alone is insufficient (Kuntz et al., 2025; Shayegani et al., 2025; Mohammadmirzaei and Flanigan, 2026). CRM-Agent-Bench studies the complementary structured-tool setting: it removes visual grounding as a confound so that workflow reasoning, routing, and capability exposure can be measured directly.

Execution Grounded Evaluation and Simulated Users. The benchmarks above rely on complementary evidence from tool calls, intermediate milestones, and final application state. No single source is sufficient to establish valid execution, and AgentRewardBench further shows

that automatic trajectory evaluators often disagree (Lù et al., 2025). TemporalBench evaluates contextual and event-informed reasoning beyond numerical forecasting accuracy (Weng et al., 2026). Simulated users introduce an additional measurement variable. SimulatorArena and RealUserSim examine how faithfully simulated behavior reproduces human conversations (Dou et al., 2025; Zhu et al., 2026). PersonaCUABench varies user personas on fixed computer-use tasks and evaluates task completion alongside satisfaction ratings provided by simulated users (Li et al., 2026). CRM-Agent-Bench uses a task specific conjunctive contract over actions, arguments, ordering, response content, target state, and protected state, while retaining component results for diagnosis.

Capability Exposure, and Policy Attribution. Research on tool retrieval asks how an agent selects a small relevant set from a large tool corpus, and tool-graph benchmarks study selection and dependency reasoning among interconnected tools (Shi et al., 2025b; Guo et al., 2026). A related literature evaluates agents that operate over untrusted data or receive harmful requests: AgentDojo measures prompt-injection attacks and defenses (Debenedetti et al., 2024), Progent enforces fine-grained privilege policies over tool calls, and ToolPrivBench studies unnecessary escalation to higher-privilege tools (Shi et al., 2025a; Yang et al., 2026). Recent validity analysis further warns that measured agent safety can be confounded with general task capability (Wang et al., 2026). This distinction is central in our setting: a forbidden action may be absent because the model rejected it, because the model failed to act, or because routing never exposed the required tool on the relevant turn.

6 Conclusion

We presented CRM-Agent-Bench, an execution-grounded benchmark of 555 stateful, multi-turn enterprise tasks across nine domains whose heterogeneous conjunctive contracts measure satisfaction of task-declared outcome, trajectory, and policy requirements alongside operational viability. Across sixteen systems, strict success ranges from 64.32% to 83.60%, yet systems with similar accuracy diverge sharply in cost, per-domain reliability, and difficulty scaling. These results support a multi-axis approach to agent selection in enterprise software as a service (*SaaS*), weighing execution cost, per-domain reliability, and policy-constrained completion alongside strict accuracy. For *agent* design, the findings support optimizing and reporting valid process and policy-constrained completion jointly, with tool exposure and routing treated as measurable design choices.

Acknowledgments

We sincerely thank Indira Iyer, Bhavesh Doshi, Daniel Tisher, Huan Wang, and Shelby Heinecke for their guidance and support throughout this project. We are particularly grateful for their help in making research resources available to support the development and evaluation of CRM-Agent-Bench. Their support played an important role in bringing this work to completion.

References

Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik Narasimhan. τ^2 -bench: Evaluating conversational agents in a dual-control environment. *arXiv preprint arXiv:2506.07982*,

2025. URL <https://arxiv.org/abs/2506.07982>.

Léo Boisvert, Megh Thakkar, Maxime Gasse, Massimo Caccia, Thibault Le Sellier De Chezelles, Quentin Cappart, Nicolas Chapados, Alexandre Lacoste, and Alexandre Drouin. Workarena++: Towards compositional planning and reasoning-based common knowledge work tasks. In *Advances in Neural Information Processing Systems*, volume 37, pages 5996–6051, 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/0b82662b6c32e887bb252a74d8cb2d5e-Abstract-Datasets_and_Benchmarks_Track.html.

Yutong Dai, Krithika Ramakrishnan, Jing Gu, Matthew Fernandez, Yanqi Luo, Viraj Prabhu, Zhenyu Hu, Silvio Savarese, Caiming Xiong, Zeyuan Chen, and Ran Xu. SCUBA: Sales-force computer use benchmark. In C. Vondrick, B. Hariharan, C. Raffel, L. Pinto, D. Yang, and A. Faust, editors, *International Conference on Learning Representations*, volume 2026, pages 24363–24386, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/29c8c615b3187ee995029284702d3f43-Paper-Conference.pdf.

Edoardo Debenedetti, Jie Zhang, Mislav Balunović, Luca Beurer-Kellner, Marc Fischer, and Florian Tramèr. AgentDojo: A dynamic environment to evaluate prompt injection attacks and defenses for LLM agents. In *Advances in Neural Information Processing Systems*, volume 37, 2024. URL <https://arxiv.org/abs/2406.13352>.

Yao Dou, Michel Galley, Baolin Peng, Chris Kedzie, Weixin Cai, Alan Ritter, Chris Quirk, Wei Xu, and Jianfeng Gao. SimulatorArena: Are user simulators reliable proxies for multi-turn evaluation of AI assistants? *arXiv preprint arXiv:2510.05444*, 2025. URL <https://arxiv.org/abs/2510.05444>.

Xiaojie Guo, Yang Zhang, Bing Zhang, Ryo Kawahara, Mikio Takeuchi, and Yada Zhu. UniTool-Bench: A benchmark for tool-augmented LLMs in cross-domain, universal task automation. In *Findings of the Association for Computational Linguistics: EACL 2026*, pages 4726–4736, 2026. doi: 10.18653/v1/2026.findings-eacl.248. URL <https://aclanthology.org/2026.findings-eacl.248/>.

Kung-Hsiang Huang, Akshara Prabhakar, Sidharth Dhawan, Yixin Mao, Huan Wang, Silvio Savarese, Caiming Xiong, Philippe Laban, and Chien-Sheng Wu. Crmarena: Understanding the capacity of llm agents to perform professional crm tasks in realistic environments. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics*, pages 3830–3850, 2025a. doi: 10.18653/v1/2025.naacl-long.194. URL <https://aclanthology.org/2025.naacl-long.194/>.

Kung-Hsiang Huang, Akshara Prabhakar, Onkar Thorat, Divyansh Agarwal, Prafulla Kumar Choubey, Yixin Mao, Silvio Savarese, Caiming Xiong, and Chien-Sheng Wu. Crmarena-pro: Holistic assessment of llm agents across diverse business scenarios and interactions. *arXiv preprint arXiv:2505.18878*, 2025b. URL <https://arxiv.org/abs/2505.18878>.

Thomas Kuntz, Agatha Duzan, Hao Zhao, Francesco Croce, Zico Kolter, Nicolas Flammarion, and Maksym Andriushchenko. OS-Harm: A benchmark for measuring safety of computer use agents. *arXiv preprint arXiv:2506.14866*, 2025. URL <https://arxiv.org/abs/2506.14866>.

- Haoming Li, Defu Cao, Xu Hu, Yingtao Luo, Yiyang Wang, Xianghe Xu, Jiashu He, Wanting Jiang, Yifan Zhang, Yunhan Zhao, Siyang Wu, Chen Yang, Haixin Wang, Zheyuan Deng, Taoran Li, Jingwen Zhang, Zhejian Peng, Shushu Liang, Yue Zhao, Bingzhe Li, Feng Chen, Yifan Wang, Jintao Huang, Yijun Wang, Yuexing Hao, and Xiaomin Li. PersonaCUABench: Benchmarking computer-use agents serving diverse personas. *arXiv preprint arXiv:2610.22990*, 2026. URL <https://arxiv.org/abs/2610.22990>.
- 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. In *International Conference on Learning Representations*, 2024. URL <https://openreview.net/forum?id=zAdUB0aCTQ>.
- Jiarui Lu, Thomas Holleis, Yizhe Zhang, Bernhard Aumayer, Feng Nan, Haoping Bai, Shuang Ma, Shen Ma, Mengyu Li, Guoli Yin, Zirui Wang, and Ruoming Pang. Toolsandbox: A stateful, conversational, interactive evaluation benchmark for llm tool use capabilities. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pages 1160–1183, 2025. doi: 10.18653/v1/2025.findings-naacl.65. URL <https://aclanthology.org/2025.findings-naacl.65/>.
- Xing Han Lù, Amirhossein Kazemnejad, Nicholas Meade, Arkil Patel, Dongchan Shin, Alejandra Zambrano, Karolina Stańczak, Peter Shaw, Christopher J. Pal, and Siva Reddy. AgentRewardBench: Evaluating automatic evaluations of web agent trajectories. *arXiv preprint arXiv:2504.08942*, 2025. URL <https://arxiv.org/abs/2504.08942>.
- Shiva Krishna Reddy Malay, Shravan Nayak, Sagar Davasam, Srinivas Sunkara, and Sai Rajeswar. EnterpriseOps-Gym: Environments and evaluations for stateful agentic planning and tool use in enterprise settings. In *Forty-third International Conference on Machine Learning*, 2026. URL <https://openreview.net/forum?id=QKy6cKEiEB>.
- Mina Mohammadmirzaei and Jeffrey Flanigan. OSGuard: A benchmark for safety in computer-use agents. *arXiv preprint arXiv:2606.15034*, 2026. URL <https://arxiv.org/abs/2606.15034>.
- Shishir G. Patil, Huanzhi Mao, Fanjia Yan, Charlie Cheng-Jie Ji, Vishnu Suresh, Ion Stoica, and Joseph E. Gonzalez. The berkeley function calling leaderboard (BFCL): From tool use to agentic evaluation of large language models. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pages 48371–48392, 2025. URL <https://proceedings.mlr.press/v267/patil25a.html>.
- Christopher Rawles, Sarah Clinckemaiellie, 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. *arXiv preprint arXiv:2405.14573*, 2024. URL <https://arxiv.org/abs/2405.14573>.
- Salesforce Developers. Agent script recipes. GitHub repository, 2026. URL <https://github.com/trailheadapps/agent-script-recipes>. Accessed 2026-09-24.
- Erfan Shayegani, Keegan Hines, Yue Dong, Nael Abu-Ghazaleh, Roman Lutz, Spencer Whitehead,

- Vidhisha Balachandran, Besmira Nushi, and Vibhav Vineet. Just do it!? computer-use agents exhibit blind goal-directedness. *arXiv preprint arXiv:2510.01670*, 2025. URL <https://arxiv.org/abs/2510.01670>.
- Tianneng Shi, Jingxuan He, Zhun Wang, Hongwei Li, Linyu Wu, Wenbo Guo, and Dawn Song. Progent: Securing AI agents with privilege control. *arXiv preprint arXiv:2504.11703*, 2025a. URL <https://arxiv.org/abs/2504.11703>.
- Zhengliang Shi, Yuhan Wang, Lingyong Yan, Pengjie Ren, Shuaiqiang Wang, Dawei Yin, and Zhaochun Ren. Retrieval models aren’t tool-savvy: Benchmarking tool retrieval for large language models. In *Findings of the Association for Computational Linguistics: ACL 2025*, 2025b. URL <https://aclanthology.org/2025.findings-acl.1258/>.
- Harsh Trivedi, Tushar Khot, Mareike Hartmann, Ruskin Manku, Vinty Dong, Edward Li, Shashank Gupta, Ashish Sabharwal, and Niranjana Balasubramanian. Appworld: A controllable world of apps and people for benchmarking interactive coding agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*, pages 16022–16076, 2024. doi: 10.18653/v1/2024.acl-long.850. URL <https://aclanthology.org/2024.acl-long.850/>.
- Youting Wang, Xiao Han, Dingyan Shang, Yuan Tang, and Bowen Liu. Safety, or just capability? a validity audit of agent-safety benchmarks. *arXiv preprint arXiv:2607.28685*, 2026. URL <https://arxiv.org/abs/2607.28685>.
- Muyan Weng, Defu Cao, Wei Yang, Yashaswi Sharma, and Yan Liu. TemporalBench: A benchmark for evaluating LLM-based agents on contextual and event-informed time series tasks. In *Proceedings of the 32nd ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 2*, pages 9997–10008, 2026. URL <https://arxiv.org/abs/2602.13272>.
- Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, Yitao Liu, Yiheng Xu, Shuyan Zhou, Silvio Savarese, Caiming Xiong, Victor Zhong, and Tao Yu. OSWorld: Benchmarking multimodal agents for open-ended tasks in real computer environments. *arXiv preprint arXiv:2404.07972*, 2024. URL <https://arxiv.org/abs/2404.07972>.
- Kaiyue Yang, Yuyan Bu, Jingwei Yi, Yuchi Wang, Biyu Zhou, Juntao Dai, Songlin Hu, and Yaodong Yang. When lower privileges suffice: Investigating over-privileged tool selection in LLM agents. *arXiv preprint arXiv:2606.20023*, 2026. URL <https://arxiv.org/abs/2606.20023>.
- Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=WE_vluYUL-X.
- Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ -bench: A benchmark for tool-agent-user interaction in real-world domains. In *International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/1b126cc38b8638e07bef37e7b2bb72bf-Abstract-Conference.html.
- 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. *arXiv preprint arXiv:2307.13854*, 2023. URL <https://arxiv.org/abs/2307.13854>.

Ming Zhu, Juntao Tan, Rithesh Murthy, Jieliu Qiu, Liangwei Yang, Wenting Zhao, Silvio Savarese, Shelby Heinecke, and Huan Wang. RealUserSim: Bridging the reality gap in agent benchmarking via grounded user simulation. *arXiv preprint arXiv:2605.20204*, 2026. URL <https://arxiv.org/abs/2605.20204>.

Appendix

A	Evaluation Protocol and Benchmark Construction Details	20
A.1	Formal Problem Definition and Evidence Boundary	20
A.2	Executable Contracts and Grading Semantics	22
A.3	Frozen Evaluation Protocol and Reporting	23
A.4	Task Construction, Provenance, and Validation	24
A.5	Extension Interface and Release Workflow	25
A.6	Evaluation-Set Composition and Coverage	26
A.7	Reference Scaffold and Routing	26
A.8	Domain Realization Profiles	29
B	Aggregate Diagnostics for the 16 Completed Runs	29
B.1	Full-Suite Leaderboard	29
B.2	Failure Composition	31
B.3	Grader Diagnostic Matrix	32
B.4	Success by Domain	33
B.5	Success by Difficulty Stratum	33
B.6	Per-Metric Standing	34
B.7	Metric-Dependent Ranking	34
B.8	Policy-Constrained Tasks: Compliance versus Completion	35
C	Three-Run Reliability and Ranking Stability	36
C.1	Repeated Evaluation and Aggregation	36
C.2	Aggregate and Ranking Stability	37
C.3	Consistency on Individual Tasks	37
D	Qualitative Trace Evidence and Provenance	38
D.1	Case A: Recovery after a Structured Business Rejection	39
D.2	Case B: Plausible Communication, Duplicate Execution	39
D.3	Case C: Timeout from an On-Policy Retry Loop	40

D.4 Trace Artifact Mapping and Provenance	41
E Construct Validity and Benchmark Scope	41
E.1 Long-Horizon, Dependency-Laden Execution	41
E.2 Stateful Business Workflows	42
E.3 Probabilistic, On-Policy Dynamics	42
F Per-Domain System Flow Atlas	42

Scope and ongoing extensions. CRM-Agent-Bench prioritizes controlled, reproducible evaluation of headless, structured-tool workflows. Appendix C reports three evaluation rounds for all sixteen leaderboard systems. We are actively extending this setting through further controlled repetitions, human and domain-expert validation, and broader document-rich, visual, voice, and live-service workflows. These extensions will further test robustness, measurement sensitivity, and external validity while preserving executable, auditable grading.

A Evaluation Protocol and Benchmark Construction Details

This appendix expands the benchmark definition in Section 2 and the construction account in Section 3. Sections A.1–A.3 specify what constitutes evidence, how a verdict is computed, and how the 16 reported runs were executed; Sections A.4–A.6 document how the audited 555-task evaluation set was built, how the benchmark can be extended through versioned releases, and how the set is composed; Sections A.7–A.8 then record the reference scaffold, exposure analysis, and illustrative domain realizations. The simulator and semantic judge use the common model `gpt-5-mini`; Section A.3 distinguishes nominal settings from the execution settings preserved in the run manifests.

A.1 Formal Problem Definition and Evidence Boundary

POMDP formulation. For task i , the released specification $\mathcal{T}_i = (u_i, s_i^0, \mathcal{C}_i, d_i)$ parameterizes a finite-horizon partially observable Markov decision process (POMDP)

$$\mathcal{M}_i = (\mathcal{S}_i, \mathcal{A}_i, \mathcal{O}_i, P_i, \Omega_i, \rho_i, R_i, H).$$

The initial distribution ρ_i fixes the private user instruction u_i and initializes the CRM state as $x_0 = \text{merge}(x_{d_i}^{\text{default}}, s_i^0)$. An augmented latent state $\tilde{s}_t = (x_t, u_i, \ell_t, z_t)$ contains the mutable CRM state x_t , the accumulated dialogue–route–tool log ℓ_t , and the remaining simulator and harness state z_t . Including ℓ_t makes the terminal trajectory objective Markov even though the contract depends on the path, not only on x_T .

At a primitive event, the agent emits either a typed tool call or a user-visible message; one visible dialogue turn can therefore contain several tool calls before the reply. The observation kernel Ω_i reveals the visible dialogue, returned tool results, and the currently exposed instructions and tool schemas, but not the raw CRM state, u_i , or the evaluator-only contract $\mathcal{C}_i$. The transition kernel P_i combines programmatic, seed-controlled tool execution with the fixed on-policy user simulator. A tool may update x_t or return a structured business rejection without the requested state change. After a visible reply, the simulator emits the next user utterance or a stop decision conditioned on u_i and the visible dialogue; later observations therefore depend on earlier agent replies rather than on a fixed transcript. The interaction ends at the stop decision or the executor limits in Table 3.

An orchestration condition η induces, possibly stochastically, an admissible tool mask $M_\eta(h_t) \subseteq \mathcal{A}_{\text{tool}}$ from visible history h_t , so the actions available at event t are

$$\mathcal{A}_t^\eta = \mathcal{A}_{\text{msg}} \cup M_\eta(h_t).$$

The reported reference scaffold obtains this mask from a model-based per-turn router. Changing the router or exposed catalog changes the access policy, not the task initialization or contract. No intermediate reward is exposed to the agent. For a gradeable terminal trajectory τ_i , the

contract induces the ex-post score

$$R_i(\tilde{s}_T) = Y_i(\tau_i, x_T) = \prod_{c \in D_i} c(\tau_i, x_T), \quad c(\tau_i, x_T) \in \{0, 1\},$$

where D_i contains exactly the checks declared by $\mathcal{C}_i$. Thus a standard reward embedding assigns zero before termination and Y_i at the augmented terminal state; this is a benchmark score, not a training signal. A model-attributable execution failure, such as a hallucinated tool or verified non-convergence at timeout, sets $Y_i = 0$. By contrast, infrastructure failures receive $\perp$ and are excluded from the scored denominator under the reporting rule in Section A.3.

Operational evidence boundary. A trial begins by loading one task contract and resetting the paired CRM environment to its declared initial state. The on-policy simulator sees only the agent’s user-visible replies and generates the next user turn; it never sees private reasoning, tool calls, tool results, routes, or database state. On every user turn, the reference scaffold selects one capability node or a deterministic refusal, and the selected node acts through its scoped tools. When the interaction stops or reaches a limit, the harness freezes four evidence channels—visible dialogue, the ordered route log, the complete tool trace, and terminal state—and applies only the checks declared by that task.

Figure 9 expands the compact setup in Figure 2(a). It repeats the same task-contract $\rightarrow$ on-policy-user $\leftrightarrow$ graph-routed-agent $\leftrightarrow$ fresh-CRM-state $\rightarrow$ conjunctive-grader backbone, then opens the blocks to expose the frozen domain specification, per-turn access policy, evidence channels, and diagnostic readouts.

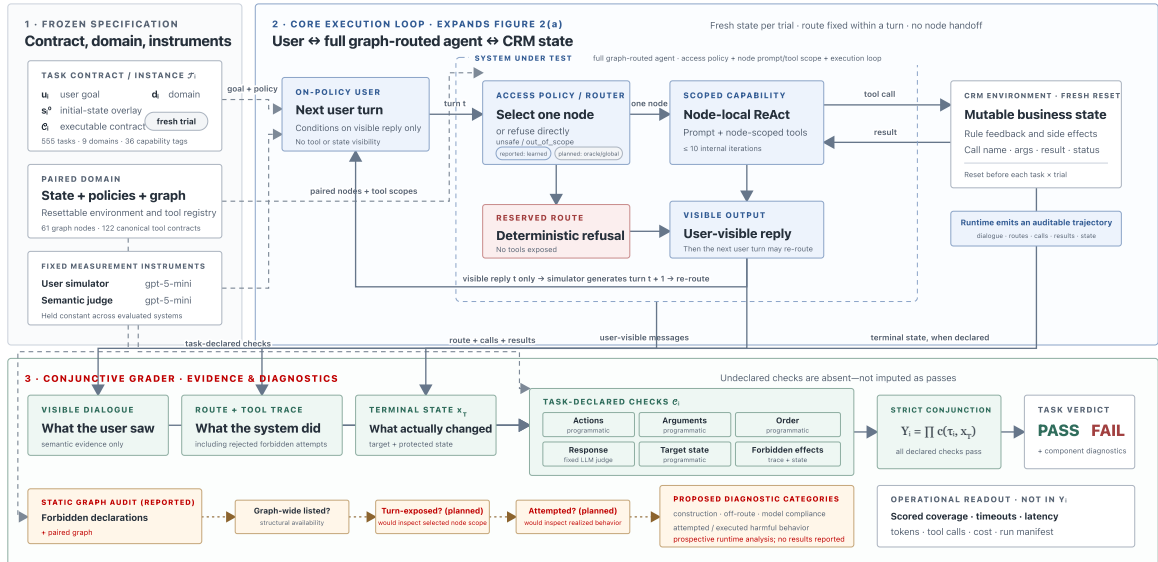


Figure 9: Detailed expansion of the audited evaluation setup in Figure 2(a). The task contract, on-policy user, graph-routed agent, freshly reset CRM environment, and conjunctive grader form the shared backbone. This appendix view opens those blocks to show the frozen domain specification, one-node-per-turn access policy, evidence channels, task-declared checks, the capability-exposure audit, and operational readouts.

A.2 Executable Contracts and Grading Semantics

Contract check families. Each task declares only the checks semantically applicable to its goal. The six top-level families are expected actions, expected arguments, action ordering, semantic response criteria, expected final state, and forbidden evidence (actions, arguments, or state). Expected actions use count-aware matching over successfully executed calls; argument constraints use either any matching successful call or only the first successful call. Ordering uses subsequence matching or an exact filtered sequence over the tools named by the constraint, so unrelated calls may interleave unless separately forbidden. State checks address sparse paths and support exact values, non-null requirements, and minimum collection lengths. Forbidden checks reject any attempted prohibited action or argument pattern and assert protected non-effects in terminal state. Task success is the strict conjunction of every declared check, while undeclared checks are absent rather than automatically passed. This combines the trajectory sensitivity of milestones and minefields with the endpoint rigor of programmatic state tests (Lu et al., 2025; Trivedi et al., 2024).

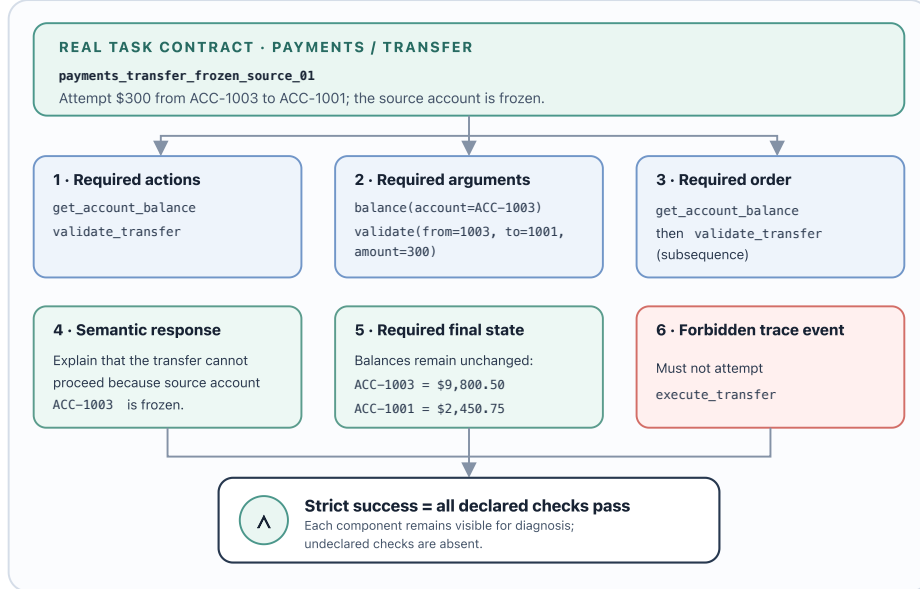


Figure 10: Anatomy of a task contract, shown for the real transfer task `payments_transfer_frozen_source_01`. Its required actions, arguments, ordered subsequence, semantic explanation, unchanged balances, and forbidden transfer attempt must all pass; component results remain visible to localize a failure.

Tool and state evidence. Tools implement the environment transition function. Every registered-tool attempt that reaches the execution layer records its name, arguments, returned value, execution status, and turn index. A well-formed call that reaches a tool can return a structured business rejection (for example, an authorization or conflict response); this is evidence that the action was *attempted*, not that the requested state change *occurred*. Where available, malformed-call and executor-error signals are retained as trajectory diagnostics and never count as successful state changes. The separate trace and final-state graders therefore distinguish attempted behavior from persistent business effects.

Semantic response evaluation. For a task with semantic criteria, the judge receives only the user instruction, the ordered criteria, and the user-visible assistant messages. Tool traces, tool results, routes, hidden reasoning, and final state are withheld, and the transcript is marked as untrusted data so that a reply cannot redefine the rubric. For each criterion the judge returns a binary decision, an exact evidence span in the visible transcript, and an explanation; a separate transcript-level flag records contradiction. A local validator enforces the JSON schema and verifies that cited evidence actually occurs; every criterion must pass and the global contradiction flag must remain false. Programmatic checks remain authoritative wherever structured evidence exists, so the semantic judge cannot override an action, ordering, or state failure.

Release-level contract inventory. Table 2 reports both task incidence and the number of declared contract objects. Rows overlap because one task can combine trace, communication, and state requirements. For state contracts, one declaration is a flattened leaf path in the sparse expected or forbidden state specification; for argument contracts, it is one task–tool specification. Full-suite strict accuracy measures satisfaction of task-specific contracts. State assertions occur in 200 of the 555 tasks, including protected-state assertions in 45, and cover the paths declared in those contracts.

Contract component	Tasks	Declarations	Declaration unit
Expected actions	529	887	action entry
Expected arguments	493	648	task–tool specification
Explicit ordering	164	419	ordered slot
Semantic response	402	465	criterion
Expected state	180	248	flattened leaf assertion
Forbidden actions	107	206	action entry
Forbidden arguments	31	49	task–tool specification
Forbidden state	45	49	flattened leaf assertion

Table 2: Executable-contract inventory over the audited 555-task evaluation set. Task counts are incidence counts rather than a partition: 200 tasks contain an expected or forbidden state check, and 165 contain at least one negative assertion of any type.

A.3 Frozen Evaluation Protocol and Reporting

Table 3 summarizes the nominal settings of the formal first-round protocol, which supplies the unified evaluation standard.

Run manifest and validation. Run manifests and per-task trace artifacts retain recorded benchmark and dataset versions, task and configuration paths, model and provider identifiers, sampling settings, executor limits, and simulator and judge configuration fields. The first-round archive preserves each extracted field’s presence or absence. Invalid judge output and judge-service failures are classified as infrastructure failures and excluded from the scored denominator.

Component	Frozen configuration
User simulator	<code>gpt-5-mini</code> ; temperature 0; low reasoning effort; <code>react</code> strategy; 1,086-token generation budget; 90-second step timeout; at most 4,000 output characters; up to three structured-output attempts including repair attempts; <code>###STOP###</code> stop signal.
Semantic judge	<code>gpt-5-mini</code> ; temperature 0; low reasoning effort; 16,384-token generation budget; at most three attempts; prompt <code>semantic-communication-v6</code> ; cache keyed by the complete judge input and configuration.
Executor	One trial per task; fixed base seed; agent temperature 0 where supported; low reasoning effort; 16,384-token agent budget; 240-second agent-step timeout; 30-turn limit; 1,200-second task timeout.

Table 3: Nominal settings of the formal first-round evaluation protocol for the 16 original leaderboard runs. The first-round protocol supplies the unified standard for the repeated-run comparison.

Reporting conventions. Model-attributable failures such as hallucinated tools and human-verified non-convergence at timeout remain in the scored denominator G as failures, whereas infrastructure failures are excluded. All sixteen runs are evaluated on the same frozen 555-task canonical set under the formal first-round standard, and each produced a valid grade on all 555 tasks; hence $|G| = 555$ for every reported run, with no infrastructure exclusions. Success counts are paired with failure-category counts over that fixed denominator. Latency and cache-neutral estimated cost cover the tested agent model only and exclude simulator and judge usage. The cost comparison charges all prompt tokens at the ordinary input rate and all completion tokens at the output rate, irrespective of cache status. The original leaderboard uses one trial per task with a fixed framework base seed and therefore constitutes a single-run benchmark snapshot. Appendix C reports three directly comparable rounds for all sixteen systems, including the formal first-round scores. It gives equal-round means, sample standard deviations, ranking stability, and the all-success pass³ reliability metric used by τ -bench (Yao et al., 2025).

A.4 Task Construction, Provenance, and Validation

Construction strata. CRM-Agent-Bench draws on two construction lineages: task instances descended from the original five CRM applications, and task instances adapted from 22 substantive tool-using workflow recipes in a public, versioned collection (Salesforce Developers, 2026). Historical names predate the final domain taxonomy: Store Manager maps to Retail, Customer Service maps to Customer Support, and Sales Development Representative (SDR) maps to Sales. Recipe-derived commerce and support workflows were merged with the corresponding seed domains, while external-integration workflows were distributed according to the business state they affect.

Portable encoding. For either lineage, a candidate scenario was admitted only when its actor, goal, stopping condition, required information, policy constraints, intended effects, and prohibited effects could be represented in the portable harness. Source records became resettable domain state or task-specific overlays. Source actions became typed tools over that state, including structured business rejections for valid but disallowed requests. Topics or recipe subagents

	Retail	Customer support	Field service	IT service mgmt.	Payments	Travel	Onboarding	Sales	Marketing	Total
Tasks	95	125	23	66	73	93	41	18	21	555
Lineage	Hybrid	Hybrid	Seed	Recipe	Recipe	Recipe	Recipe	Seed	Seed	–

Table 4: Construction provenance by domain in the audited evaluation set. In the table, *Seed* denotes tasks descended from the original five CRM applications; *Recipe* denotes tasks adapted from substantive tool-using workflow recipes; *Hybrid* domains draw on both. These labels describe construction lineage, not separate evaluation splits or required agent architectures.

became specialist nodes in the optional reference graph. The user journey became an instruction for the on-policy simulator, while correctness became a sparse contract over observable trace, communication, and final-state evidence.

A.5 Extension Interface and Release Workflow

CRM-Agent-Bench separates benchmark content from the agent and execution engine, exposing three extension surfaces with different implementation costs. First, a new business scenario inside an existing domain is a declarative YAML record in `src/tasks/<domain>.yaml`. At authoring time this requires no new environment, runner, or grader logic; admission to a canonical release still passes the gates below. The record supplies a unique identifier, an on-policy user instruction with a stopping condition, domain and capability metadata, a difficulty label, and an optional initial-state overlay. Its executable contract declares only the applicable expected and forbidden checks over actions, arguments, ordering, semantic response, and state. Unasserted dimensions are not graded. Second, a new agent or orchestration design can reuse the same task catalog and tool world by supplying a graph configuration with a system prompt, entry node, specialist-node instructions, and scoped tools. The evaluator does not grade graph shape, node names, route labels, or prompt wording, so this change does not alter the task contract. Third, a workflow requiring new actions or persistent records adds a registered `ToolEnvironment`: its `default_state()` defines the resettable world and `register_tools()` implements typed operations over that state. Tool schemas are derived from Python type signatures, and target- and protected-state assertions are evaluated against the resulting environment state. A wholly new domain additionally contributes a policy, task catalog, reference graph, environment registration, and versioned manifest entry; it does not require a new runner or grader.

Admission to a canonical release. Admission to a comparable release follows four steps:

1. **Specify the job.** Define the actor, business goal, stopping condition, required information, intended effects, prohibited effects, and the smallest contract that makes those requirements executable.
2. **Bind the evidence.** Check that every asserted action names a registered tool, argument assertions match its typed schema, state paths originate in the resettable domain state, and, when evaluating the reference agent, its configuration exposes the intended tools. The reference graph is an example scaffold, not a gold route that a tested agent must follow.

3. **Validate from a fresh world.** Run structural validation for unique IDs, YAML schema, tool references, state roots, contract consistency, and agent configuration. Then execute the architecture-independent gold path from a fresh environment and run the relevant unit tests plus an end-to-end domain spot check. The gold path establishes executability of the positive path and its target state; semantic and negative checks still require their focused tests. Refusal tasks may have an empty action path, but their negative contract is validated structurally.
4. **Freeze a new version.** Update the manifest version and release invariants, regenerate the content-addressed lock, rerun all gold paths and the full test suite, and only then evaluate systems on the enlarged set.

The repository exposes these release gates directly:

```
uv run python -m harness.benchmark_spec -gold -write-lock
uv run python -m harness.benchmark_spec -gold
uv run pytest
```

A.6 Evaluation-Set Composition and Coverage

The audited evaluation set contains 555 tasks and 767 task–tag assignments: 422 tasks have one capability tag and 133 have multiple tags (73 with two, 41 with three, and 19 with four). The 36 globally unique tag strings instantiate 38 domain–capability pairs. Figure 11 assigns each task to its first tag to obtain a mutually exclusive composition view; all tags remain available for coverage analysis.

A.7 Reference Scaffold and Routing

Graph-routed reference agent. The reference scaffold pairs each domain with a directed capability graph. At the start of every agent turn, a classifier based on the tested model maps the current user utterance and dialogue context to either one capability node or a reserved `unsafe/out_of_scope` label that returns a deterministic refusal. A selected node receives a domain prompt, its node-specific instructions, and only its assigned tools, then runs a local ReAct-style reason-act-observe loop (Yao et al., 2023) for at most ten internal iterations before the visible reply. The route is fixed within a turn (no node-to-node handoffs), and the classifier is invoked again only after the next user message. A trial may therefore traverse several capabilities across turns while never exposing the domain’s full tool inventory at once. Figure 12 makes this per-turn routing and scoped tool exposure concrete for the payments domain.

Domain graph inventory. Table 5 records the exact audited scaffold inventory. Each domain has one tool-free general fallback, so the *Specialists* row is one below the *Nodes* row in every domain. *Placements* counts graph-local tool assignments, whereas *Tools* counts unique canonical contracts.

Capability exposure audit. We audit `forbidden_actions` in all 555 task contracts by exact identifier matching against the union of tools assigned to the paired domain graph’s nodes. Each

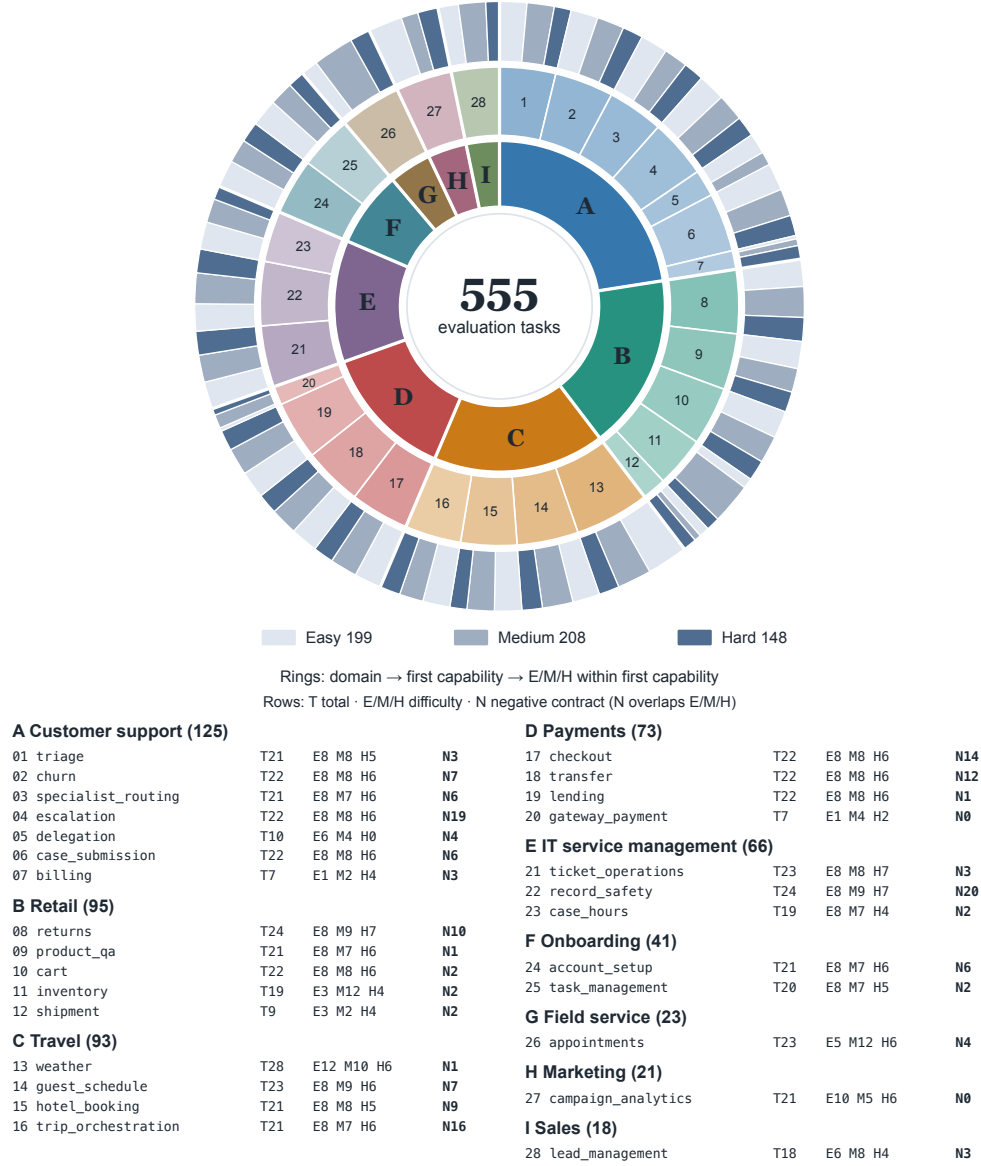


Figure 11: Composition of the audited 555-task evaluation set: domain (inner) → mutually exclusive first capability (middle) → easy/medium/hard within capability (outer). Each keyed row reports its total and E/M/H counts; N is the overlapping count of tasks with a forbidden action, argument, or protected-state check, not a fourth partition.

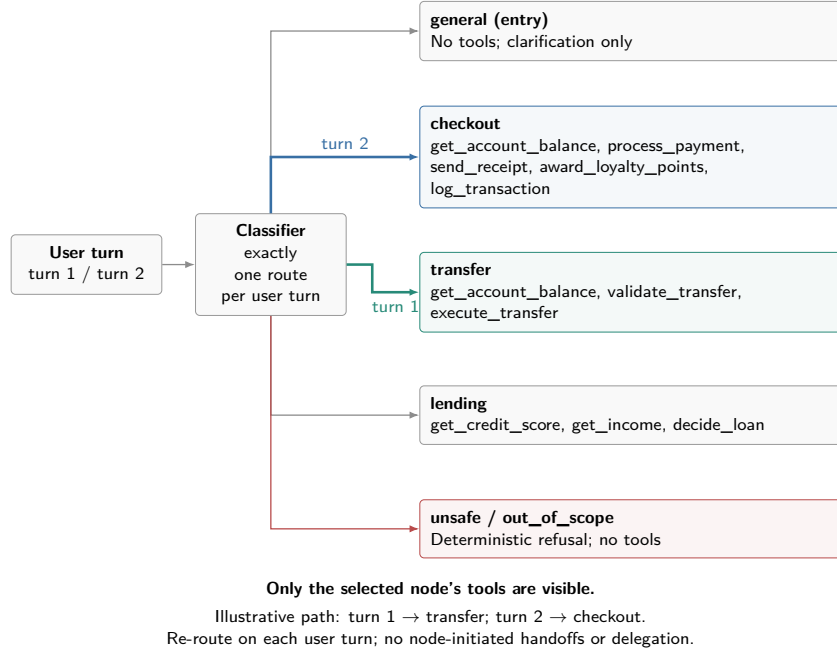


Figure 12: Graph-routed reference scaffold for the payments domain. A per-turn classifier selects exactly one capability node (exposing only that node’s scoped tools) or returns a deterministic **unsafe/out_of_scope** refusal, with no node-initiated handoffs. The highlighted path (turn 1 → transfer, turn 2 → checkout) shows how a trial traverses several capabilities across turns without exposing the full tool inventory at once.

	Retail	Customer support	Field service	IT service mgmt.	Payments	Travel	Onboarding	Sales	Marketing	Total
Task-tag pairs	7	8	3	3	5	5	2	3	2	38
Nodes	9	12	5	4	6	10	3	8	4	61
Specialists	8	11	4	3	5	9	2	7	3	52
Tools	20	25	12	12	11	13	8	11	10	122
Placements	25	37	14	18	13	16	8	15	10	156
Forbidden-action tasks	8	34	0	13	24	24	4	0	0	107
Forbidden-action entries	8	66	0	15	38	68	11	0	0	206

Table 5: Capability and graph inventory by domain for the reference scaffold used in the reported runs. The 38 domain–capability pairs correspond to 36 globally unique capability-tag strings because one tag is shared across three domains. The final two rows count tasks with forbidden-action declarations and the declarations themselves; every declaration matches a tool in its paired domain graph.

list entry is one declaration: a tool forbidden in two tasks counts twice, regardless of how many nodes contain it. Forbidden arguments and protected-state assertions are separate contract components (Table 2).

All 206 declarations across 107 tasks match paired graph tools, with zero unmatched entries (Table 5). The audit script, `evidence/audit_capability_exposure.py`, produces task-level results in `evidence/capability_exposure_audit.json`, recording the pinned source revision, input-file hashes, and matching nodes.

Graph-wide availability means that a forbidden tool belongs to at least one domain node. Turn-level exposure depends on the router’s selected node (Figure 12). For example, Payments balance-only tasks require `get_account_balance` while forbidding `process_payment`. Both tools belong to the checkout node; `process_payment` is exposed on a turn only if that node is selected.

A.8 Domain Realization Profiles

Representative construction profiles. The generic construction pipeline in Section A.4 becomes concrete in the per-domain gallery of Appendix F. Three profiles span the construction lineages: Customer Support (Figure 20) shows a hybrid urgent-billing journey; Payments (Figure 23) shows a recipe-derived checkout chain with ordered state changes; and Field Service (Figure 21) shows a seed-workflow appointment reschedule with ownership and scheduling gates. The full gallery (Figures 19–27) covers all nine domains and illustrates nodes, scoped tools, policy gates, state transitions, and contract observables. Appendix F describes the diagram scope and cross-turn tool lists; Table 5 reports the audited scaffold inventory.

Qualitative-trace provenance. The illustrative four-verdict trajectory in the main text (Figure 1) and the case traces in Appendices D and D.4 are single-run qualitative examples.

B Aggregate Diagnostics for the 16 Completed Runs

The headline strict-success rate is reported over the full 555-task canonical set in Figure 4. Every one of the 16 completed runs yielded a strict binary verdict on all 555 canonical tasks with zero infrastructure exclusions, so scored coverage is 100% for each run and all-task yield coincides with strict success; neither is reported separately here. Two tasks in the entire matrix errored on a hallucinated tool call and provide no failed component verdict for primary attribution, but they are counted as model-attributable strict failures over the same 555 denominator, not as infrastructure exclusions. These strict verdicts are distinct from the component-level diagnostic rates below. This section decomposes the same 16 runs by failure type, grader behavior, domain, difficulty, and metric.

B.1 Full-Suite Leaderboard

Table 6 gives the full ranking summarized in Figure 4: strict accuracy, passed / 555, the domain macro, and cache-neutral estimated tested-agent cost for all sixteen runs. The usage inputs are

sums over each run’s nine canonical domain-level `traces.json` files, after excluding smoke data and merging the split retail shards where present.

Rank	Model	Strict accuracy	Passed / 555	Domain macro	Normalized cost
1	 Gemini 3.8 Flash	83.60%	464	87.43%	\$16.12
2	 Claude Opus 5	82.70%	459	86.68%	\$191.00
3	 Claude Sonnet 4.6	81.98%	455	87.15%	\$71.40
4	 GPT-5.6 Terra	81.62%	453	86.41%	\$51.59
5=	 Claude Sonnet 5	81.26%	451	86.58%	\$72.41
5=	 GPT-5.5	81.26%	451	86.39%	\$92.62
5=	 GPT-5.6 Sol	81.26%	451	86.20%	\$106.95
8	 GPT-5.6 Luna	80.72%	448	85.46%	\$4.77
9	 Inkling Small NVFP4	79.64%	442	83.58%	n/a
10=	 Gemini 3.1 Pro Preview	79.10%	439	83.94%	\$61.50
10=	 GPT-5.4 Nano	79.10%	439	84.23%	\$5.86
12	 Muse Glimmer 30B	78.74%	437	83.04%	n/a
13	 Claude Haiku 4.5	76.04%	422	79.76%	\$40.62
14	 Nemotron Super 3 120B	65.59%	364	69.55%	\$8.60
15=	 Nemotron 3.5 Lightning 30B	64.32%	357	67.06%	n/a
15=	 GPT-4.1 Nano	64.32%	357	69.88%	\$2.14

Table 6: Full-suite strict accuracy on the audited 555-task canonical set, ranked by strict accuracy with ties on passed/555 marked “=”. Domain macro is the unweighted mean of the nine per-domain strict-success rates and is shown for reference but does not determine the rank order. Normalized cost is a cache-neutral estimate for the tested agent and excludes simulator and judge usage. We charge every observed prompt token at the provider’s ordinary input rate and every completion token at its output rate, regardless of whether the run used a prompt cache. This applies the same accounting rule to every model but does not estimate the realized bill for a cache-enabled run. Usage is summed from the nine canonical domain-level `traces.json` files for each run (list prices accessed 2026-09-25). Gemini 3.1 Pro Preview uses the ≤ 200 k-token-prompt tier; Gemini 3.8 Flash uses its introductory global price through 2026-12-31; GPT-5.6 Sol uses promotional pricing disclosed as valid at least through 2026-11-21. The three Bedrock Claude runs use the US geo rate identified by their `us.anthropic` model IDs. Remaining “n/a” entries are self-hosted open-weight runs without a metered per-token price, not zero-cost runs.

B.2 Failure Composition

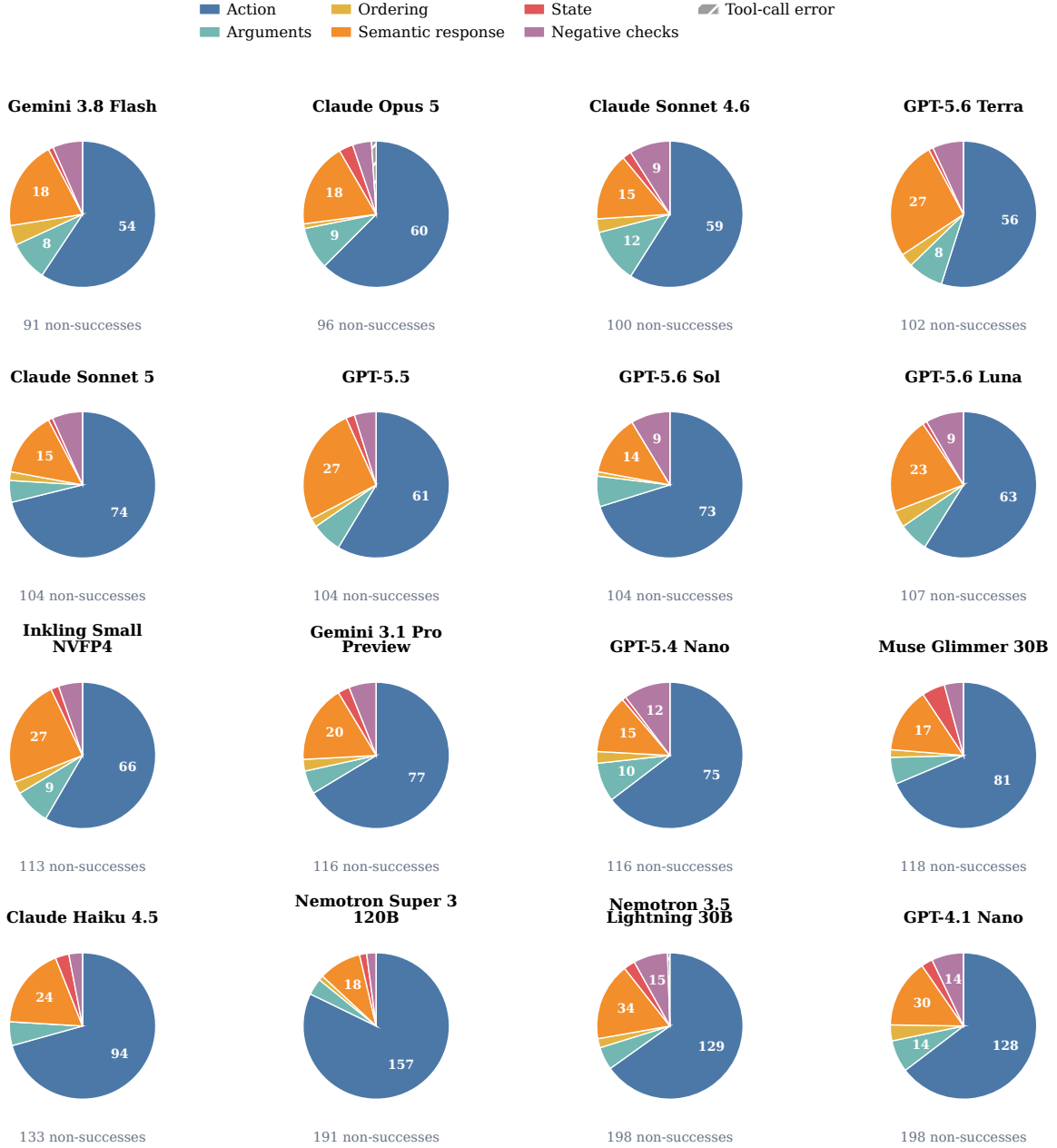


Figure 13: Composition of all scored tasks that did not become strict successes for each of the 16 runs (of 555 scored). Each pie corresponds to one model, and its subtitle gives the total number of non-successes. To make the slices mutually exclusive, each non-success receives one primary label using the harness status precedence: action, forbidden, state, arguments, semantic response, ordering, then collateral state. “Negative checks” combines the forbidden-action/argument and collateral-state labels. The hatched “Tool-call error” segment contains two tasks that errored (e.g., on a hallucinated tool call) without a failed component verdict available for primary attribution. Both remain model-attributable strict failures, and there are zero timeouts or infrastructure exclusions. In-slice labels are task counts, not percentages, and are shown only when space permits. Models are ordered by total non-success count.

Action mismatch is the largest non-success category for every completed run, ranging from 54 tasks (Gemini 3.8 Flash) to 157 tasks (Nemotron Super 3 120B) across all 16 runs under the exclusive attribution rule. The remaining categories are much smaller and vary by run: semantic-response-primary failures span 14–34, argument-primary 5–14, and the combined negative checks 4–15. Because every run scored all 555 canonical tasks, the failure pies contain no timeout or infrastructure segment; the only non-grader outcomes are two errored tasks (one each for Claude Opus 5 and Nemotron 3.5 Lightning 30B, both hallucinated tool calls). Among the 16 runs, total non-success counts run from 91 (Gemini 3.8 Flash) to 198 (Nemotron 3.5 Lightning 30B and GPT-4.1 Nano). Figure 14 separately retains the overlapping per-grader view that a primary-cause stack necessarily suppresses.

B.3 Grader Diagnostic Matrix

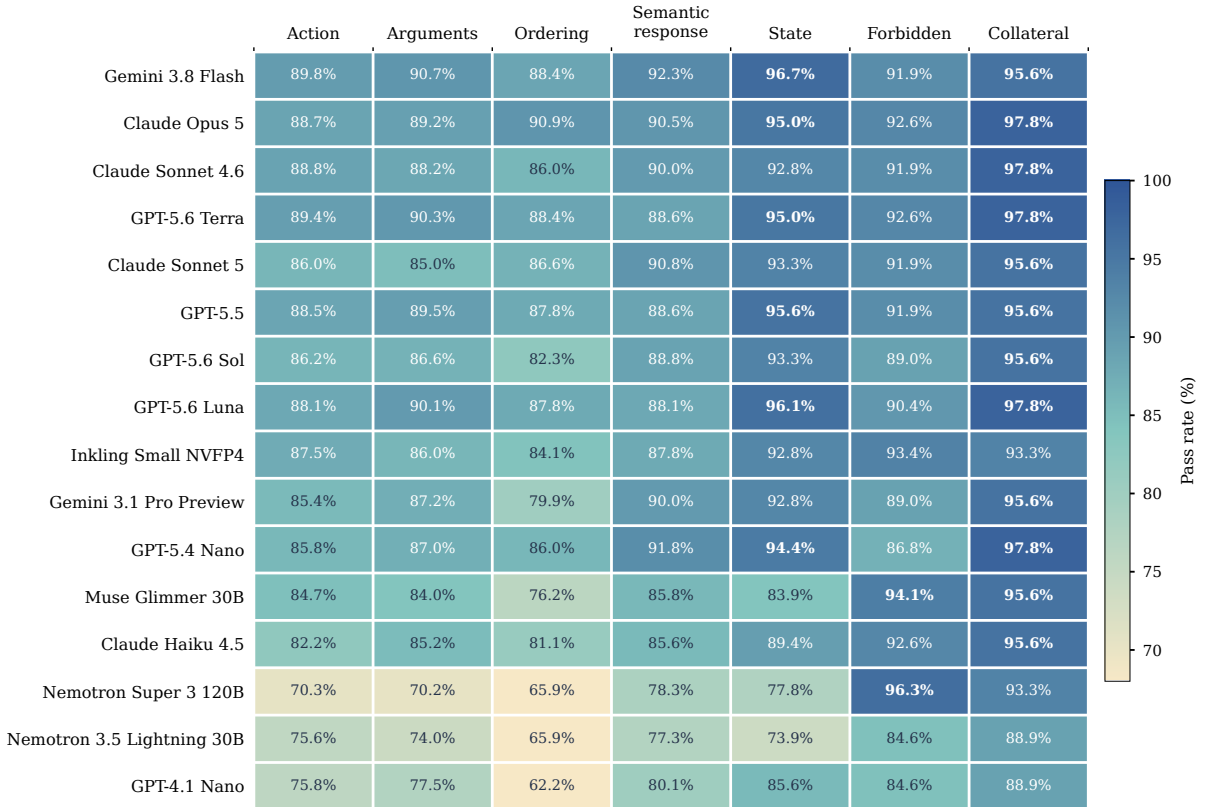


Figure 14: Component-grader pass rates across the 16 runs, each computed over that component’s recorded verdicts. Denominators are 529 (action), 493 (arguments), 164 (ordering), 180 (state), 136 (forbidden), and 45 (collateral). Semantic response has 402 recorded verdicts per run, except Opus (401), Nemotron Super (396), and Nemotron 3.5 (392). Undeclared checks and missing component verdicts are omitted from these rates. Columns therefore describe different task subsets. “Forbidden” aggregates forbidden-action and forbidden-argument checks; “Collateral” evaluates protected state.

The matrix uses each component’s recorded verdicts, omitting undeclared checks and unavailable verdicts. For Gemini 3.8 Flash, the matrix reports a semantic-response pass rate of $371/402 = 92.3\%$. The response-only stage in Figure 6 additionally retains 153 tasks without a response

criterion, giving $(371 + 153)/555 = 94.4\%$. The cascade treats undeclared or unavailable verdicts as neutral at partial stages and applies the error gate at the strict endpoint.

Across systems, recorded semantic-response pass rates exceed 77%, action rates range from 70.3–89.8%, argument rates from 70.2–90.7%, and state rates from 73.9–96.7%. These marginal rates summarize different subsets; the cumulative, same-task comparison in Figure 6 measures the effect of adding execution checks. The forbidden-action/argument (136 tasks) and protected-state (45 tasks) columns summarize the corresponding declared negative checks.

B.4 Success by Domain

Figure 5 in Section 4.1 reports per-domain strict-success rates, using the same domain denominators across all sixteen systems. Customer support is the single lowest-scoring domain for all 16 runs (across-model mean 56.35%, against 90.24% for onboarding, the highest across-domain mean), and no run is uniformly strong—the within-model gap between a run’s best and worst domain ranges from 23.3 points (Muse Glimmer 30B) to 53.7 points (GPT-4.1 Nano). The domain-leadership reshuffle these rates produce is discussed in Section 4.

B.5 Success by Difficulty Stratum

Model	Easy (199)	Medium (208)	Hard (148)
◆ Gemini 3.8 Flash	88.44% (176/199)	81.73% (170/208)	79.73% (118/148)
✱ Claude Opus 5	87.94% (175/199)	80.77% (168/208)	78.38% (116/148)
✱ Claude Sonnet 4.6	86.93% (173/199)	80.77% (168/208)	77.03% (114/148)
☼ GPT-5.6 Terra	84.42% (168/199)	81.25% (169/208)	78.38% (116/148)
✱ Claude Sonnet 5	84.42% (168/199)	79.81% (166/208)	79.05% (117/148)
☼ GPT-5.5	86.93% (173/199)	78.85% (164/208)	77.03% (114/148)
☼ GPT-5.6 Sol	86.93% (173/199)	80.77% (168/208)	74.32% (110/148)
☼ GPT-5.6 Luna	86.43% (172/199)	80.29% (167/208)	73.65% (109/148)
■ Inkling Small NVFP4	84.42% (168/199)	79.33% (165/208)	73.65% (109/148)
◆ Gemini 3.1 Pro Preview	86.43% (172/199)	75.96% (158/208)	73.65% (109/148)
☼ GPT-5.4 Nano	83.92% (167/199)	77.40% (161/208)	75.00% (111/148)
∞ Muse Glimmer 30B	81.91% (163/199)	77.88% (162/208)	75.68% (112/148)
✱ Claude Haiku 4.5	80.90% (161/199)	78.37% (163/208)	66.22% (98/148)
👁 Nemotron Super 3 120B	71.86% (143/199)	62.98% (131/208)	60.81% (90/148)
👁 Nemotron 3.5 Lightning 30B	73.87% (147/199)	65.38% (136/208)	50.00% (74/148)
☼ GPT-4.1 Nano	77.39% (154/199)	62.02% (129/208)	50.00% (74/148)

Table 7: Strict success within each difficulty stratum. Cells give the stratum rate with *successes/scored*. Scored denominators are identical across all 16 runs because every run scored all 555 canonical tasks; the column headers are the per-stratum sizes from Figure 11 (199 easy, 208 medium, 148 hard). Rows are ordered by overall strict accuracy. Difficulty labels are author-assigned descriptive strata and are not visible to the agent, simulator, or grader.

All 16 systems decline monotonically (non-increasing) from easy to hard tasks (Table 7). Easy-task success spans 71.86–88.44%, medium 62.02–81.73%, and hard 50.00–79.73%; the within-model

easy-to-hard drop ranges from 5.37 points (Claude Sonnet 5) to 27.39 points (GPT-4.1 Nano) and is the clearest headroom signal in the completed runs. System ordering is not preserved across strata: Gemini 3.8 Flash leads all three, but below it the ranking reshuffles—Claude Sonnet 5 is joint-fifth overall yet attains the second-highest hard-task rate (79.05%), ahead of the second- and third-ranked Claude Opus 5 and Sonnet 4.6, while GPT-5.6 Sol, also tied at 81.26% overall, slips to ninth on hard tasks (74.32%).

B.6 Per-Metric Standing

Figures 15 and 16 provide complementary views of metric sensitivity: the former preserves each metric’s absolute scale, whereas the latter shows how system order changes across metrics.

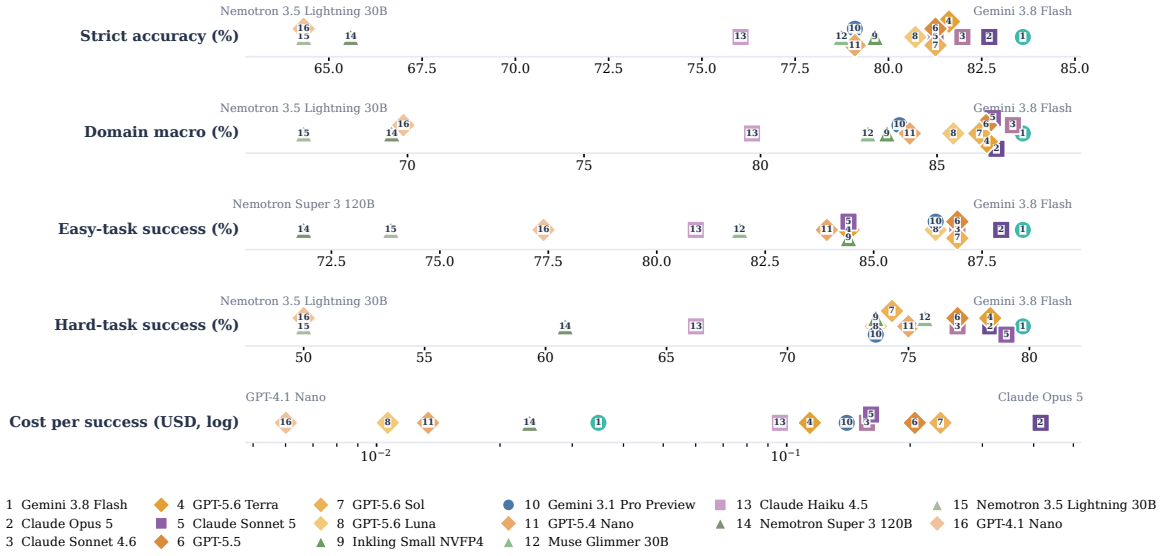


Figure 15: Per-metric standing of the 16 completed runs. Each row is an independent number line placing every run at its real value on that metric’s own scale. Marker IDs follow the Figure 4 rank order and map to the key below; small vertical offsets only separate colliding markers and do not encode value. Runs that share a headline strict-success number (three tie at 81.26%, with two further ties at 79.10% and 64.32%) nonetheless separate on one or more additional axes: domain macro, easy- and hard-task success, and a $69.4\times$ range in cache-neutral cost per success (log axis; shown for the 13 priced runs, from \$0.00599 for GPT-4.1 Nano to \$0.416 for Claude Opus 5). This per-metric view is the complement to the rank crossings in Figure 16.

B.7 Metric-Dependent Ranking

Figure 16 converts the same performance axes into per-metric ranks to make those ordering changes explicit. The ranking view complements overall scored success with metric-specific system comparisons.

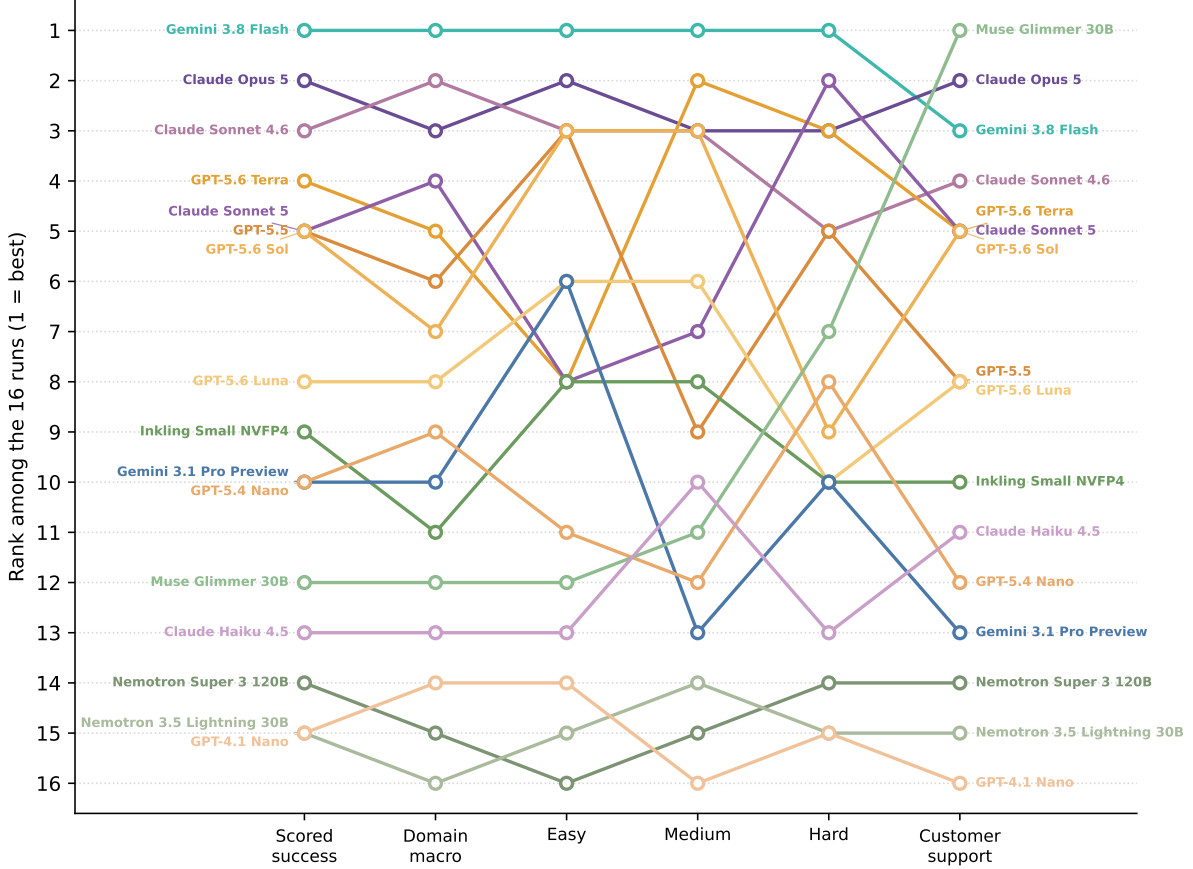


Figure 16: Rank of each completed run (1 = best of 16) across six higher-is-better metrics: scored success, domain macro, easy-, medium-, and hard-task success, and customer-support success. Shared competition ranks are used: equal values receive the same rank, and subsequent ranks advance by the number of tied systems. Endpoint labels are offset only for legibility; data points remain at their shared ranks. No system dominates every axis. Gemini 3.8 Flash leads five of the six metrics but falls to third on customer support, where Muse Glimmer 30B—twelfth on scored success—ranks first and Claude Opus 5 (second overall) places second; Claude Sonnet 5 climbs from joint-fifth to second on hard tasks. Ranks are computed from the same one-trial values used in Figure 4 and Table 7. Δ Cost per success is omitted here because three open-weight runs are unpriced; it is shown on its own log axis in Figure 15.

B.8 Policy-Constrained Tasks: Compliance versus Completion

This section gives the per-model numbers behind Figure 8. The canonical set contains 390 tasks without an explicit negative assertion and 165 policy-constrained tasks declaring at least one forbidden-action, forbidden-argument, or protected-state check. Because these declarations are fixed by the contracts, the partition is identical across all sixteen runs. For compactness, the table retains *positive* and *safety* as column labels for the two subsets. Both columns report strict success on the full conjunctive contract. Safety-check compliance is narrower: it reports the share of policy-constrained tasks on which all applicable negative checks held, regardless of the remaining contract components.

Section 4.4 reports the aggregate compliance and completion results. Nemotron Super 3 120B attains the highest observed negative-check compliance (95.2%) while ranking 14th overall.

Model	Positive success (390)	Safety success (165)	Δ	Safety-check compliance (165)
🔹 Gemini 3.8 Flash	88.2% (344/390)	72.7% (120/165)	+15.5	92.1% (152/165)
🌟 Claude Opus 5	86.7% (338/390)	73.3% (121/165)	+13.3	93.3% (154/165)
🌟 Claude Sonnet 4.6	84.6% (330/390)	75.8% (125/165)	+8.9	92.7% (153/165)
🌀 GPT-5.6 Terra	86.7% (338/390)	69.7% (115/165)	+17.0	93.3% (154/165)
🌟 Claude Sonnet 5	86.4% (337/390)	69.1% (114/165)	+17.3	92.1% (152/165)
🌀 GPT-5.5	86.9% (339/390)	67.9% (112/165)	+19.0	92.1% (152/165)
🌀 GPT-5.6 Sol	87.2% (340/390)	67.3% (111/165)	+19.9	89.7% (148/165)
🌀 GPT-5.6 Luna	85.4% (333/390)	69.7% (115/165)	+15.7	91.5% (151/165)
■ Inkling Small NVFP4	82.8% (323/390)	72.1% (119/165)	+10.7	92.7% (153/165)
🔹 Gemini 3.1 Pro Preview	83.3% (325/390)	69.1% (114/165)	+14.2	89.7% (148/165)
🌀 GPT-5.4 Nano	84.9% (331/390)	65.5% (108/165)	+19.4	88.5% (146/165)
∞ Muse Glimmer 30B	82.6% (322/390)	69.7% (115/165)	+12.9	93.9% (155/165)
🌟 Claude Haiku 4.5	81.3% (317/390)	63.6% (105/165)	+17.6	92.7% (153/165)
👁 Nemotron Super 3 120B	68.2% (266/390)	59.4% (98/165)	+8.8	95.2% (157/165)
👁 Nemotron 3.5 Lightning 30B	68.7% (268/390)	53.9% (89/165)	+14.8	84.8% (140/165)
🌀 GPT-4.1 Nano	70.3% (274/390)	50.3% (83/165)	+20.0	84.8% (140/165)
<i>Mean</i>	82.1%	66.8%	+15.3	91.2%

Table 8: Strict success partitioned by contract structure on the 555-task canonical set. “Positive” denotes the 390 tasks without explicit negative assertions; “Safety” denotes the 165 policy-constrained tasks with at least one such assertion. Safety-check compliance is the share of the latter on which all applicable negative checks held. Every model has lower strict success on the policy-constrained subset ($\Delta = +8.8$ to $+20.0$ points), while negative-check compliance is 84.8–95.2%. Rows follow Table 6.

Compliance and workflow completion are distinct measured axes under the fixed scaffold. These rates do not isolate intrinsic model safety or establish a causal safety-and-utility trade-off.

C Three-Run Reliability and Ranking Stability

C.1 Repeated Evaluation and Aggregation

The first-round leaderboard in Table 6 places Gemini 3.8 Flash ahead of Claude Opus 5 by only five of 555 tasks (0.90 percentage points). To assess whether this ordering persists across executions, we report three evaluation rounds: the formal first-round results and two additional runs. All three rounds are compared on the same 555 task contracts using the common strict-success definition. Strict success requires every applicable action, argument, state, semantic-response, ordering, and negative check to pass. Model-attributable errors count as failures; unscored slots follow the benchmark’s exclusion convention. Each round starts fresh task environments. The first round is the official result used in the main leaderboard.

For model m , let $a_{m,r}$ denote its strict-success rate over scored tasks in round r . We give the three rounds equal weight and report their mean and sample standard deviation:

$$\bar{a}_m = \frac{1}{3} \sum_{r=1}^3 a_{m,r}, \quad s_m = \sqrt{\frac{1}{2} \sum_{r=1}^3 (a_{m,r} - \bar{a}_m)^2}. \quad (2)$$

The standard deviation describes variation across the three executions and is reported in percentage points. This average is computed from round-level rates, rather than by pooling

attempts across rounds. We also report $\widehat{\text{pass}}^3$, the fraction of tasks successful in all three executions among tasks scored in all three, following the all-success reliability convention of τ -bench (Yao et al., 2025).

Model	Round 1	Round 2	Round 3	Mean $\pm$ SD	$\widehat{\text{pass}}^3$
◆ Gemini 3.8 Flash	464/555	480/555	480/555	85.53 $\pm$ 1.66	78.20
✱ Claude Opus 5	459/555	462/555	465/554	83.29 $\pm$ 0.62	73.47
⌘ GPT-5.5	451/555	462/555	465/555	82.76 $\pm$ 1.33	74.77
✱ Claude Sonnet 4.6	455/555	460/555	460/555	82.58 $\pm$ 0.52	75.14
⌘ GPT-5.6 Terra	453/555	447/555	462/555	81.80 $\pm$ 1.36	72.43
⌘ GPT-5.6 Sol	451/555	457/555	451/554	81.67 $\pm$ 0.59	73.65
◆ Gemini 3.1 Pro Preview	439/555	449/555	461/554	81.07 $\pm$ 2.06	73.83
✱ Claude Sonnet 5	451/555	445/555	441/555	80.30 $\pm$ 0.91	70.45
⌘ GPT-5.6 Luna	448/555	444/555	442/555	80.12 $\pm$ 0.55	70.63
⌘ GPT-5.4 Nano	439/555	435/555	451/555	79.58 $\pm$ 1.50	67.57
■ Inkling Small NVFP4	442/555	436/555	433/555	78.74 $\pm$ 0.83	68.47
∞ Muse Glimmer 30B	437/555	431/555	441/555	78.62 $\pm$ 0.91	67.93
✱ Claude Haiku 4.5	422/555	432/555	428/555	77.00 $\pm$ 0.91	68.65
👁 Nemotron Super 3 120B	364/555	382/555	375/555	67.33 $\pm$ 1.63	50.99
👁 Nemotron 3.5 Lightning 30B	357/555	363/555	363/555	65.05 $\pm$ 0.62	54.95
⌘ GPT-4.1 Nano	357/555	346/555	348/555	63.12 $\pm$ 1.06	52.43

Table 9: Three directly comparable evaluation rounds under the common scoring standard. Counts are successes/scored tasks; mean rates and $\widehat{\text{pass}}^3$ are percentages, and SD is in percentage points ($n = 3$, sample SD). Round 1 is the formal first-round result. All 16 systems are ordered by three-round mean. Each round schedules 555 tasks.

C.2 Aggregate and Ranking Stability

Across all 16 systems from the original leaderboard, Gemini 3.8 Flash ranks first in every round and by the three-round mean (85.53%, SD 1.66 points). Claude Opus 5 remains second in each round, including a tie with GPT-5.5 in round 2, and averages 83.29% (SD 0.62 points). The Gemini–Opus gaps are 0.90, 3.24, and 2.55 points across the three rounds; their mean gap is 2.23 points. Thus, the original five-task lead is preserved in both additional runs, and the three-run summary supports the observed ordering of these two systems.

The broader ordering is also consistent: pairwise Spearman rank correlations among all 16 systems are 0.922 for rounds 1–2, 0.869 for rounds 1–3, and 0.934 for rounds 2–3, using average ranks for ties. Sample SDs range from 0.52 to 2.06 percentage points. Exact positions among nearby systems nevertheless move: GPT-5.5 rises from joint fifth in round 1 to joint second and third, while Sonnet 4.6 moves from third to fourth and sixth. Sol leads Terra in round 2 but falls behind it in round 3; Inkling and Muse also reverse order between those rounds. These changes support reporting multi-run means alongside individual-round results when models have similar scores.

C.3 Consistency on Individual Tasks

Stable aggregate scores do not imply that the same tasks succeed each time. Gemini 3.8 Flash passes 480 tasks in both rounds 2 and 3, but 30 task outcomes switch between those executions. Across all three rounds, 434/555 tasks succeed every time ($\widehat{\text{pass}}^3 = 78.20\%$), compared with its

85.53% mean strict-success rate. For Opus, 407/554 jointly scored tasks pass all three rounds (73.47%). The all-three-success metric therefore complements the mean by measuring sustained task-level reliability. Taken together, the three-run results show a persistent lead across all sixteen leaderboard systems and broadly consistent rankings, with residual variation among close systems and on individual tasks.

D Qualitative Trace Evidence and Provenance

What is a “demo” in this appendix?

A **demo** is a compact, trace-grounded view of one benchmark trial. It connects four layers that aggregate scores hide: the task’s executable contract, the route and tool observations produced at runtime, the visible assistant behavior, and the task-declared grader vector.

This appendix presents three diagnostic single-run traces covering strict success, a graded failure, and a timeout. The cases explain execution mechanisms; failure-mode frequencies are reported in the aggregate diagnostics over the 555 tasks. Case C uses an earlier **grok-4.6** run with a 600-second task deadline, outside the 16-system leaderboard; the formal leaderboard protocol specifies 1,200 seconds (Appendix A.3), and its release contains no timeouts. Figure 17 defines the four-layer reading order, and Table 10 summarizes the selected cases and their diagnostic roles. Task and tool identifiers are reproduced from the benchmark. Customer names, account identifiers, and transaction records are synthetic; transaction identifiers are generated by the resettable benchmark environment. Excerpts condense repeated turns and preserve the recorded tool arguments, results, and grader outcomes.

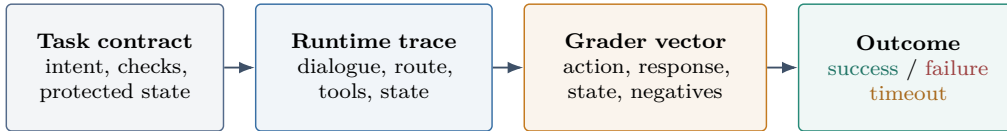


Figure 17: How to read a trace demo. Each card links the declared contract to runtime evidence and the individual grader decisions that produce the reported outcome.

Case	Model	Domain	Outcome	Diagnostic value
A	◆ Gemini 3.1 Pro Preview	Payments	SUCCESS	Recovery after a business-rule rejection
B	✱ Claude Sonnet 4.6	Payments	FAILURE	Duplicate side effect detected only in final state
C	🔗 Grok 4.6	Retail	TIMEOUT	On-policy interaction fails to converge

Table 10: Selected single-run traces and their diagnostic roles.

D.1 Case A: Recovery after a Structured Business Rejection

Demo A at a glance

STRICT SUCCESS

Model: Gemini 3.1 Pro Preview

Trace scale: 2 turns, 4 tool calls, 81.27 s

What to watch: the first amount is rejected by a business rule; the agent revises the plan without executing the forbidden \$1,000 transfer.

Source and contract. This trace comes from `gemini-3.1-pro-preview`, seed 997688223, task `payments_transfer_insufficient_then_retry_01`. The simulated user first requests a transfer of \$1,000 from ACC-1002 to ACC-1001; after an insufficient-funds response, the user corrects the amount to \$50. The contract requires two calls to `validate_transfer`, followed by `execute_transfer` for \$50. Executing the rejected \$1,000 amount is forbidden. The final balances must be \$130.00 and \$2,500.75, respectively.

Observed trajectory. The two-turn interaction took 81.27 seconds and remained in the `transfer__transfer` node. On the first turn, the agent called `get_account_balance` and observed a balance of \$180. It then called `validate_transfer(amount=1000)`, which returned `valid=false`, reason `Insufficient funds`, and status 422. The assistant accurately told the user that the transfer could not be completed. After the user replied, “Oh, sorry, I meant \$50,” the agent called `validate_transfer(amount=50)` and received `valid=true`; it then executed the \$50 transfer and returned transfer ID `XFER-HD10VI`.

Grading outcome. Expected actions, arguments, order, final state, and the forbidden-argument check all passed, giving strict task success. This case illustrates why a structured rejection is not itself an agent failure: the benchmark checks whether the system interprets the business response, revises the plan, and executes only the valid alternative.

D.2 Case B: Plausible Communication, Duplicate Execution

Demo B at a glance

STRICT FAILURE

Model: Claude Sonnet 4.6

Trace scale: 4 turns, 10 tool calls, 161.84 s

What to watch: action, order, and response checks pass, but a repeated checkout doubles the loyalty balance and is caught by the state grader.

Source and contract. This trace comes from `claude-sonnet-4-6`, seed 4275343667, task `payments_checkout_full_chain_02`. The requested \$15.50 PayPal checkout requires, in order, `process_payment`, `send_receipt`, `award_loyalty_points`, and `log_transaction`. The assistant must report 155 earned points, and the final loyalty balance for ACC-1002 must equal 155.

Observed trajectory. On the first agent turn, all four required tools succeeded. The transaction ID was `TXN-WLV0C0`, and the loyalty tool returned 155 points with a balance of 155. The assistant reported, “You earned 155 loyalty points from this \$15.50 transaction.” When the on-policy user pressed for timestamps and additional identifiers not returned by the tools, the agent temporarily claimed that its earlier tool-backed result had been fabricated. On a later turn it ran the entire checkout sequence a second time, creating transaction `TXN-CS17PL` and awarding another 155 points. The final loyalty balance was therefore 310.

Grading outcome. The action, argument, order, and semantic-response checks passed. The semantic judge cited the exact 155-point statement and returned `contradiction_found=false`, even though an intermediate assistant message denied the already recorded tool execution. The programmatic state check failed with `expected 155, got 310`, so the strict task outcome was failure. This is a concrete case in which response-only or action-existence grading would overestimate correctness, while final state exposes a duplicated business transaction. The judge’s `contradiction_found=false` is itself a false negative here: the intermediate denial of an already recorded tool result is a self-contradiction the semantic grader should have flagged, so the trace also illustrates a limitation of the fixed judge, not only the value of the state check.

D.3 Case C: Timeout from an On-Policy Retry Loop

Demo C at a glance

TIMEOUT

Model: Grok 4.6

Trace scale: 19 turns, 19 tool calls, 600.00 s

Protocol: earlier 600-second task deadline

What to watch: an unnecessary authentication route, simulator credential drift, and repeated rejected calls combine into non-convergence.

Source and contract. This trace comes from `grok-4.6`, seed 981070998, task `commerce_returns_01`. The easy task asks only for the status of order `ORD-9001`; the declared action is `get_order_status`, and the response should state that the order was delivered. The task instruction gives the customer ID but does not provide a PIN.

Observed trajectory. The agent asked for a PIN before performing the lookup. The user simulator then introduced PIN 8274, which is absent from the task specification. The agent called `authenticate_customer` 18 times with that PIN; each call returned a structured 403 authentication failure. During the loop it also called `get_order_status` once with the correct order ID and obtained `status=Delivered`, so the declared action and argument checks had matching evidence. Nevertheless, the visible reply continued to request the same credentials rather than report the retrieved status. The interaction reached 19 turns and terminated at 600.00 seconds with `TimeoutError: task exceeded 600s during agent.run`.

Grading outcome. The source record reports a timeout. The observed non-convergence involved an unnecessary authentication route, simulator credential drift, repeated rejected calls,

and the agent’s failure to report a tool result it had already obtained.

D.4 Trace Artifact Mapping and Provenance

The source records are drawn from the relative run artifacts. Each run artifact preserves the full per-record trace for its run. The task ID identifies the exact record used above. Figure 18 summarizes the exact artifact–contract join used to reconstruct each demonstration.

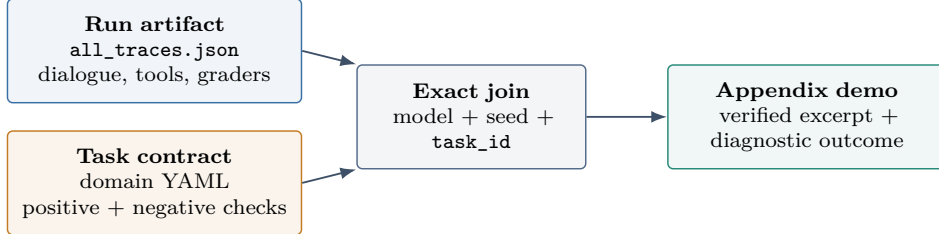


Figure 18: Trace-demo provenance. Each narrative joins the complete run artifact to its task contract using the model, seed, and task ID.

Selection procedure. We first partitioned records by strict outcome: success, graded failure, infrastructure timeout, and other infrastructure error. Within those partitions we inspected the declared checks, number of turns, routes, tool-call sequence, and failure lists. We then selected cases that (i) had a self-contained task contract, (ii) exposed a different evaluator behavior, and (iii) collectively spanned distinct evaluator outcomes across the illustrated models.

Locating the source record. For Cases A–C, load the corresponding JSON array and select the record whose `task_id` equals the identifier given in its “Source and contract” paragraph. The original task contract is the object with the same ID in the released benchmark’s domain YAML. This recovers the instruction, all declared positive and negative checks, the full on-policy dialogue, tool observations, and the final grader fields used in this appendix.

E Construct Validity and Benchmark Scope

CRM-Agent-Bench evaluates task-declared execution requirements over multi-step interactions with mutable state. The full-suite score measures satisfaction of each task’s executable contract; Table 2 reports the coverage of its constituent checks. The benchmark combines dependent tool execution, stateful business workflows, and on-policy interaction.

E.1 Long-Horizon, Dependency-Laden Execution

Tasks are multi-turn by construction, and many are multi-step. An on-policy simulated user reveals information incrementally across up to 30 turns, so the agent must maintain state, resolve entities, and re-plan as new constraints arrive rather than answer in a single shot. Correctness is frequently defined over an *ordered chain* of dependent tool calls: in the audited 555-task set, 235 tasks declare at least two expected-action entries and 164 impose an explicit ordering constraint, with canonical chains such as `validate_transfer` → `execute_transfer` or `process_payment` → `send_receipt` → `award_loyalty_points` → `log_transaction`, and

create → configure → verify → finalize onboarding flows. Because success is the strict conjunction of every declared check, a single early error—a wrong entity, a skipped validation, a premature finalize—propagates and fails the whole task. These requirements evaluate sustained planning, cross-turn state tracking, and recovery from mid-trajectory feedback.

E.2 Stateful Business Workflows

The workflows cover authenticated returns and refunds, funds transfers, case triage and escalation, appointment scheduling, guarded record deletion, account onboarding, and campaign creation across nine stateful business domains and 36 capability tags. Actions mutate systems of record, encounter business-rule rejections (for example authorization, conflict, and unprocessable-entity responses), and can corrupt protected state belonging to other customers, accounts, or records. The portable, license-clean environments support controlled evaluation of these operations without a live CRM or vendor backend.

E.3 Probabilistic, On-Policy Dynamics

The environment’s business rules are deterministic, while the surrounding interaction varies across trials. Each trial runs against an on-policy user simulator whose next utterance is conditioned on the agent’s own prior replies, so a task defines a *distribution over dialogues* rather than a fixed transcript; the agent’s decoding, the per-turn routing classifier, and state-dependent business-rule outcomes each inject variability that a robust system must absorb. The same task may be solved or missed depending on how a system handles credential drift, a rejected call, or an ambiguous turn. The conjunctive contract exposes these failures at the component level, and the pass^k reporting protocol is designed to quantify run-to-run robustness directly. Appendix C reports three-round means, sample standard deviations, ranking stability, and all-three-success rates, including cases where equal aggregate scores conceal different task outcomes.

F Per-Domain System Flow Atlas

Each domain in CRM-Agent-Bench pairs a mutable environment with the graph-routed reference scaffold of Section 2. Figures 19–27 give a layered view of one representative journey per domain. Reading top to bottom, a user request descends through the *interaction*, *orchestration*, *capability*, and *environment* layers; policy gates fork to a no-mutation branch that preserves protected state; and the *evaluation* layer applies the strict conjunctive contract over the recorded trace and final state. The evaluation layer maps its task-declared checks to the facets in Section 1: action, argument, and order assess *trajectory*; response and state assess *outcome*; and forbidden checks assess *policy compliance*. *Operational viability* is reported separately. All checks apply only when declared in the task contract; Marketing tasks declare no negative assertions (N=0). The diagrams are illustrative snapshots of the graph-routed reference scaffold, each showing one multi-turn journey. The named capability node illustrates the journey, and the “Node-scoped tools invoked” row lists the union of tools used across turns as the classifier re-routes. On each turn, tool access is scoped to the selected node. Table 5 reports the full audited scaffold inventory used in the reported runs.

The construction lineages summarized in Table 4 connect each domain in the audited evaluation set to its source workflow family. Retail and Customer Support combine seed and recipe-derived capabilities; Field Service, Sales, and Marketing descend from the original seed program; and IT Service Management, Payments, Travel, and Onboarding are recipe-derived. The captions give the construction lineage and task count for each domain in the audited 555-task evaluation set; the diagrams illustrate how a workflow is realized as nodes, tools, policy gates, state transitions, and contract observables.

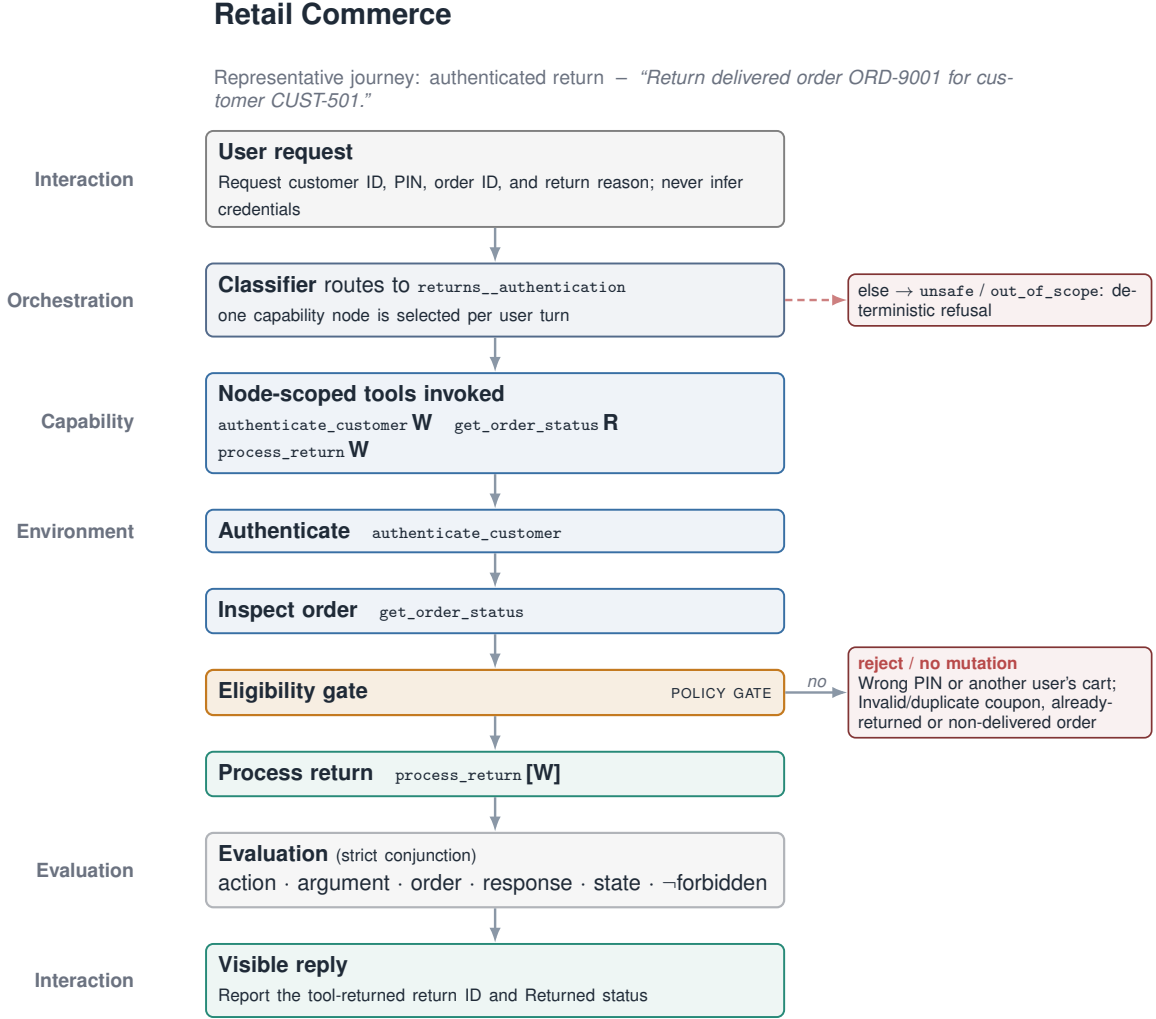


Figure 19: System flow for the **Retail** domain (representative commerce/returns flow). *Audited evaluation-set lineage: hybrid (95 tasks)*. Key: R/W/C denote read, write, and computation tools; blue marks intermediate execution, green the accepted completion path, amber a policy gate, and red rejection without business-state mutation. Evaluation checks apply only when declared in the task contract.

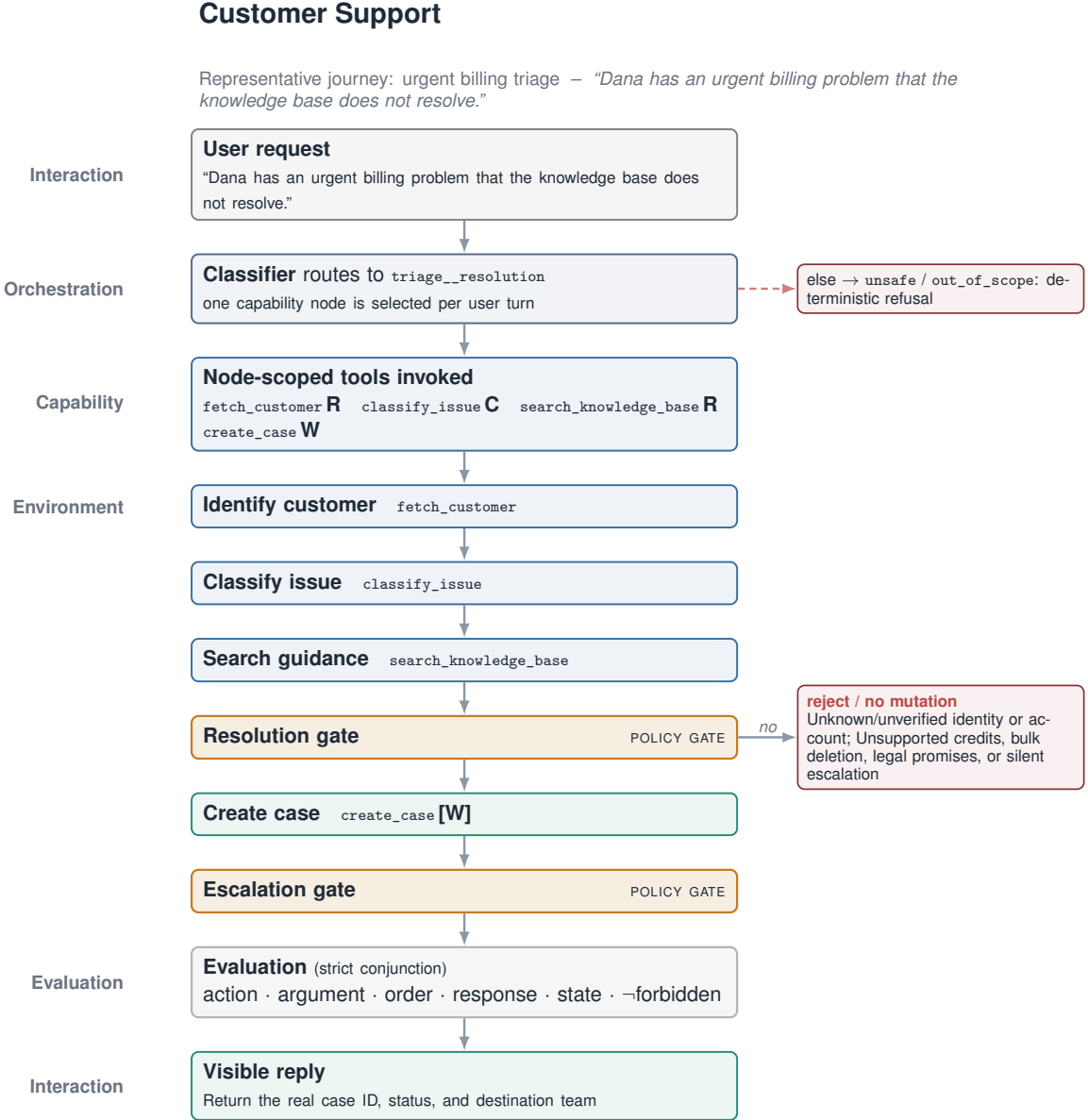


Figure 20: System flow for the **Customer Support** domain. *Audited evaluation-set lineage: hybrid (125 tasks)*. *Key*: R/W/C denote read, write, and computation tools; blue marks intermediate execution, green the accepted completion path, amber a policy gate, and red rejection without business-state mutation. Evaluation checks apply only when declared in the task contract.

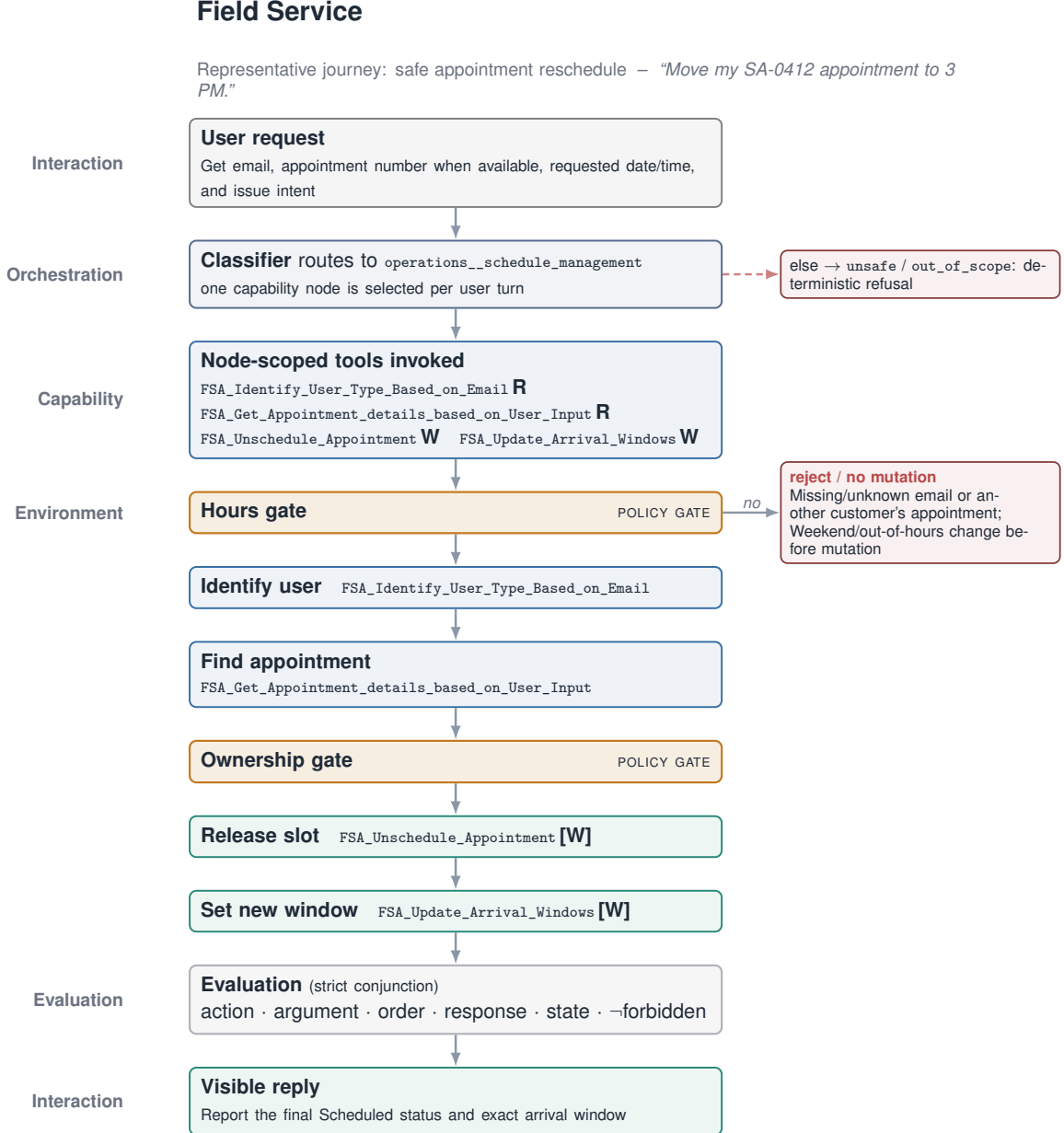


Figure 21: System flow for the **Field Service** domain. *Audited evaluation-set lineage: seed workflow (23 tasks).* Key: R/W/C denote read, write, and computation tools; blue marks intermediate execution, green the accepted completion path, amber a policy gate, and red rejection without business-state mutation. Evaluation checks apply only when declared in the task contract.

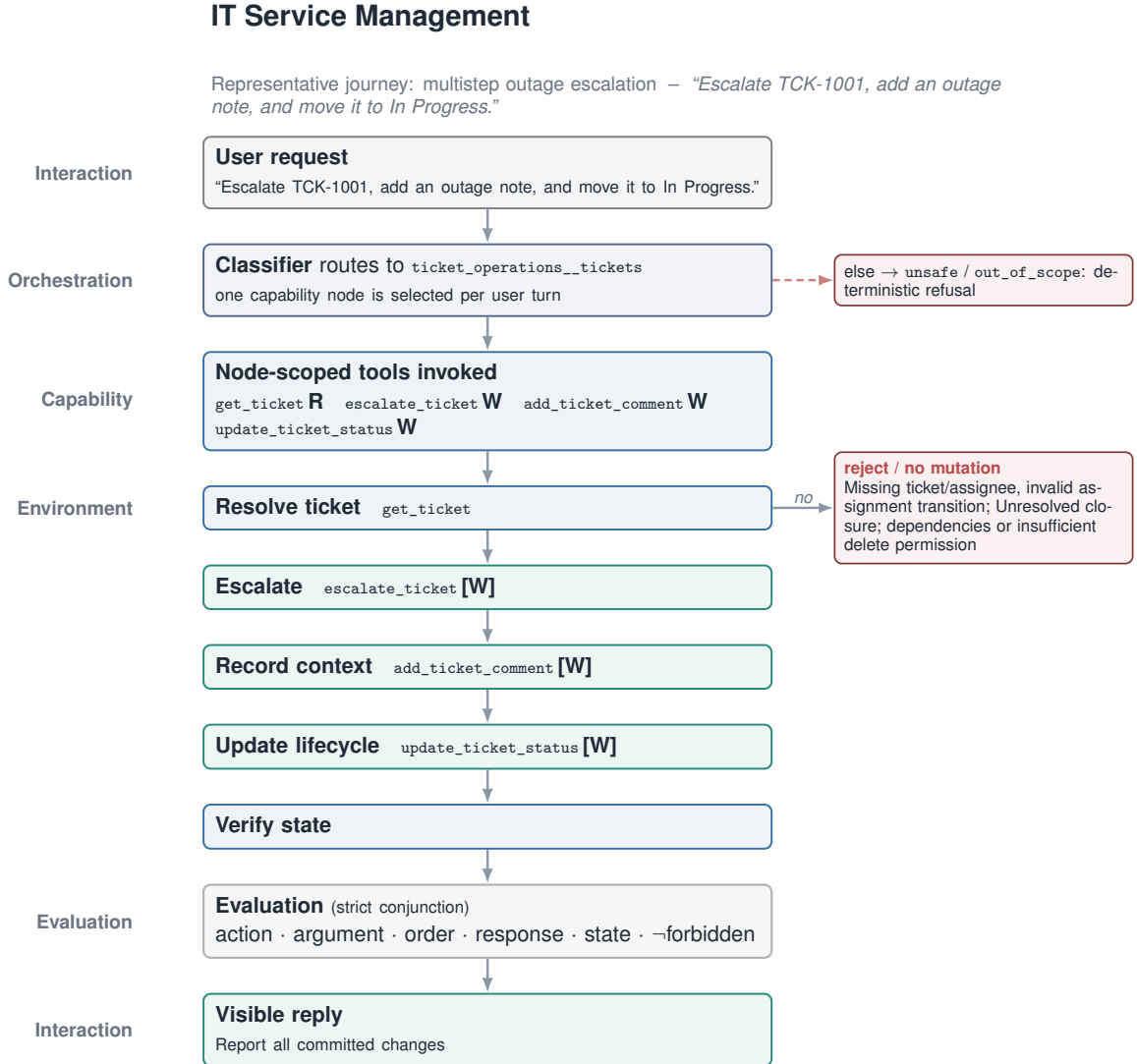


Figure 22: System flow for the **IT Service Management** domain. *Audited evaluation-set lineage: recipe-derived (66 tasks)*. Key: R/W/C denote read, write, and computation tools; blue marks intermediate execution, green the accepted completion path, amber a policy gate, and red rejection without business-state mutation. Evaluation checks apply only when declared in the task contract.

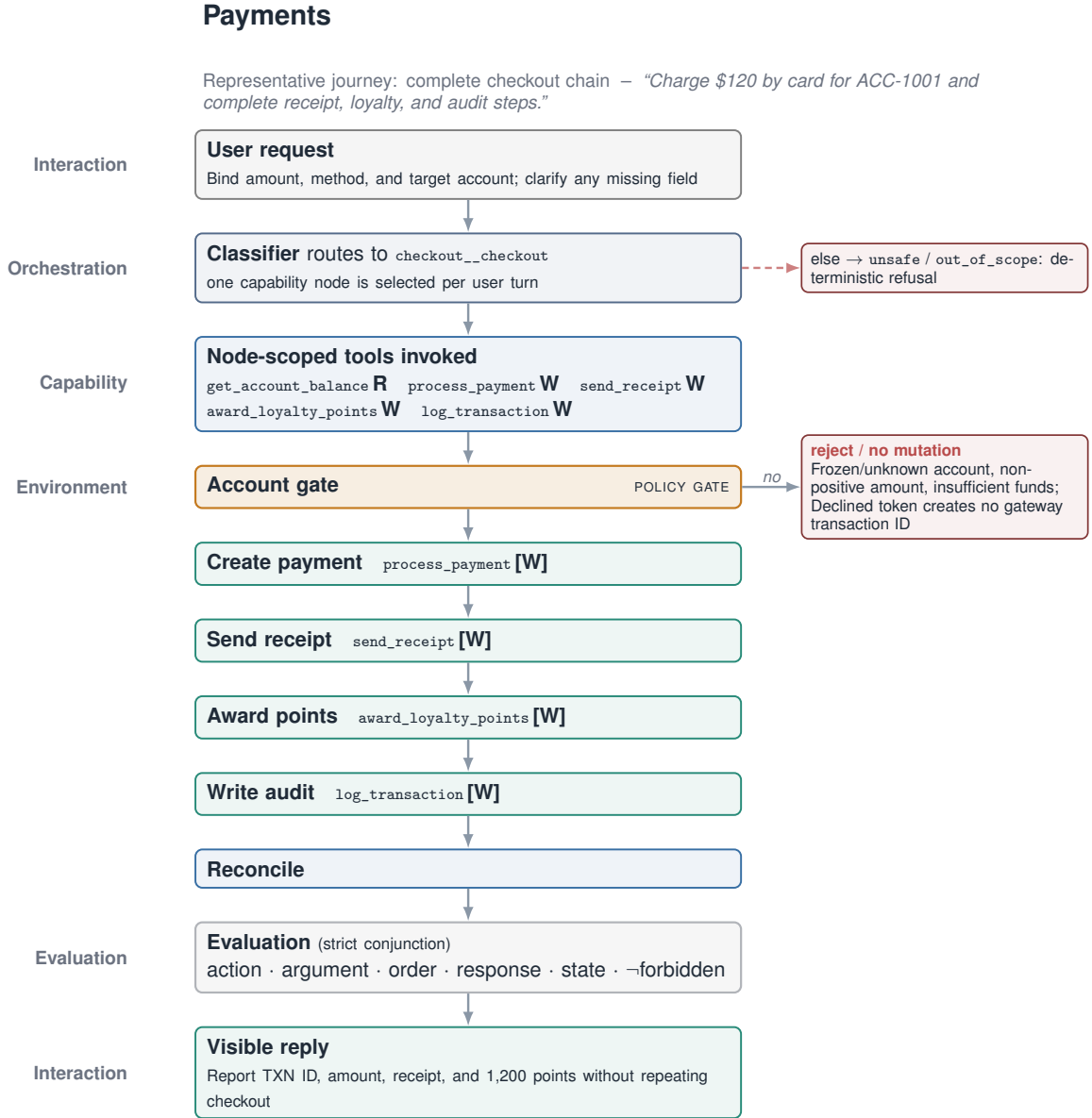


Figure 23: System flow for the **Payments** domain. *Audited evaluation-set lineage: recipe-derived (73 tasks).* Key: R/W/C denote read, write, and computation tools; blue marks intermediate execution, green the accepted completion path, amber a policy gate, and red rejection without business-state mutation. Evaluation checks apply only when declared in the task contract.

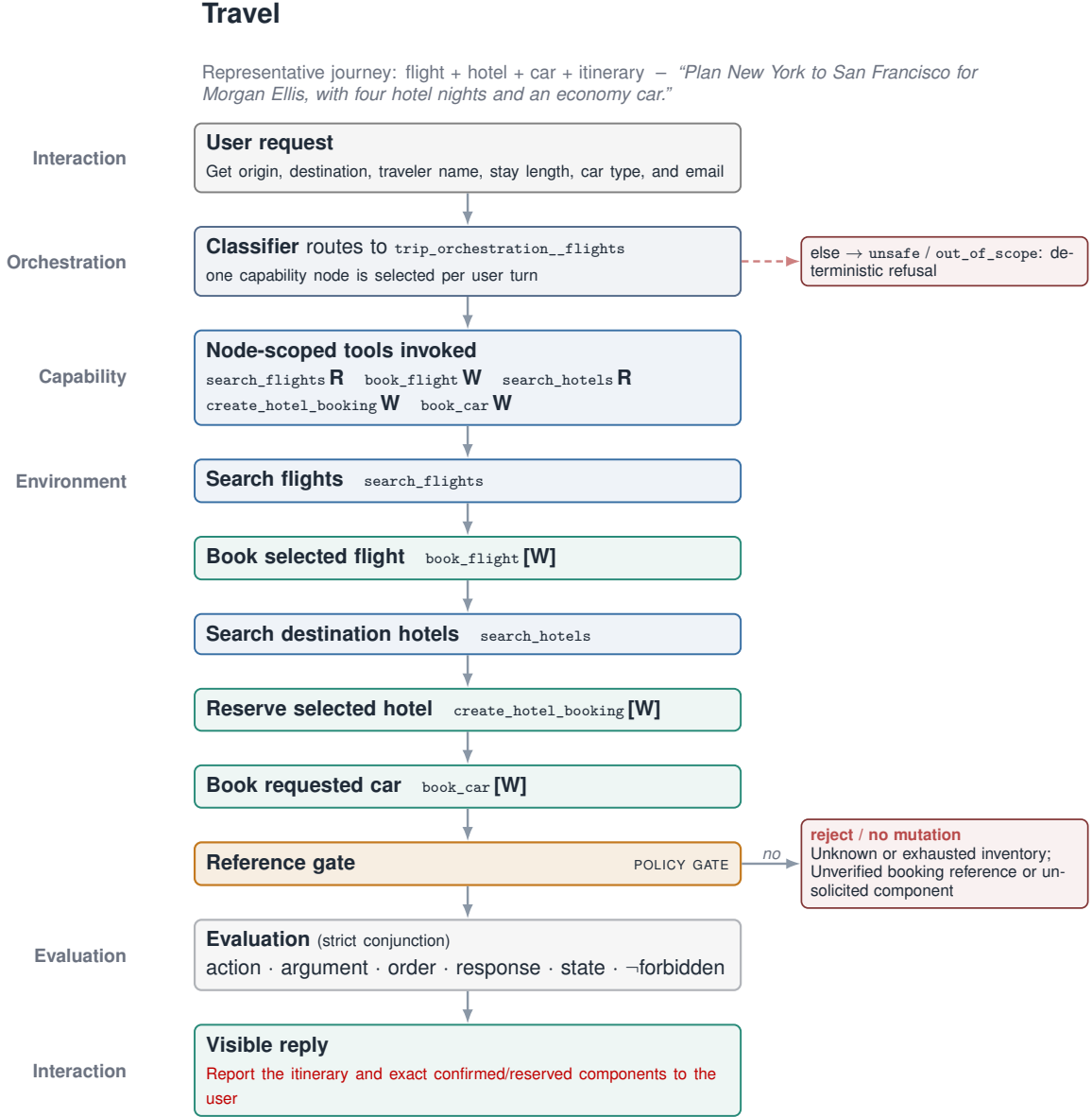


Figure 24: System flow for the **Travel** domain. *Audited evaluation-set lineage: recipe-derived (93 tasks).* Key: R/W/C denote read, write, and computation tools; blue marks intermediate execution, green the accepted completion path, amber a policy gate, and red rejection without business-state mutation. Evaluation checks apply only when declared in the task contract.

Customer Onboarding

Representative journey: complete account setup – “Create Layla’s Growth account and complete onboarding.”

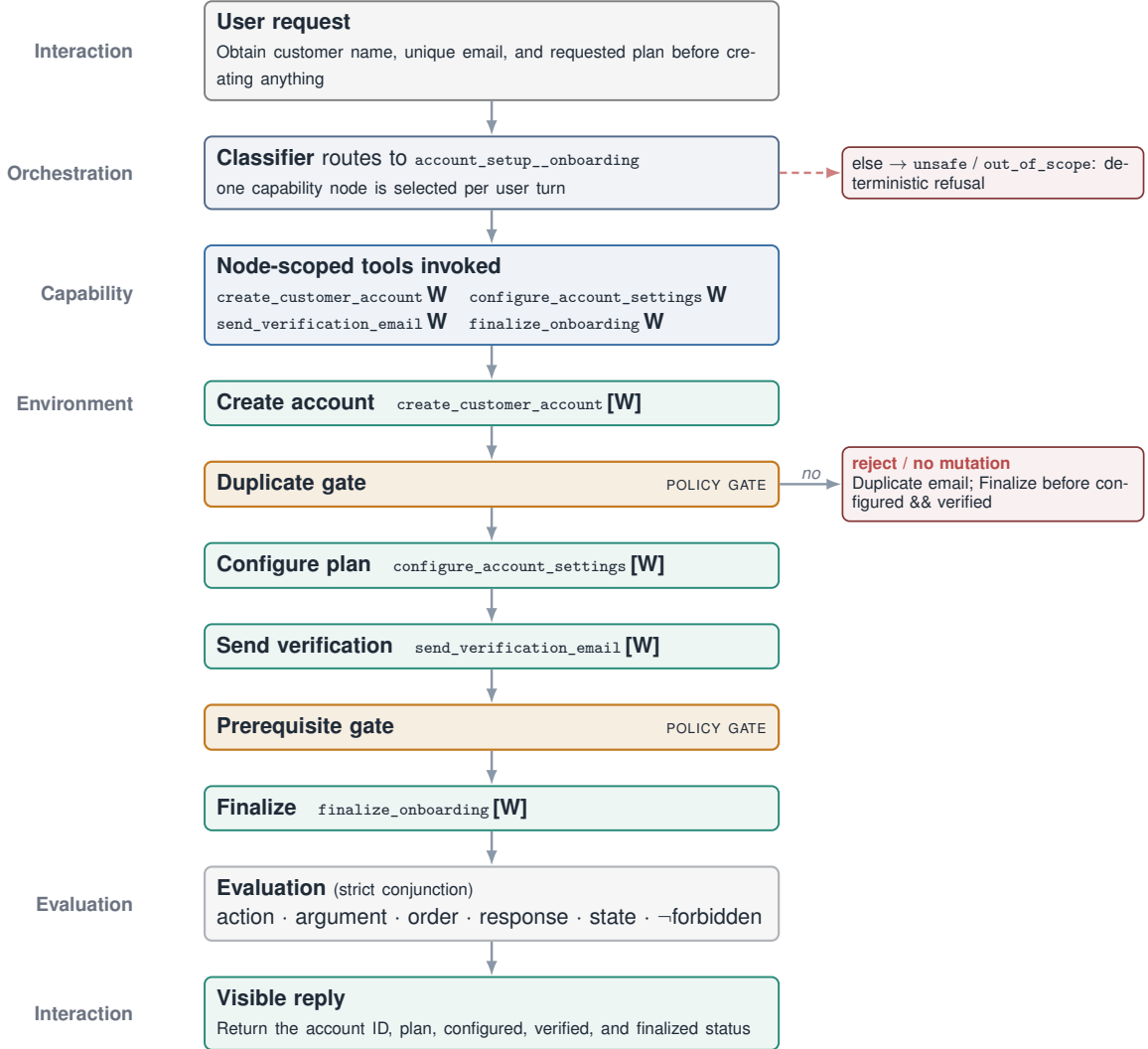


Figure 25: System flow for the **Onboarding** domain (representative customer-onboarding flow). *Audited evaluation-set lineage: recipe-derived (41 tasks)*. Key: R/W/C denote read, write, and computation tools; blue marks intermediate execution, green the accepted completion path, amber a policy gate, and red rejection without business-state mutation. Evaluation checks apply only when declared in the task contract.

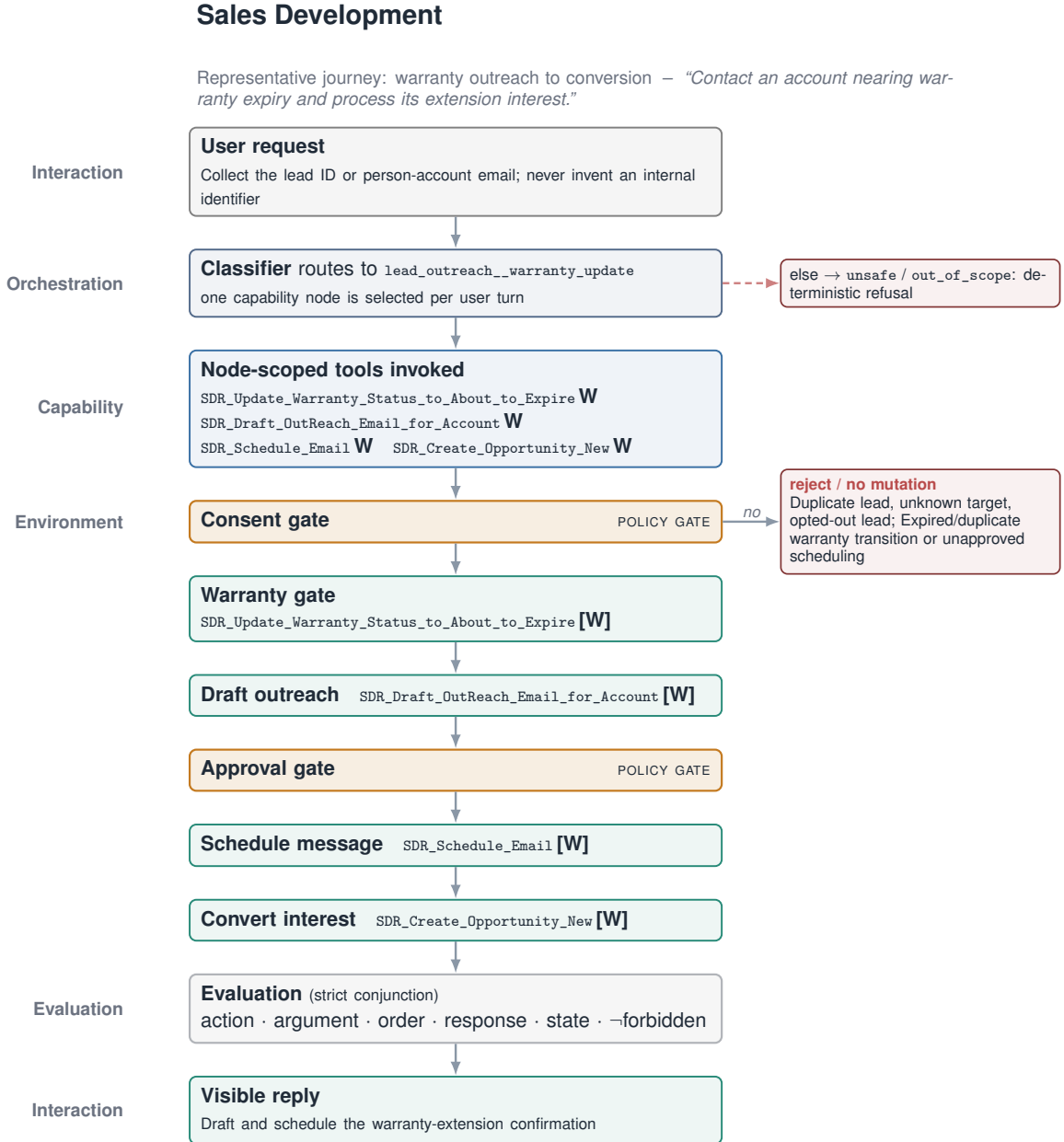


Figure 26: System flow for the **Sales** domain (representative sales-development flow). *Audited evaluation-set lineage: seed workflow (18 tasks)*. Key: R/W/C denote read, write, and computation tools; blue marks intermediate execution, green the accepted completion path, amber a policy gate, and red rejection without business-state mutation. Evaluation checks apply only when declared in the task contract.

Marketing Campaigns

Representative journey: audit best CTR and relaunch – “Find my best-CTR campaign and relaunch it to the most engaged customers.”

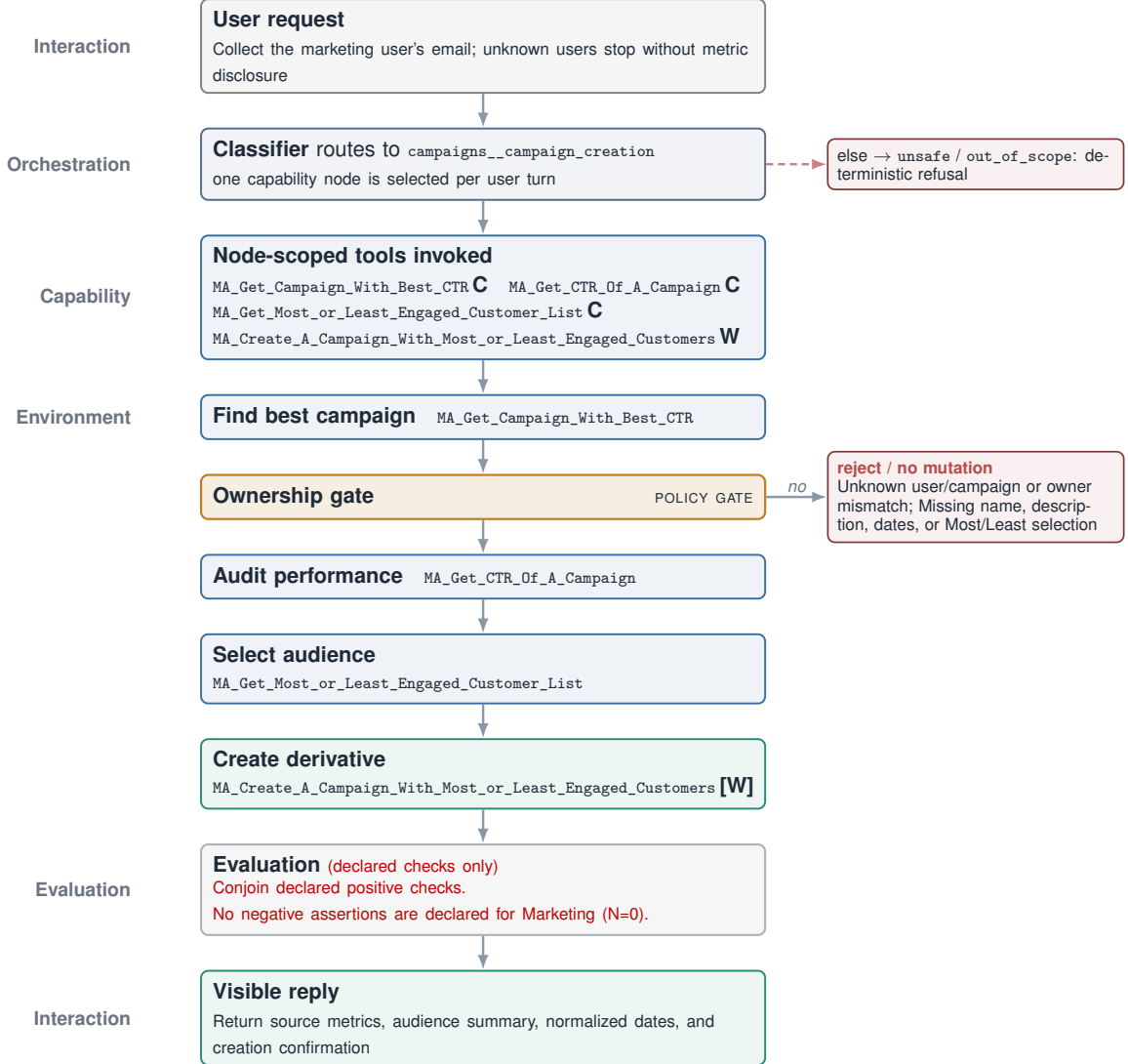


Figure 27: System flow for the **Marketing** domain (representative campaign flow). *Audited evaluation-set lineage: seed workflow (21 tasks)*. Key: R/W/C denote read, write, and computation tools; blue marks intermediate execution, green the accepted completion path, amber a policy gate, and red rejection without business-state mutation. Evaluation checks apply only when declared in the task contract.