EA-BENCH: MEASURING INFORMATION AND ACTION EFFICIENCY UNDER IMPERFECT INSTRUCTIONS

Anonymous authors
Paper under double-blind review

ABSTRACT

In real-world settings, users often provide incomplete, distracting, or inconsistent information. A useful agent must recognize these imperfections and resolve task-relevant uncertainty through efficient, targeted questioning. Measuring this ability requires separating execution difficulty from the quality of the instruction. To make this distinction measurable, we introduce the Efficient Agent Benchmark (EA-Bench). It holds task records and authoritative policies fixed while systematically varying missing information, redundancy, and conflicting instructions. EA-Bench comprises 270 paired conditions across 18 manually selected synthetic task families. We use task dependency graphs to enforce multi-step execution and counterfactual policy worlds to identify necessary clarifications, creating challenging imperfect instructions consistent with user intent. Experiments on nine mainstream open- and closed-weight model services show that combining multiple imperfections reduces pooled task accuracy from 76.5% to 60.2%. Across missing and mixed conditions, the hardest cases achieve accuracies of only 5.6% and 22.2%. More seriously, even when agents still complete tasks correctly under imperfect information, they make 23.6% more tool calls, request 847.5% more information items, and take 41.0% longer to complete the tasks.

1 INTRODUCTION

Recent language models have demonstrated strong capabilities in practical tasks, from specialized software engineering to everyday document and spreadsheet processing (Anthropic, 2026; OpenAI, 2026). As agents, these models integrate tool use with user interaction to execute tasks (Trivedi et al., 2024; Yao et al., 2022). In this context, a range of interactive benchmarks assess whether agents can solve tasks through interaction with simulated users. For example, tau-bench evaluates reliable task completion and policy adherence through simulated user dialogue and tool use (Yao et al., 2024). Building on this setting, τ^2-bench evaluates communication and coordination by allowing both agents and users to operate tools in a shared environment (Barres et al., 2025).

However, these benchmarks generally formulate tasks with predefined user goals, domain policies, and verifiable outcomes (Yao et al., 2024; Barres et al., 2025). In practice, user instructions may omit necessary information, include redundant details, or contain conflicting requirements. For example, a user may say, "Return the shirt I bought last week," without specifying which of several purchases they mean. To address these issues, some benchmarks, such as AppWorld-UL, introduce incomplete or ambiguous requests across tasks involving multiple apps to evaluate whether agents obtain missing information from users. AppWorld-UL finds that agents may overlook information gaps or fill them with unsupported assumptions (Chen et al., 2026). Nevertheless, these benchmarks still emphasize task success under imperfect instructions. Although AppWorld-UL also reports interaction quality and cost, the efficiency of resolving imperfect instructions relative to the information and actions actually required remains underexamined. More interestingly, a correct result does not necessarily mean that the agent had enough information to justify it. An agent may overlook an unstated constraint yet happen to produce an answer that satisfies it. Such accidental correctness does not demonstrate reliable recovery from imperfect instructions. Therefore, we introduce EA-Bench to study agent execution efficiency in real-world workflows. We quantitatively assess whether state-of-the-art agents can detect information gaps, redundancy, and conflicts, and use tools and user feedback efficiently to address them.

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

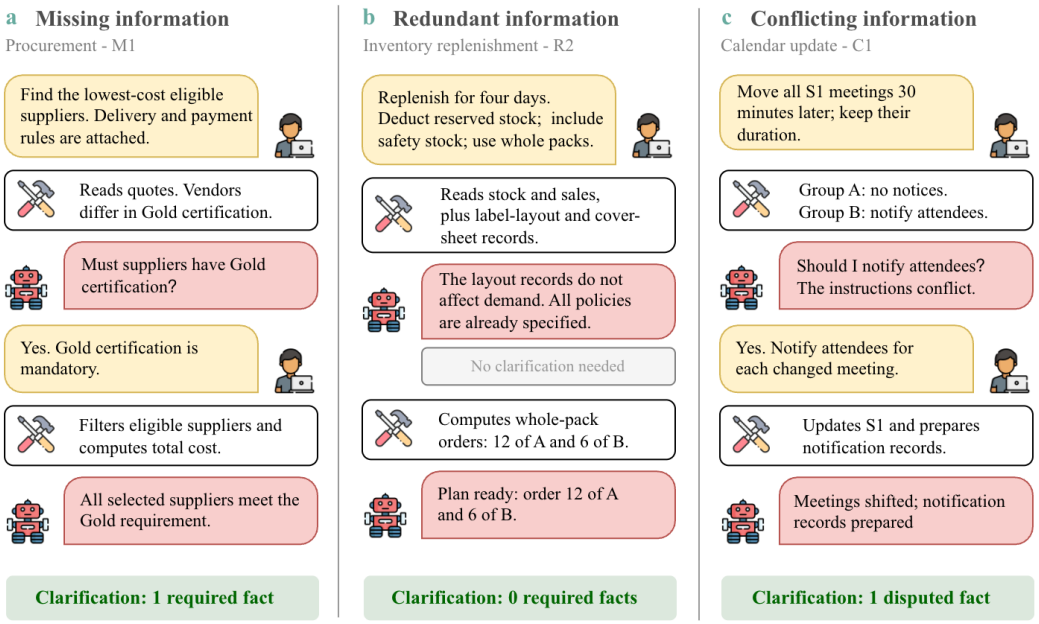

Figure 1: Three imperfect-information conditions illustrated as user-agent dialogues. Yellow bubbles denote user turns, pink bubbles agent turns, and bordered boxes environment operations.

To illustrate the imperfect information users may provide, we present three examples in Figure 1. When expressing their requirements, users may omit necessary information (left), include related but unnecessary information (middle), or provide contradictory instructions (right). All three are plausible situations in everyday user requests.

To this end, we select work-related scenarios from existing benchmarks (Li et al., 2026; Chen et al., 2026; Styles et al., 2024) and represent each workflow as a task dependency graph. Our design follows two principles. First, user clarification must be necessary to guarantee correctness when essential information is missing or conflicting. Second, task-specific lower bounds must make execution and information efficiency measurable. To satisfy the first principle, we construct counterfactual policy worlds with identical public observations but different correct outputs. The facts needed to distinguish these worlds define the information minimum. For the second, tool operations enforce graph dependencies, giving an attainable action minimum. Task records and authoritative policies remain fixed across variants to isolate the effects of instruction imperfections. In addition, we design a user simulator that answers clarification questions using predefined task knowledge, enabling controlled and reproducible interactions.

Finally, EA-Bench contains 270 conditions across 18 synthetic task families, with six families from each source group. Each family has one complete-information condition, four levels each of missingness, redundancy, and conflict, and two mixed conditions. We evaluate nine mainstream open- and closed-weight model services on all conditions. Our contributions are threefold.

EA-Bench quantifies efficiency under imperfect instructions. To our knowledge, EA-Bench is the first benchmark to jointly measure excess tool use and information requests under paired missing, redundant, and conflicting instructions using explicit task-level lower bounds.

EA-Bench provides a reusable construction procedure. The EA-Bench construction procedure can be extended to other task benchmarks by specifying dependency graphs, hidden policy facts, and executable outcome checks. This supports controlled and comparable evaluations across a wider range of perturbations and task domains.

EA-Bench reveals inefficient recovery from imperfect instructions. Across nine models, combined imperfections reduce accuracy from 76.5% to 60.2%. Even on matched, information-sufficient

successes, tool calls increase by 23.6%, information requests by 847.5%, and execution time by 41.0%. Correct outputs also frequently conceal unresolved information gaps.

2 Related Work

Execution and efficiency evaluation. Since early frameworks such as ReAct linked reasoning with external actions (Yao et al., 2022), agent evaluation has increasingly focused on effective tool use and task execution. BFCL evaluates function calling (Patil et al., 2025). WebArena, SWE-bench, and AppWorld check executed outcomes in realistic environments (Zhou et al., 2024; Jimenez et al., 2024; Trivedi et al., 2024). Efficiency benchmarks additionally measure the resources consumed by successful solutions. EffiBench measures code runtime and memory against canonical programs (Huang et al., 2024), and OSWorld-Human compares computer-use trajectories with human references (Abhyankar et al., 2026). AgentBoard tracks intermediate progress, supplies golden actions for tool subsets, and reports runtime estimates (Ma et al., 2024). These references support different notions of necessary execution; they should not all be interpreted as theoretical optima. Joint accuracy and cost reporting is also motivated by AI Agents That Matter (Kapoor et al., 2024).

Information and user interaction. User interaction is central to clarifying task requirements and aligning agent behavior with user intent (Subramonyam et al., 2024). Tau-bench, τ^2-bench, ToolSandbox, and UserBench make communication part of agent evaluation (Yao et al., 2024; Barres et al., 2025; Lu et al., 2025; Qian et al., 2025). AppWorld-UL additionally controls user-held knowledge and measures coverage of expected questions and interaction burden (Chen et al., 2026). Context benchmarks examine complementary abilities. CL-bench studies applying supplied knowledge (Dou et al., 2026), RULER and HELMET test long-context use (Hsieh et al., 2024; Yen et al., 2025), and IFEval and FollowBench test verifiable instruction constraints (Zhou et al., 2023; Jiang et al., 2024). Simulated-user results remain conditional on the proxy's validity (Seshadri et al., 2026). EA-Bench brings execution references and necessary-information accounting together with separate missing, redundant, and conflicting-policy variants. Table 1 compares these selected dimensions; its coverage labels describe the metrics and conditions provided by the cited protocols.

Table 1: **Comparison of Efficiency Benchmarks.** Costs include steps, time, memory, or API spend. Missing/Redundant/Conflict denote controlled information variants. †: golden-action references for AgentBoard tool subsets. EA-Bench is the only compared benchmark that combines explicit action and information baselines with controlled missing, redundant, and conflicting instructions.

	Execution and efficiency			Information and instruction conditions			
Benchmark	Exec. check	Cost reported	Exec. reference	Required info	Missing	Redundant	Conflict
EffiBench (code)	✓	✓	✓	×	×	×	×
OSWorld-Human (GUI)	✓	✓	✓	×	×	×	×
AgentBoard (agents)	✓	✓	✓†	×	×	×	×
AppWorld-UL (agents)	✓	✓	×	✓	✓	×	×
EA-Bench (ours)	✓	✓	✓	✓	✓	✓	✓

3 Task and Interaction Formulation

In this section, we describe the task formulation and interaction protocol of EA-Bench. We explain what information agents receive, how tool actions are executed and checked, and how agents obtain missing information through clarification with a simulated user.

3.1 Task specification and paired variants

To support paired comparisons, we group instruction variants of the same underlying task into a task family. For each family, we specify input data, task requirements, a dependency graph, and an automated evaluator. Input data include records such as supplier quotations or calendar entries.

Task requirements determine which outcomes are valid, such as whether suppliers need a particular certification or whether schedule changes require attendee notifications. To assess task completion, we check submitted results against these requirements.

To model information asymmetry, we provide agents with natural-language instructions and read tools for accessing task data. We then select four decision-relevant requirements in each family and vary how they are communicated. If a requirement is omitted or stated inconsistently, the agent can resolve it by issuing an `ask_user` request to the simulated user.

Building on this setup, we create missing-information variants by withholding selected requirements. To introduce conflicts, we provide incompatible statements without specifying which should govern. To introduce redundancy, we add related but unnecessary material. We further combine omissions and conflicts on different requirements with redundant material to form mixed variants. To isolate the effects of instruction quality, we keep the task-relevant data, intended requirements, dependency graph, and evaluation criterion fixed across all variants.

3.2 Execution and terminal grading

The model emits one JSON action at a time. Reads return fixed snapshots and opaque handles; artifact operations require valid prerequisite handles. Replacing an upstream artifact invalidates dependent handles, making the DAG operational.

The first report ends the episode, even if malformed. Intermediate artifacts may be revised before reporting; intermediate checks enforce structure and dependencies, while final grading checks semantics. Final grades cannot be used for repair within the same attempt.

Each attempt permits 40 model turns. The recorded `tool_calls` counter includes user requests and invalid or empty actions. We retain this action-attempt definition, with empty and invalid turns extracted separately for diagnosis.

3.3 Knowledge-bounded user simulation

A Qwen3-8B service mediates natural-language questions. Its scope stage receives an answer-blind topic catalog and must support selected topics with verifiable textual evidence. The answering stage receives authoritative values only for those topics; host validation controls what reaches the agent. A user-tool action can therefore require multiple simulator calls. Scope errors and rejected quotations remain evaluation-system failures; later fixes are not retroactively applied to historical runs.

Post-run atomic annotation counts independent requested facts without giving feedback to the agent. Repeated wording within a turn is merged; requests repeated across turns and non-required facts still count toward burden. Annotation records which required facts were requested and returned. Strict information analysis requires complete annotation accounting for every `ask_user` attempt.

4 Benchmark Construction and Efficiency Measures

4.1 Task sources and instruction variants

For EA-Bench, we draw task scenarios from three established benchmarks, JobBench, AppWorld-UL, and WorkBench (Li et al., 2026; Chen et al., 2026; Styles et al., 2024). We select six scenarios from each source and adapt them into 18 synthetic task families covering professional and everyday workflows.

To vary instruction quality systematically, we construct 15 conditions per family, covering complete, missing, redundant, conflicting, and mixed information. This gives 270 conditions in total. Appendices A and B provide the task catalog and perturbation settings.

4.2 Efficiency measures and lower bounds

To quantify agent efficiency, we measure excess tool calls and excess information requests. Both compare observed consumption with task-specific lower bounds.

Information overhead. Let H be the necessary facts unavailable in the initial input and Q the number of requested information items. We define the information minimum and excess as

$$I^* = |H|, \qquad E_I = Q - I^*. \tag{1}$$

Counterfactual policy worlds determine which facts belong to H.

Tool-call overhead. Let K be the number of required task operations and A the observed number of tool calls. Since all required facts can be requested together, the action minimum and excess are

$$A^* = K + \mathbf{1}[I^* > 0], \qquad E_A = A - A^*. \tag{2}$$

We compute both measures on tasks completed correctly with all required information. Lower values indicate greater efficiency. Appendix C provides the lower-bound derivations.

5 Evaluation Protocol and Metrics

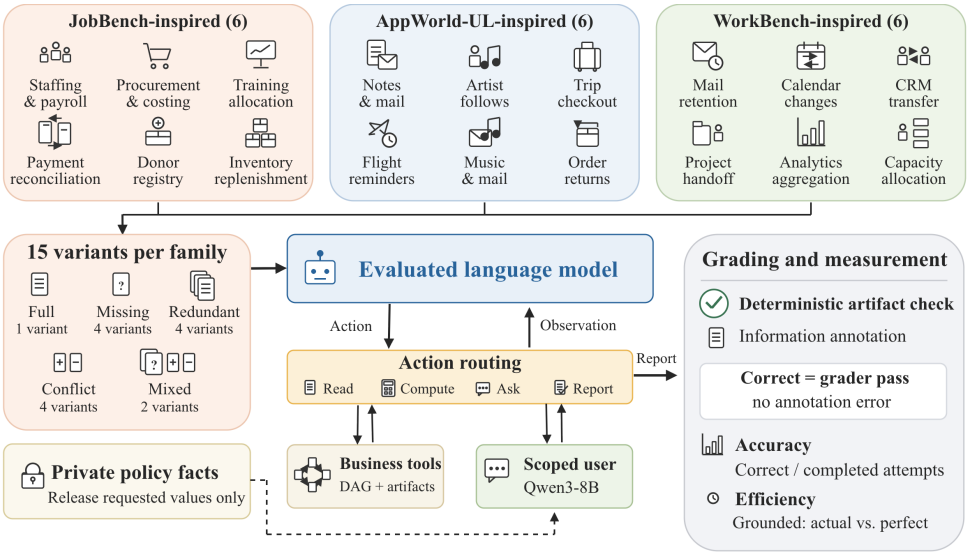

Figure 2: EA-Bench evaluation flow. Icons identify all 18 task families and the 15 variants per family (1 Full, 4 Missing, 4 Redundant, 4 Conflict, 2 Mixed), yielding 270 conditions. Actions interact with business tools or the scoped user; the first report triggers grading. Annotation and runtime errors count as failures; efficiency uses correct, information-sufficient tasks.

5.1 Experimental Setup

We evaluate nine model services across all 270 conditions, for a total of 2,430 original attempts. All models that support this setting use low reasoning effort; MiniMax-M2.5 uses native thinking. Each condition allows up to 40 model turns and 16,384 output tokens per reply, with Qwen3-8B as the user simulator. Every model has the same task coverage, and annotation and runtime errors count as failures. Tables 2 and 3 report detailed performance and efficiency results. Models are grouped into open- and closed-weight families and ranked by decreasing overall accuracy within each group. Appendix H provides model sources and licensing details.

5.2 Outcome Metrics

Task success alone does not establish that an agent identified and resolved all relevant information gaps through reasoning, tool use, or user interaction. We therefore distinguish two outcome measures, raw outcome success and grounded success.

Table 2: **Performance on the original attempts**: annotation-aware accuracy (%). Snapshot: 2026-09-26 18:33 China time. Cells show accuracy and completed sample size n; annotation and runtime errors count as failures. Each model targets 270 conditions. Open- and closed-weight families are grouped separately; rows within each group follow decreasing Overall accuracy. All nine models completed 270 conditions through hosted APIs.

Model Names	Overall (%)	Full (%)	Missing (%)	Redundant (%)	Conflict (%)	Mixed (%)
Open-weight model families						
Kimi-K3	**80.4** n=270	**83.3** n=18	**59.7** n=72	**90.3** n=72	**90.3** n=72	**80.6** n=36
HY4-Preview	**78.5** n=270	**83.3** n=18	**63.9** n=72	**87.5** n=72	**88.9** n=72	**66.7** n=36
DeepSeek-V4-Pro	**75.6** n=270	**88.9** n=18	**68.1** n=72	**77.8** n=72	**77.8** n=72	**75.0** n=36
Qwen3.8-Max	**74.1** n=270	**83.3** n=18	**56.9** n=72	**88.9** n=72	**81.9** n=72	**58.3** n=36
GLM-5.3	**59.3** n=270	**72.2** n=18	**45.8** n=72	**70.8** n=72	**62.5** n=72	**50.0** n=36
MiniMax-M2.5	**20.4** n=270	**27.8** n=18	**9.7** n=72	**37.5** n=72	**16.7** n=72	**11.1** n=36
Closed-weight model families						
GPT-6-Astra	**78.5** n=270	**77.8** n=18	**72.2** n=72	**88.9** n=72	**79.2** n=72	**69.4** n=36
Gemini-3.5-Flash	**78.1** n=270	**94.4** n=18	**77.8** n=72	**81.9** n=72	**75.0** n=72	**69.4** n=36
GPT-5.5	**75.2** n=270	**77.8** n=18	**75.0** n=72	**86.1** n=72	**70.8** n=72	**61.1** n=36

We define **raw outcome success** as passing the deterministic evaluator on the first final submission. Our primary accuracy also accounts for annotation status. Annotation errors and runtime errors both count as failures.

We define **grounded success** as a correct result supported by recorded information acquisition. In addition to passing the evaluator without annotation errors, the agent must have requested and obtained every task fact that requires clarification. This criterion checks the interaction record, rather than assuming that a correct answer implies sufficient information.

Using these measures and the recorded execution status, we assign each attempt to one of six mutually exclusive categories: grounded correct, ungrounded correct, failed first submission, action-budget exhaustion, runtime error, or annotation error. Runtime errors take precedence, followed by annotation errors and then the task outcome. This partition prevents unsuccessful or insufficiently informed attempts from appearing efficient merely because they stop early.

5.3 Efficiency metrics

We measure efficiency using the average extra tool calls and requested information items per successful task relative to the perfect baseline. Both are lower-is-better, with zero denoting no excess. Let S_A contain grounded successes and S_I the subset with complete accounting for every question attempt; n_A and n_I are their sizes.

$$\Delta_A = \frac{\sum_{i \in S_A} (A_i - A_i^*)}{n_A}, \qquad \Delta_I = \frac{\sum_{i \in S_I} (Q_i - I_i^*)}{n_I}. \tag{3}$$

Tools retain the recorded action-attempt counter, including user requests and invalid turns. Full and redundancy have $I^* = 0$, so the same information metric measures extra requests directly. Table 3 reports model-specific subsets with sample sizes; Figure 6 supplies grounded-success coverage. Matched comparisons use identical tasks. Failures retain their costs separately in Appendix E.

6 Experimental Results

In this section, we analyze how imperfect instructions affect task accuracy and agent efficiency across nine models. We examine information gaps in correct outputs, compare tool use and questioning on matched tasks, and assess how perturbation severity and task depth relate to performance.

Table 3: **Mean extra consumption per grounded-success task** (↓): (a) tool calls and (b) requested information items. Zero is optimal; shorter bars are better. Each cell gives its own sample size n; information requires complete question accounting. Full and Redundant use the same formula with $I^* = 0$. – means no eligible successes ($n = 0$). Open- and closed-weight families are grouped separately and sorted by Overall accuracy within each group, as in Table 2. Read with grounded-success coverage in Figure 6.

(a) Tools: mean extra calls per task ↓ (bar scale: 0–3)

Model Names	Overall	Full	Missing	Redundant	Conflict	Mixed
Open-weight model families						
Kimi-K3	**0.45** n=198	**0.07** n=15	**0.68** n=31	**0.54** n=65	**0.38** n=64	**0.39** n=23
HY4-Preview	**0.15** n=190	**0.00** n=15	**0.27** n=30	**0.14** n=63	**0.09** n=64	**0.33** n=18
DeepSeek-V4-Pro	**1.95** n=188	**0.62** n=16	**2.59** n=37	**1.45** n=56	**2.25** n=56	**2.35** n=23
Qwen3.8-Max	**1.27** n=173	**0.20** n=15	**2.55** n=22	**1.02** n=64	**1.08** n=59	**2.38** n=13
GLM-5.3	**1.11** n=139	**0.08** n=13	**0.89** n=19	**0.53** n=51	**1.78** n=45	**2.64** n=11
MiniMax-M2.5	**1.67** n=45	**0.60** n=5	**1.00** n=3	**2.07** n=27	**1.38** n=8	**1.00** n=2
Closed-weight model families						
GPT-6-Astra	**0.13** n=212	**0.50** n=14	**0.00** n=52	**0.23** n=64	**0.09** n=57	**0.04** n=25
Gemini-3.5-Flash	**0.20** n=188	**0.06** n=17	**0.41** n=39	**0.07** n=59	**0.22** n=54	**0.21** n=19
GPT-5.5	**0.20** n=185	**0.00** n=14	**0.20** n=40	**0.03** n=62	**0.37** n=51	**0.44** n=18

(b) Information: mean extra requested items per task ↓ (bar scale: 0–2)

Model Names	Overall	Full	Missing	Redundant	Conflict	Mixed
Open-weight model families						
Kimi-K3	**0.27** n=183	**0.00** n=15	**0.69** n=26	**0.28** n=65	**0.19** n=57	**0.10** n=20
HY4-Preview	**0.16** n=178	**0.00** n=15	**0.29** n=24	**0.06** n=63	**0.21** n=61	**0.27** n=15
DeepSeek-V4-Pro	**0.16** n=161	**0.00** n=16	**0.37** n=27	**0.00** n=56	**0.20** n=44	**0.33** n=18
Qwen3.8-Max	**0.06** n=164	**0.00** n=15	**0.06** n=17	**0.08** n=64	**0.07** n=56	**0.00** n=12
GLM-5.3	**0.10** n=117	**0.00** n=13	**0.53** n=15	**0.00** n=51	**0.12** n=33	**0.00** n=5
MiniMax-M2.5	**0.02** n=44	**0.00** n=5	**0.33** n=3	**0.00** n=27	**0.00** n=7	**0.00** n=2
Closed-weight model families						
GPT-6-Astra	**0.94** n=208	**1.71** n=14	**1.48** n=52	**0.88** n=64	**0.49** n=53	**0.52** n=25
Gemini-3.5-Flash	**0.03** n=178	**0.00** n=17	**0.00** n=34	**0.00** n=59	**0.10** n=52	**0.00** n=16
GPT-5.5	**0.16** n=183	**0.00** n=14	**0.33** n=39	**0.03** n=62	**0.24** n=50	**0.17** n=18

6.1 Accuracy and cost under combined imperfections

Complex instructions reduce accuracy and increase cost. Across nine models, accuracy declines from 76.5% on Full to 60.2% on Mixed. On 134 correct, information-sufficient Full–Mixed pairs, tool calls increase by 23.6% and execution time by 41.0%; mean extra calls rise from 0.21 to 0.76. Across Missing and Mixed, the lowest family accuracies are 3/54 for music selection and mailing and 12/54 for procurement. Their Full scores are 0/9 and 9/9; the former is already difficult under Full.

Correct artifacts can conceal unresolved requirements. Only 22 of Qwen's 41 correct missing-information outputs are information-sufficient. The corresponding counts are 30/46 for HY4 and 39/56 for Gemini, compared with 52/52 for Astra. Correctness alone therefore overstates reliable information acquisition. Efficiency must be read together with information-sufficient success coverage.

6.2 Efficient tool use does not imply selective questioning

Execution overhead and user burden differ. Table 3 shows that Astra stays close to the action minimum but asks unnecessary questions even in Full and redundancy. Qwen asks more selectively yet incurs more execution overhead. Across 116 strictly annotated Full–Mixed pairs, actual information requests increase from 40 to 379 items (847.5%). Required information also increases, while mean excess requests decrease. Thus the increase in total requests is not entirely avoidable overhead.

Differences persist on identical successful tasks. The nine models share 20 information-sufficient successes for tool use and 17 for strict information analysis (Figure 3). HY4 uses exactly the 122

required tool calls, while MiniMax uses 159. Astra requests 32 facts against a minimum of 11; Gemini and GLM each request 11. This small intersection controls task identity but selects jointly solved cases. Pairwise comparisons and overall coverage remain necessary.

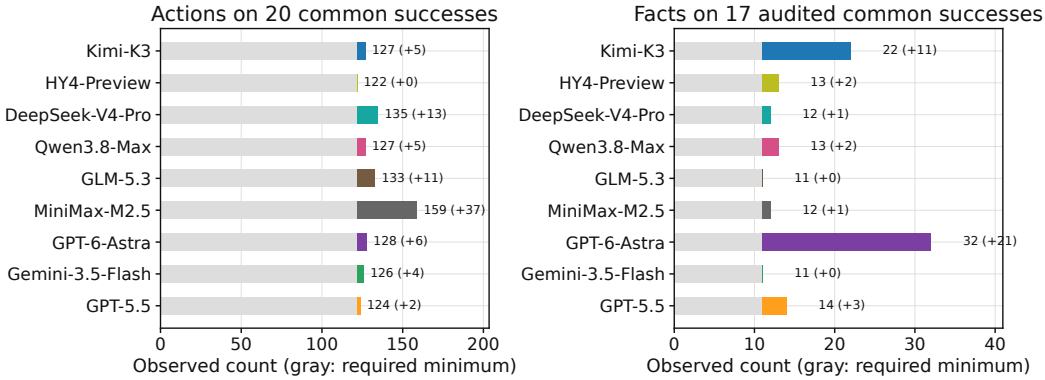

Figure 3: Resource use on shared information-sufficient successes across all nine models. Gray denotes the required minimum; color denotes excess. Tools use 20 tasks (122 required calls); information uses 17 strictly annotated tasks (11 required facts). Labels show actual totals and excess.

6.3 Severity does not define a universal difficulty ordering

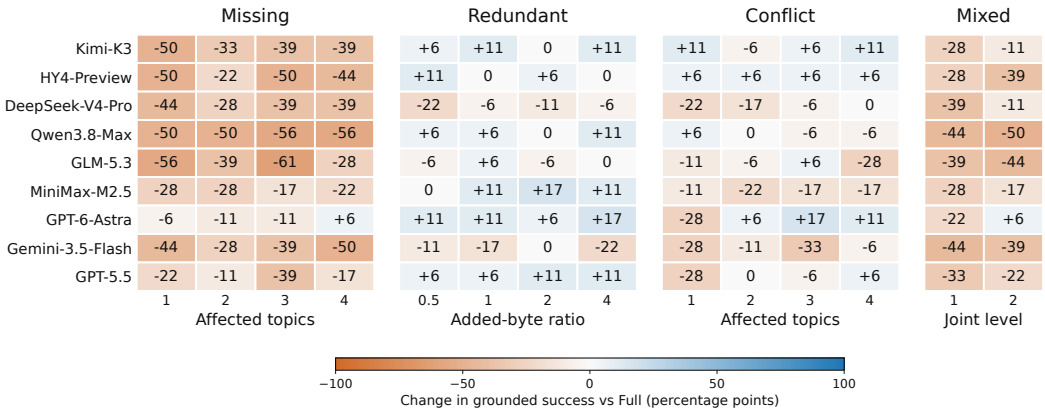

Figure 4: Change in information-sufficient success rate relative to Full. Orange indicates a decrease and blue an increase; cells show percentage points. Every model uses the same 18 families at each level. Redundancy uses added-byte ratios; mixed levels vary all three perturbations.

Omission is more disruptive than additional irrelevant material. Figure 4 compares each perturbation with its matched Full baseline. Several models lose much of their grounded performance as soon as clarification becomes necessary, whereas redundancy stays comparatively close to Full. The contrast suggests that detecting an information gap is a more consequential obstacle in these tasks than filtering the added material.

Interestingly, higher missingness or conflict levels sometimes yield better observed scores. These levels change which requirements need clarification and may make uncertainty more apparent.

6.4 Graph depth does not explain difficulty by itself

Depth groups also differ in task composition. All nine models lose accuracy from depth five to six; eight recover at depth seven, while MiniMax solves none of that group (Figure 5). Depth six covers three families, but depth seven contains only capacity-based assignment. These results cannot establish a causal effect of depth independently of task content.

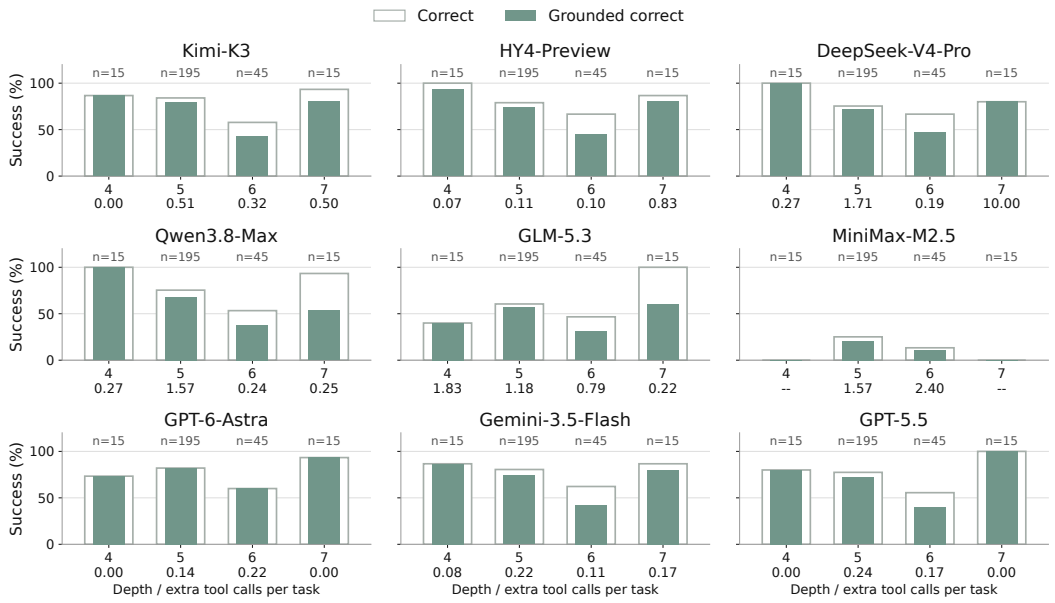

Figure 5: Correctness (outlined bars) and information-sufficient correctness (filled bars) by graph depth. Labels give attempt counts; values beneath depth ticks give mean extra tool calls per information-sufficient success. Depths 4/5/6/7 contain 1/13/3/1 families. – denotes no eligible successes.

Successful execution cost and failure probability need not move together. Qwen uses fewer extra calls on its successful depth-six tasks despite lower accuracy in that group. Conversely, DeepSeek's depth-seven successes still involve substantial extra execution. A model can thus complete a task with extra execution, or appear economical because the harder cases are absent from its successful subset. Reporting both outcome coverage and excess consumption is essential to distinguish these behaviors.

7 Conclusion

We introduced EA-Bench to quantify how imperfect user instructions affect agent performance and efficiency. Its 270 conditions span 18 synthetic task families and systematically vary missing information, redundancy, and conflict. Task dependency graphs and counterfactual policy worlds provide explicit action and information baselines. This design separates necessary work from excess resource use and distinguishes correct outputs from information-sufficient success. The construction procedure also supports extensions to additional workflows and perturbations.

Experiments on nine models show that mixed imperfections reduce pooled accuracy from 76.5% to 60.2%. Even on matched, information-sufficient successes, mean extra tool calls rise from 0.21 to 0.76 and execution time increases by 41.0%. Correct outputs can conceal unresolved requirements, while low tool overhead does not imply selective questioning. These findings motivate agents that better recognize when clarification is needed and incorporate user feedback with less unnecessary effort. Evaluation should therefore consider correctness, information sufficiency, and efficiency jointly.

Limitations. EA-Bench focuses on a subset of work-related tasks and provides less coverage of scientific and coding workflows. We will further explore task diversity across these domains. Our current evaluation setup includes Codex-based runs. We are also deploying evaluations with other agent harnesses to assess whether our findings generalize across execution frameworks.

AI USE STATEMENT

We used generative AI tools for translation, language editing, literature synthesis, analysis and plotting code, statistical checks, and discussions of methodology, experimental design, and result interpretation. The authors reviewed the AI-assisted outputs and take responsibility for the final content, including all text, claims, and artifacts.

REPRODUCIBILITY STATEMENT

Evaluation configurations, task objectives, and evaluation protocols are documented in Sections 4–5 and Appendices A–D and F. We will open-source the evaluation code and make the complete EA-Bench dataset publicly available upon paper acceptance.

REFERENCES

Reyna Abhyankar, Qi Qi, and Yiying Zhang. OSWorld-Human: Benchmarking the efficiency of computer-use agents. *Proceedings of Machine Learning and Systems*, 8:482–494, 2026.

Anthropic. Claude Opus 5.5 System Card. System card, Anthropic, September 2026. URL https://www.anthropic.com/claude-opus-5-5-system-card. Published September 22, 2026.

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.

Junzhi Chen, Harsh Trivedi, Jane Pan, Michael JQ Zhang, Tejas Srinivasan, Niranjan Balasubramanian, and Ashish Sabharwal. AppWorld-UL: Benchmarking diverse agent-user interactions for tool-use. *arXiv preprint arXiv:2607.20536*, 2026.

Shihan Dou, Ming Zhang, Zhangyue Yin, Chenhao Huang, Yujiong Shen, Junzhe Wang, Jiayi Chen, Yuchen Ni, Junjie Ye, Cheng Zhang, et al. CL-bench: A benchmark for context learning. *arXiv preprint arXiv:2602.03587*, 2026.

Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, Yang Zhang, and Boris Ginsburg. RULER: What's the real context size of your long-context language models? *arXiv preprint arXiv:2404.06654*, 2024.

Dong Huang, Yuhao Qing, Weiyi Shang, Heming Cui, and Jie Zhang. EffiBench: Benchmarking the efficiency of automatically generated code. *Advances in Neural Information Processing Systems*, 37:11506–11544, 2024.

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

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. SWE-bench: Can language models resolve real-world GitHub issues? In *International Conference on Learning Representations*, volume 2024, pp. 54107–54157, 2024.

Sayash Kapoor, Benedikt Stroebl, Zachary S Siegel, Nitya Nadgir, and Arvind Narayanan. AI agents that matter. *arXiv preprint arXiv:2407.01502*, 2024.

Yuetai Li, Yichen Feng, Zhangchen Xu, Zixian Ma, Kaiyuan Zheng, Fengqing Jiang, Xinghua Sun, Rulin Shao, Zichen Chen, Yue Huang, et al. JobBench: Aligning agent work with human will. *arXiv preprint arXiv:2605.26329*, 2026. URL https://arxiv.org/abs/2605.26329.

Jiarui Lu, Thomas Holleis, Yizhe Zhang, Bernhard Aumayer, Feng Nan, Haoping Bai, Shuang Ma, Shen Ma, Mengyu Li, Guoli Yin, et al. ToolSandbox: A stateful, conversational, interactive evaluation benchmark for LLM tool use capabilities. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pp. 1160–1183, 2025.

Chang Ma, Junlei Zhang, Zhihao Zhu, Cheng Yang, Yujiu Yang, Yaohui Jin, Zhenzhong Lan, Lingpeng Kong, and Junxian He. AgentBoard: An analytical evaluation board of multi-turn LLM agents. *Advances in Neural Information Processing Systems*, 37:74325–74362, 2024.

OpenAI. GPT-6 Astra System Card. System card, OpenAI, September 2026. URL https://deploymentsafety.openai.com/gpt-6-astra. Published September 3, 2026; updated September 22, 2026.

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 *Forty-second International Conference on Machine Learning*, 2025.

Cheng Qian, Zuxin Liu, Akshara Prabhakar, Zhiwei Liu, Jianguo Zhang, Haolin Chen, Heng Ji, Weiran Yao, Shelby Heinecke, Silvio Savarese, et al. UserBench: An interactive gym environment for user-centric agents. *arXiv preprint arXiv:2507.22034*, 2025.

Preethi Seshadri, Samuel Cahyawijaya, Ayomide Odumakinde, Sameer Singh, and Seraphina Goldfarb-Tarrant. Lost in simulation: LLM-simulated users are unreliable proxies for human users in agentic evaluations. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 47423–47439, 2026.

Olly Styles, Sam Miller, Patricio Cerda-Mardini, Tanaya Guha, Victor Sanchez, and Bertie Vidgen. WorkBench: A benchmark dataset for agents in a realistic workplace setting. *arXiv preprint arXiv:2405.00823*, 2024. URL https://arxiv.org/abs/2405.00823.

Hari Subramonyam, Roy Pea, Christopher Pondoc, Maneesh Agrawala, and Colleen Seifert. Bridging the gulf of envisioning: Cognitive challenges in prompt based interactions with LLMs. In *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems*, pp. 1–19, 2024.

Harsh Trivedi, Tushar Khot, Mareike Hartmann, Ruskin Manku, Vinty Dong, Edward Li, Shashank Gupta, Ashish Sabharwal, and Niranjan Balasubramanian. AppWorld: A controllable world of apps and people for benchmarking interactive coding agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 16022–16076, 2024.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. *arXiv preprint arXiv:2406.12045*, 2024.

Howard Yen, Tianyu Gao, Minmin Hou, Ke Ding, Daniel Fleischer, Peter Izsak, Moshe Wasserblat, and Danqi Chen. HELMET: How to evaluate long-context language models effectively and thoroughly. *arXiv preprint arXiv:2410.02694*, 2025.

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

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

All empirical figures use the current nine-service snapshot. Tables explicitly labeled historical retain earlier two-GPT diagnostics for traceability; their raw-grader counts and retry results are not the primary annotation-aware scores.

A Task Catalog and Source Provenance

The eighteen families below are synthetic adaptations. The source group identifies the scenario reference, not the execution environment or official benchmark membership. Each family contributes fifteen conditions, and each source group contributes ninety conditions. All names, numerical instances, record collections, and canonical artifacts used by EA-Bench belong to its controlled task construction.

Table A1: Complete family catalog. K is the business-action minimum; depth refers to the prescribed graph.

Family ID	Scenario	Source inspiration	K	Depth
workforce	Staffing and payroll	JobBench	6	5
procurement	Procurement and costing	JobBench	7	5
training	Training allocation	JobBench	7	5
reconciliation	Payment reconciliation	JobBench	7	5
donor_registry	Donor registry	JobBench	8	5
inventory_replenishment	Inventory replenishment	JobBench	8	6
aw_note_mail	Note export and mail	AppWorld-UL	6	5
aw_artist_follow	Artist recommendations	AppWorld-UL	7	5
aw_trip_checkout	Trip checkout	AppWorld-UL	7	5
aw_flight_alarms	Flight reminders	AppWorld-UL	7	5
aw_music_mail	Music selection and mail	AppWorld-UL	7	6
aw_order_returns	Order returns	AppWorld-UL	6	5
wb_mail_cleanup	Mail retention	WorkBench	4	4
wb_calendar_shift	Calendar changes	WorkBench	5	5
wb_crm_transfer	CRM transfer	WorkBench	6	5
wb_project_handoff	Project handoff	WorkBench	6	5
wb_analytics_plot	Analytics aggregation	WorkBench	6	6
wb_capacity_lead	Capacity-based assignment	WorkBench	9	7

The manifest records a source locator, adaptation description, source hash, family identifier, graph, and policy metadata for every episode. JobBench references are pinned to a dataset revision, WorkBench templates to a repository commit and zero-based CSV row, and AppWorld-UL references to the supplied arXiv version and appendix scenarios. For AppWorld-inspired families, the original scenario deficiency and its resolution in Full are recorded separately from the four new perturbable policy topics. This prevents an inherited scenario description from being mistaken for the current information manipulation.

No AppWorld application databases or production office accounts are executed. Read tools return local snapshots; written artifacts are evaluated locally. State-preservation checks are relevant for tasks such as email retention and calendar modification because a correct target change must not alter unrelated records. Source licensing and attribution are described in the repository's THIRD_PARTY.md; this manuscript adds no claim of endorsement by those projects.

B Perturbation Specification

Missingness removes the authoritative public statement; it does not remove the concept from every schema or tool description. A model may infer which kinds of facts matter from the task structure, but the alternative-world check prevents the actual hidden value from being distinguished by public records. Conflict variants contain opposed policy statements without a decisive authority ordering. Missing and conflicting topics do not overlap in a mixture, so its information lower bound is their sum.

Table A2: The fifteen conditions. M and C are counts of affected policy topics. R is the target ratio of added serialized UTF-8 bytes to the Full representation. Each condition has eighteen instances.

Condition	Missing topics	Conflicting topics	Added-byte target	Required facts
Full	0	0	0	0
M1	1	0	0	1
M2	2	0	0	2
M3	3	0	0	3
M4	4	0	0	4
R1	0	0	0.5	0
R2	0	0	1.0	0
R3	0	0	2.0	0
R4	0	0	4.0	0
C1	0	1	0	1
C2	0	2	0	2
C3	0	3	0	3
C4	0	4	0	4
M1-R1-C1	1	1	0.5	2
M2-R2-C2	2	2	1.0	4

Redundancy is operationalized through related auxiliary material, including familiar entities and plausible document context. The target byte ratio divides the added material by the size of the Full representation. Serialization details and discrete records can affect the realized ratio; the manifest retains the observed value. We do not assume equal tokenizer cost or equal difficulty across models. In particular, a fourfold added-byte target is not a claim that the model receives exactly four times as many tokens.

The four policy topics are fixed within each family, and conditions share the business records and authoritative values. These controls support descriptive paired comparisons but do not create independent samples. A stronger study of robustness would vary records, policy values, question wording, and repetitions while respecting family-level dependence. Such an experiment is not included in the present manuscript.

C Lower-Bound Derivation and Metric Semantics

C.1 Business actions and user rounds

Proposition 1 (interface-relative attainability). Suppose the report depends on K necessary business nodes, each action can materialize at most one node, and a current valid handle is required for every prerequisite. Any successful execution requires at least K business actions. If all required user facts can be requested together and returned correctly, the additional question-round minimum is the indicator that at least one fact is unresolved.

Argument. Every necessary node must have been materialized to provide the required handle. A single action cannot satisfy two distinct node-generation requirements, giving the lower bound K. A topological traversal that computes each artifact correctly once reaches the report in K actions. No question is needed when the public input resolves all necessary policies. Otherwise one batched question obtains the required facts under the functioning-oracle assumption. The sum of these operations attains the stated total. The reference trajectories instantiate this construction for the released finite tasks.

This argument does not give an unrestricted computational lower bound. A different API could expose multiple records or artifacts in a single call, and an unconstrained program could combine operations. Even a perfect action sequence incurs network delay, artificial pacing, and user-model overhead, so elapsed time requires separate measurement.

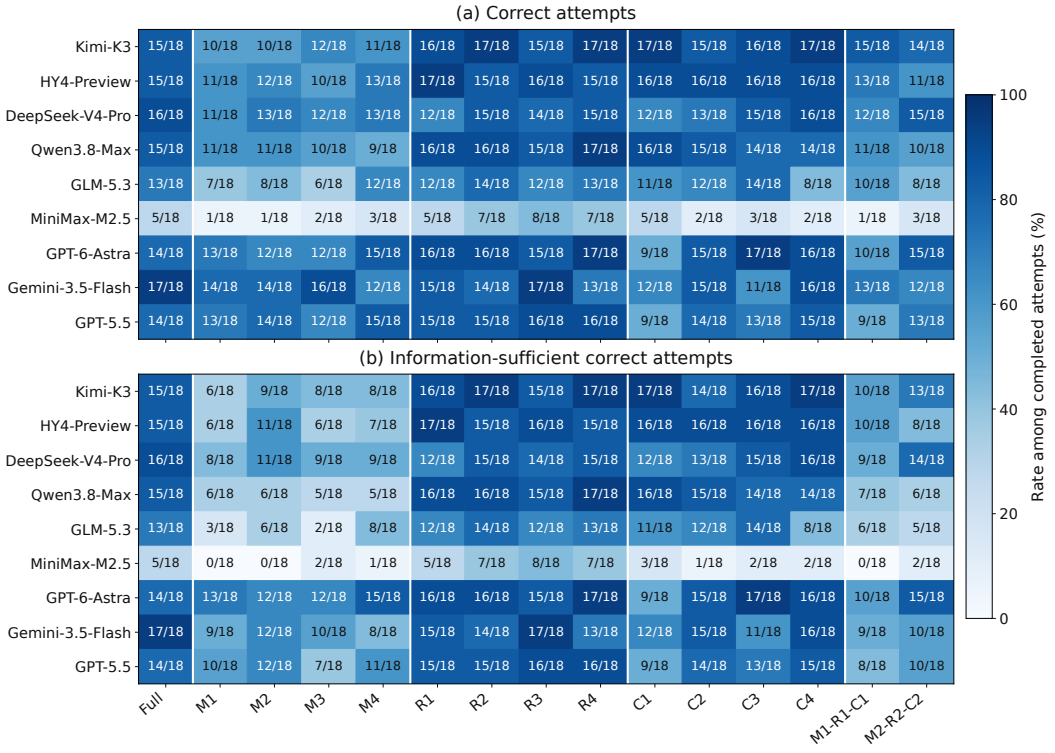

Figure 6: All nine models across all fifteen conditions. Top: correct attempts. Bottom: information-sufficient correct attempts. Each cell reports successes out of 18 tasks. Runtime and annotation errors count as failures.

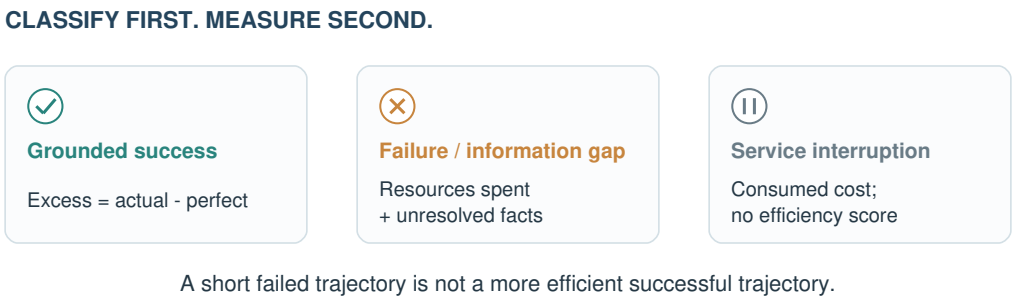

Figure 7: Outcome-aware measurement gate. Successful-task excess is measured against a perfect baseline; failed and interrupted attempts retain their costs without receiving an efficiency advantage for early termination.

C.2 Necessary information and outcome correctness

Proposition 2 (indistinguishable policy worlds). For each required policy topic, suppose the release provides two worlds with identical public observations and different correct output artifacts, differing in that topic's value. A policy that does not obtain distinguishing information about the topic cannot guarantee the correct artifact in both worlds.

Argument. Before acquiring distinguishing information, the observations available to the policy are identical. Its output distribution is therefore the same in both worlds, although a single artifact cannot satisfy both distinct correct outputs. Guaranteed success requires resolving the distinction. Applying the construction to each required topic motivates counting all required independent facts. The guarantee concerns the constructed alternative worlds. A model can still guess correctly in the fixed authoritative world used for a run.

The release stores topic-level counterfactual witnesses and checks visibility equality. The offline validation records 468 rejections of submissions computed under wrong required interpretations. These checks support the intended information requirement; they do not establish that the natural-language scope classifier recognizes every legitimate paraphrase.

C.3 Excess counts and missing annotation

Let Q count atomic information requests across turns and U count distinct required facts both requested and returned. Repeated requests can increase Q without increasing U. A fully informed trajectory has $U = I^*$; its information excess is $Q - I^*$. We average this excess over eligible successes, including tasks with $I^* = 0$. Tool excess is analogously $A - A^*$. Failed or insufficiently informed trajectories do not enter either successful-task mean; their expenditure and unresolved requirements are reported separately.

Strict information accounting requires every `ask_user` attempt to appear in annotation. An omitted or malformed question excludes the record from the strict information subset, even when its tool trajectory remains eligible. The nine-model comparison uses 20 shared tool tasks and 17 shared strictly annotated information tasks. Every actual/minimum comparison uses identical tasks. Missing annotations and empty subsets are not replaced by zero.

D Detailed Results Across Models

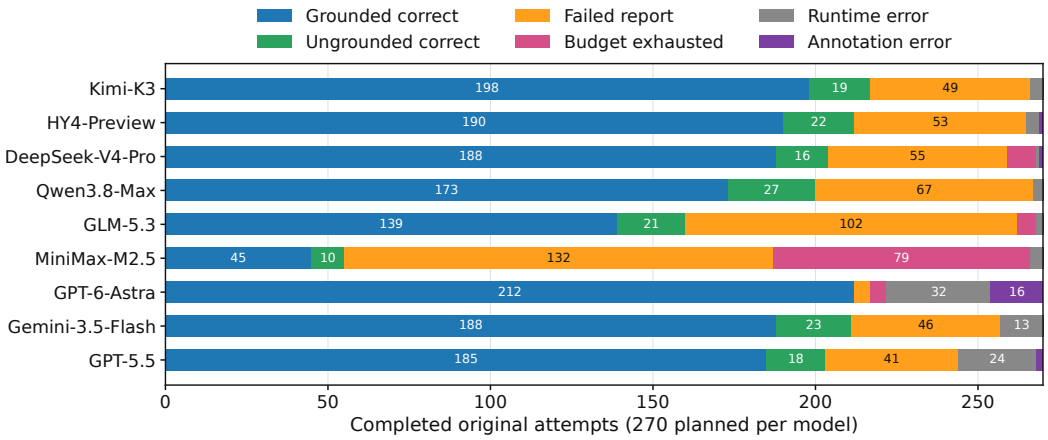

Figure 8: Outcomes of all 2,430 original attempts. Each model has 270 attempts. Runtime errors are assigned first, then annotation errors, then graded outcomes. Both error groups count as failures in Table 2.

The nine-model intersection covers 20/270 tasks for actions and 17/270 for information. Episode IDs are released in evidence/stratified_analysis.json. Pairwise comparisons retain a separate intersection and sample size for each of the 36 model pairs. Model-specific success counts appear in Table 3.

Table A3: Matched mean extra consumption across nine complete runs. All rows use the same 20 tool tasks and 17 strictly annotated information tasks.

Model	Calls/min.	Extra calls/task	Facts/min.	Extra facts/task
Kimi-K3	127/122	0.25	22/11	0.65
HY4-Preview	122/122	0.00	13/11	0.12
DeepSeek-V4-Pro	135/122	0.65	12/11	0.06
Qwen3.8-Max	127/122	0.25	13/11	0.12
GLM-5.3	133/122	0.55	11/11	0.00
MiniMax-M2.5	159/122	1.85	12/11	0.06
GPT-6-Astra	128/122	0.30	32/11	1.24
Gemini-3.5-Flash	126/122	0.20	11/11	0.00
GPT-5.5	124/122	0.10	14/11	0.18

Table A4: Current source-group results for all nine services. Counts retain actual coverage. Source labels indicate scenario provenance, not official benchmark scores.

Source	Model	n	Correct	Grounded
jobbench	Kimi-K3	90	69	65
appworld_ul	Kimi-K3	90	64	56
workbench	Kimi-K3	90	84	77
jobbench	HY4-Preview	90	66	59
appworld_ul	HY4-Preview	90	58	52
workbench	HY4-Preview	90	88	79
jobbench	DeepSeek-V4-Pro	90	64	59
appworld_ul	DeepSeek-V4-Pro	90	53	47
workbench	DeepSeek-V4-Pro	90	87	82
jobbench	Qwen3.8-Max	90	63	52
appworld_ul	Qwen3.8-Max	90	54	46
workbench	Qwen3.8-Max	90	83	75
jobbench	GLM-5.3	90	42	39
appworld_ul	GLM-5.3	90	45	41
workbench	GLM-5.3	90	73	59
jobbench	MiniMax-M2.5	90	9	7
appworld_ul	MiniMax-M2.5	90	15	12
workbench	MiniMax-M2.5	90	31	26
jobbench	GPT-6-Astra	90	72	72
appworld_ul	GPT-6-Astra	90	67	67
workbench	GPT-6-Astra	90	73	73
jobbench	Gemini-3.5-Flash	90	65	56
appworld_ul	Gemini-3.5-Flash	90	64	56
workbench	Gemini-3.5-Flash	90	82	76
jobbench	GPT-5.5	90	70	60
appworld_ul	GPT-5.5	90	55	52
workbench	GPT-5.5	90	78	73

E OUTCOME-STRATIFIED RESOURCE ACCOUNTING

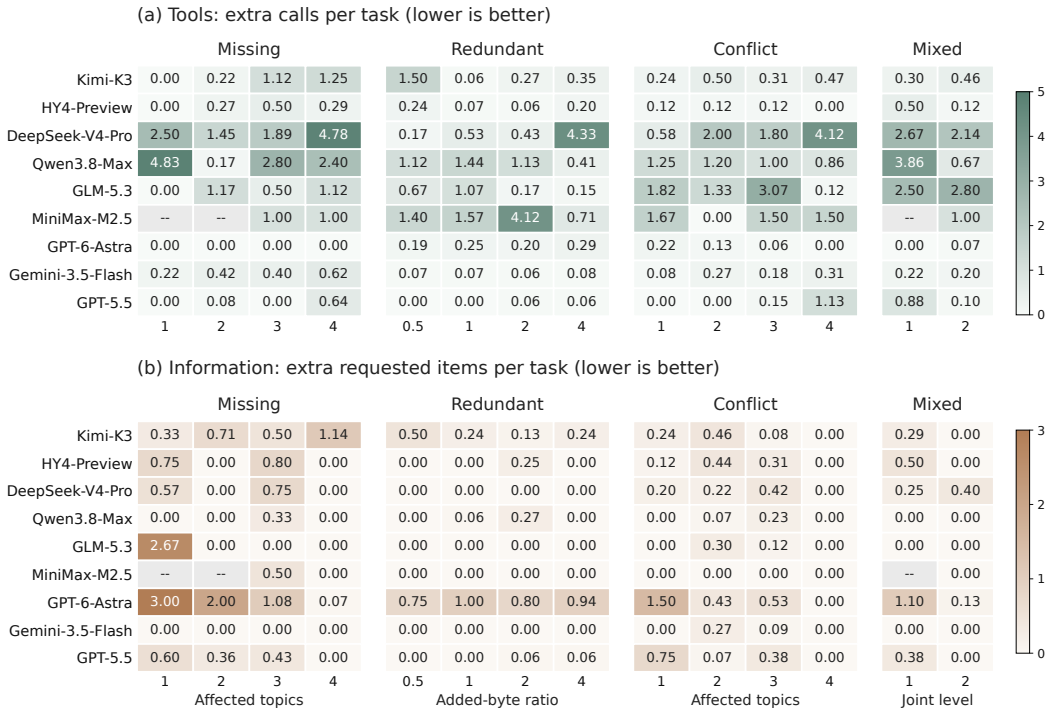

Figure 9: Mean extra consumption per grounded-success task: Tools above, Information below. Darker cells indicate more overhead; values are counts per task, not percentages. Families are fixed within each model and perturbation as in Figure 4, while successful subsets may differ by level. Gray – marks no eligible successes. Non-Full levels are shown; Full is reported in Table 3.

The figures in this appendix use the current snapshot. The following legacy resource tables retain the historical GPT raw-grader outcome partition; they are supplementary trace diagnostics and do not replace the nine-model comparisons.

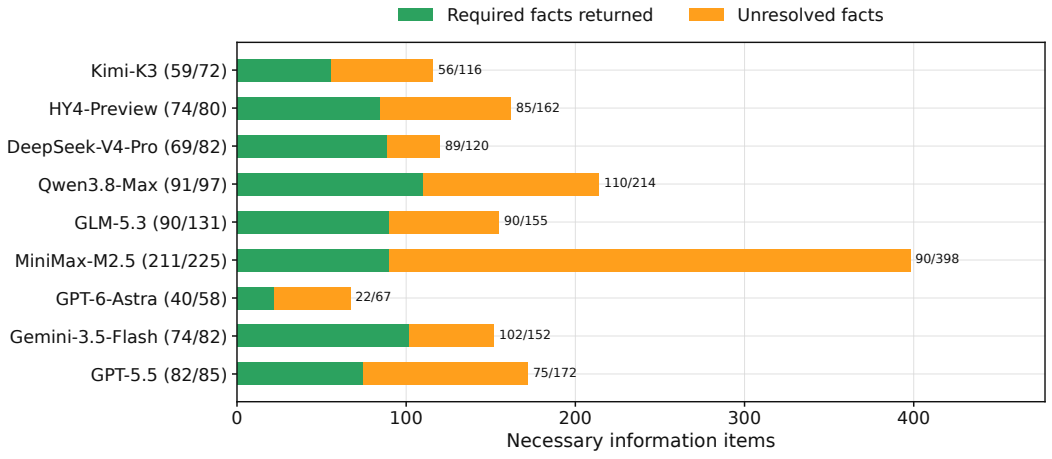

Figure 10: Required facts returned and unresolved across all non-grounded attempts with complete question accounting. All nine services are shown; labels give audited/total non-grounded attempts. Counts are restricted to required topics in returned simulator answers. This describes information state, not the cause of failure.

Table A5: Outcome-stratified means. P / A denotes perfect / actual; time has no established perfect baseline. Empty is the total empty-turn count.

Model	Outcome	n	Actions P / A	Rounds P / A	Seconds	Empty
GPT-6-Astra	Grounded success	219	7.24 / 7.38	0.63 / 0.76	42.54	0
GPT-6-Astra	Failed report	5	7.40 / 7.40	0.60 / 0.60	42.43	0
GPT-6-Astra	Budget exhausted	14	8.00 / 40.00	0.79 / 3.93	238.59	391
GPT-6-Astra	Runtime error	32	7.22 / 2.19	0.84 / 1.00	19.76	0
GPT-5.5	Grounded success	186	7.13 / 7.33	0.59 / 0.63	42.28	0
GPT-5.5	Ungrounded success	18	7.67 / 7.67	1.00 / 0.78	44.73	0
GPT-5.5	Failed report	42	7.79 / 7.67	0.74 / 0.57	43.61	0
GPT-5.5	Runtime error	24	7.21 / 2.33	0.88 / 1.04	15.63	0

GPT-6-Astra's five failed submissions use the same average action count as the group's perfect baseline, but correctness still fails. GPT-5.5's forty-two failed submissions consume slightly fewer actions than their group's baseline. Some required user interaction is absent and the outputs are not accepted, so the lower action count does not establish efficient completion. In the eighteen GPT-5.5 outcome successes without grounding, the mean action total happens to equal the full-information baseline, although question rounds and acquired facts fall short. Extra business actions can offset missing user interaction in the total, making aggregate action equality particularly misleading.

Table A6: Strict information audit. All item totals use the usable subset in column three, including the required-information baseline.

Model	Outcome	Usable / n	Required	Requested	Obtained	Unresolved
GPT-6-Astra	Grounded success	208/219	358	554	358	0
GPT-6-Astra	Failed report	5/5	9	12	9	0
GPT-6-Astra	Budget exhausted	5/14	10	14	10	0
GPT-6-Astra	Runtime error	30/32	48	81	3	45
GPT-5.5	Grounded success	183/186	288	318	288	0
GPT-5.5	Ungrounded success	18/18	52	26	22	30
GPT-5.5	Failed report	40/42	83	53	48	35
GPT-5.5	Runtime error	24/24	37	43	5	32

For the forty GPT-5.5 failed submissions with strict information records, nineteen have unresolved required information and twenty-one do not. The table reports thirty-five unresolved facts in aggregate. These are facts missing under the recorded request-and-answer accounting, not thirty-five independently diagnosed causes. The two excluded failed records do not inherit the coverage of the other forty. One excluded trajectory uses malformed nested question fields and another lacks complete annotation.

GPT-6-Astra's empty turns appear only in the fourteen budget-exhausted original trajectories. The original action-count difference between models is 2283 minus 1880, or 403; the 391 empty turns constitute 97.0% of that numerical difference. This is an accounting observation, not a causal explanation of the remaining costs. Removing the empty turns would leave failed outputs failed, and the two models' successful-task sets would still differ. We therefore do not publish an adjusted success efficiency obtained by simply deleting empty turns.

Across all attempts, accumulated episode time is 13501.83 seconds for GPT-6-Astra and 10876.72 for GPT-5.5. Dividing these totals by outcome successes gives 61.65 and 53.32 seconds per produced success. Requiring grounded success changes the latter to 58.48 seconds while GPT-6-Astra remains at 61.65. This system-level measure includes failure expenditure and answers a different question from the paired successful-trajectory means of 41.73 and 42.52 seconds. Neither measure is the elapsed wall-clock duration of the entire parallel experiment.

F Interaction Examples and Prompt Contract

F.1 Successful narrow and broad questions

The following is an English translation of the original aw_artist_follow–M1 questions. The actual task and conversation are in Chinese. Both models succeed with the required information. The missing decision is the recommendation time window; the other listed policies are already available in this condition.

GPT-6-Astra, translated. Please confirm the recommendation window in days, including recommendations whose age_days is at most that window, and confirm inclusion of email and SMS, exclusion of muted artists, and retention of existing follows while adding valid recommendations. If there are conflicting business opinions, please resolve them together.

GPT-5.5, translated. Please confirm the age_days time window for including recommendations in this batch. Recommendations with age_days less than or equal to the window are included.

The comparison illustrates information burden within a single turn. A model may use broad confirmation as a strategy for managing an unreliable user channel. The current paired evidence establishes the difference in requested facts, while controlled experiments would be needed to identify the reason for that behavior.

F.2 Correct and malformed question schemas

The supported user request uses a top-level question field. The following English question is schematic.

```
{
  "op": "ask_user",
  "question": "What time window applies?"
}
```

A saved GPT-5.5 calendar trajectory instead initially places the question under data. The example below preserves that structure while translating and shortening the text.

```
{
  "op": "ask_user",
  "data": {
    "question": "Which series scope and delay apply?"
  }
}
```

The malformed action is rejected. A later correctly structured request obtains the information and the task succeeds. The first action still consumes a turn, but its question is absent from the original atomic annotation list. Our strict information comparison excludes the pair; action accounting retains both attempts. This example explains the 154-to-153 information-sample change without changing any original outcome.

F.3 Evaluated-agent instruction

The evaluator's instruction is reproduced here in English translation. The original Chinese file is bundled separately for exact inspection; the translation is explanatory, not the text submitted to the models.

Complete the given business task using the supplied JSON action interface to read business data, communicate with the user, and prepare business artifacts. Output only one JSON action at a time, without Markdown, explanations, or batched actions. Follow the action format in the task's tool instructions. The report action is the only final submission; the task ends immediately after it, whether the answer is correct or incorrect, and it cannot be revised or resubmitted. Before final submission, intermediate plans may be checked and adjusted. Do not use the shell, filesystem, network, subagents, or other tools.

This contract makes format and dependency compliance part of the observable execution task. A terminal report rejected for its structure remains a submission failure, but a failure taxonomy should distinguish it from an incorrect business calculation. The present paper does not infer that distinction for every failure from a single aggregate status.

F.4 User scope and post-run information annotation

At a conceptual level, the scope stage identifies requested catalog topics from the current question; the host verifies the evidence before selecting answer values. The post-run annotation stage identifies independent facts explicitly requested, excludes background mentions, merges repetitions within one turn, and preserves cross-turn repetitions. A broad request for all rules is not automatically expanded into every hidden fact. Authoritative values are supplied only through the answer path, and post-run annotation cannot send a hint back to the evaluated agent.

These paragraphs describe the intended protocol. Exact versioned behavior is constrained by the run's implementation hashes. Current source files may include later fixes, so rebuilding a live experiment requires matching the historical version or explicitly declaring a new version. The manuscript's analysis uses saved metrics and trajectories and makes no new model-service calls.

G One-Retry Sensitivity and Cost

G.1 Historical GPT error retries

The original run contains 32 GPT-6-Astra and 24 GPT-5.5 runtime-error conditions. A separate experiment retries each of these once with the recorded configuration and budget. Normal task failures are not retried. The retry ledger retains successes, new task failures, and repeated interruptions; it does not keep sampling until success.

Table A7: Outcomes of the 56 additional attempts. This is a retry experiment, not another 540-attempt single-run evaluation.

Model	Retry attempts	Success	Task failure	Still interrupted
GPT-6-Astra	32	16	2	14
GPT-5.5	24	12	2	10

In a condition-level view that accepts a valid second attempt for an originally interrupted condition, GPT-6-Astra has 235 successes, 21 task failures, and 14 unresolved conditions. GPT-5.5 has 216 successes, 44 task failures, and 10 unresolved conditions; 197 of its successes meet the recorded grounding criterion. The view permits additional attempts and must not replace the original 219/270 and 204/270 single-attempt success fractions. It shows that some original errors are recoverable under the same nominal setup while leaving a substantial unresolved remainder.

Table A8: Total consumed resources including every original and retry attempt, including unsuccessful retries. Times are accumulated episode seconds.

Model	Physical attempts	Action attempts	Question rounds	Seconds
GPT-6-Astra	302	2473	290	14591.98
GPT-5.5	294	2010	205	11603.84

Dividing these totals by grounded-success outputs gives 10.52 and 10.20 action attempts per grounded output, and 62.09 and 58.90 seconds per grounded output. These ratios are system-cost summaries. They are not successful-trajectory means, do not establish token or monetary cost, and do not estimate the result of unrestricted retries. The later simulator repair is not used to reinterpret either experiment.

G.2 QIANWEN SERVICE RECOVERY RETRIES

At the 2026-09-26 18:33 China-time snapshot, MiniMax-M2.5 has three selected recovery retries with no correct results, and Qwen3.8-Max has three with three correct results. Only runtime errors were selected. These attempts remain separate from the 2,430 primary attempts and do not replace original errors.

H REPRODUCIBILITY AND EVIDENCE BOUNDARIES

H.1 DATA LINEAGE

The primary source is the frozen nine-service snapshot and its per-episode metrics. All empirical figures are regenerated from the current snapshot; explicitly historical GPT resource tables retain their earlier 540-attempt accounting. Every original metrics file was checked against its recorded SHA-256 before analysis. The released manifest supplies family identity, source group, condition, and resource lower bounds; the analysis checks that metric-level lower bounds match the manifest. In that historical GPT subset, the five outcome groups sum to 270 per model; grouped resource totals reconcile with the original records.

The publication package contains original_rows.json, the original metric hash map, the release manifest, outcome tables, perfect-baseline tables, condition/source summaries, reference validation, and the structural audit. Exact run configurations and original traces remain in the workspace run directories identified in the package README. The package is sufficient to inspect the reported aggregate values and rebuild the figures; rerunning the models additionally requires the appropriate historical code and service access.

The current tables and plots are generated from evidence/current_results_snapshot.json, which stores completed metrics, condition labels, full-question-coverage flags, and source-file hashes. Compilation uses this fixed snapshot. evidence/harness_comparison.json records plan hashes and the shared-source changes between historical GPT and later service runs; the release-manifest hash is identical. Model-family classification was checked on 26 September 2026 using the official DeepSeek-V4-Pro-0813 release, GLM-5.3 model card, Kimi-K3 model card, and Qwen3.8-2.4T-A95B model card. The latter explicitly identifies Qwen3.8-Max as the official hosted version based on those weights with additional features. MiniMax-M2.5 is described as open-source by the Qianwen model page, and is now included in the snapshot. HY4-Preview is released under Apache 2.0 in the official HY4 repository. These are weight-availability labels, not claims of open training data, permissive licensing, or exact hosted-checkpoint identity.

H.2 REBUILDING THE PAPER

The editable source is paper.md. analyze_results.py verifies source hashes and released lower bounds, then builds the stratified CSVs, figures, and evidence/stratified_analysis.json from a frozen snapshot, and a typesetting script compiles the manuscript with the supplied, unmodified ICLR 2027 style files. The editable manuscript is exported to LaTeX and compiled with a real TeX engine. The conceptual diagrams are additionally supplied as editable draw.io files. Figures are supplied in both raster PNG and vector PDF. Tables are also supplied in CSV or JSON so that values can be checked independently of the layout.

The PDF is an anonymous research draft using the supplied ICLR 2027 style. All nine model services completed 270 conditions. The primary cohort contains exactly 2,430 attempts with no imputed cells; recovery retries remain separate. Gemini uses only its completed native LOW run.

H.3 WHAT HAS AND HAS NOT BEEN VALIDATED

Reference validation checks finite-task behavior; hashes identify the evidence; group reconciliation checks numerical consistency. These checks do not independently validate every Qwen judgment, identify the internal implementation behind a model API, or establish general occupational competence. The evidence supports explicit task baselines and observed differences in success, information collection, and failure expenditure. Simulator-independent capability and deployment-cost claims require additional evidence.

I RELEASED TASKS AND ABRIDGED MODEL RESPONSES

These cases are abridged English translations of actual released tasks and saved model interactions, not idealized dialogues. Structured values are copied from frozen evidence; projections and omissions are stated. The field guides distinguish shared protocol keys from task-specific records and output keys.

I.1 HOW TO READ THE RECORDS

Reading guide

Natural language. Task and interaction boxes are translated excerpts.
Structured data. JSON blocks preserve the selected source values. Read-tool excerpts are keyed by operation name for presentation.
Field tables. Each displayed key has a type, meaning and example. Item and supplier IDs are dynamic map keys, not new schema fields.

Table A9: Shared interaction and execution keys

Key	Type	Meaning / constraint	Example
episode	string	Trial ID combining task family and condition.	procurement--M1
model	string	Recorded evaluated-service identifier.	gpt-6-astra
op	string	Requested operation; the first report ends the episode.	read_quotes; report
question	string	Clarification text, present for ask_user.	Is Gold required?
inputs	object	Maps prerequisite operations to current artifact handles.	audit: <handle>
inputs.<operation>	string	Host-issued opaque handle; upstream edits invalidate dependent artifacts.	32-character handle
data	object / array	Submitted payload; schema depends on the operation.	{total_cost: 120}
step	integer	One-based position in the execution trace.	1
model_text	string	Visible model reply before JSON parsing.	JSON action text
action	object	Parsed op, inputs, data or question.	{op: read_stock}
result	object	Host response, including observations or errors.	handle + data
handle	string	Versioned identifier for a stored artifact.	<current handle>
result.data.records	object / array	Business records returned by a read operation.	Supplier quote rows
result.data.documents	array	Auxiliary documents returned with read results.	Warehouse label layouts
documents[].name	string	Name of a policy or auxiliary document.	Policy correspondence
documents[].text	string	Document content visible to the model.	Batch policy statement

Table A10: Outcome and resource metric keys

Key	Type	Meaning / constraint	Example
status	string	Raw execution outcome: success, failed submission, budget exhaustion or runtime error.	success
success	boolean	Raw deterministic grader result; reported accuracy also rejects annotation errors.	true
grounded_success	boolean	Raw success plus recorded acquisition of all required facts.	true
information_metrics_status	string	complete is usable; needs_review is excluded from strict information efficiency; annotation_error counts as failure.	complete; needs_review
tool_calls	integer	Action-attempt count, including user requests and malformed turns.	8
actions	integer	Episode-runner alias of tool_calls in these records.	8
optimal_tool_calls	integer	Perfect action count for the same task condition.	8
actual_requested_information_items	integer	Atomic facts explicitly requested; repeat requests are counted again.	4
optimal_information_items	integer	Minimum number of additional policy facts needed.	1

I.2 MISSING INFORMATION IN PROCUREMENT

`procurement--M1` **GPT-6-Astra**

01 | Task and visible evidence

Task excerpt. Select one supplier per item, buy exactly the requested quantities, and minimize total cost. Certification and phased-delivery requirements follow the batch agreement.
Selected records. Requested quantities: 4 desks, 6 chairs, 2 cabinets. Quotes expose certification, installation, freight, warranty, and delivery attributes.
Information condition. Gold certification is not specified. Phased delivery, freight charged once per supplier, and installation charged separately are already stated.

02 | Recorded interaction

Agent question. Is Gold certification mandatory? Also confirm phased delivery, freight once per supplier, and installation charged per item.
User answer. Gold is mandatory; phased delivery is required; freight is grouped by supplier; installation is additional.

03 | Final report payload


```
{
  "total_cost": 120,
  "units": 12
}
```


All requested quantities and supplier V1 are shown. Other quotes and supplier histories are omitted. The submitted result is complete.

04 | Outcome and audit

Result and reference cost. Correct and information-sufficient. 8 actions / 8 required; 4 information requests / 1 necessary fact.
Interpretation. The agent resolves the missing rule but also reconfirms three facts already provided. Correctness and information efficiency therefore differ.

RECORD EXCERPT AND FIELD GUIDE

`procurement--M1` — *input / output schema*

Selected read-tool records


```
{
  "read_requisition": {"cabinet": 2, "chair": 6, "desk": 4},
  "read_quotes": [
    {
      "id": "V1",
      "prices": {"desk": 8, "chair": 6, "cabinet": 10},
      "quoted_qty": {"desk": 4, "chair": 5, "cabinet": 2},
      "freight": 8,
      "install": 2,
      "gold": true,
      "phased": true,
      "warranty_months": 24
    }
  ],
  "read_history": {"V1": {"open_safety_claims": 0}}
}
```


All requested quantities and supplier V1 are shown. Other quotes and supplier histories are omitted. The submitted result is complete.

Table A11: Field guide: Missing information in procurement

Key	Type	Meaning / constraint	Example
`read_requisition`	object	Required unit quantities, indexed by item ID.	desk: 4; chair: 6
`read_quotes`	array	Supplier offers; the excerpt retains V1 only.	One supplier record
`read_history`	object	Supplier history, indexed by supplier ID.	V1
`id`	string	Supplier identifier in quote rows.	V1
`prices`	object	Unit prices by item; amounts are CNY.	desk: 8
`quoted_qty`	object	Supplier-proposed quantities; purchase quantities must follow the requisition.	chair: 5
`freight`	number	Fixed charge per supplier under this batch policy.	8
`install`	number	Installation charge per unit; charged separately in this batch.	2
`gold`	boolean	Whether the supplier holds Gold certification.	true
`phased`	boolean	Whether phased delivery is supported.	true
`warranty_months`	integer	Offered warranty length in months.	24
`open_safety_claims`	integer	Unresolved safety claims; a nonzero count disqualifies a supplier.	0
`total_cost`	number	Final total including goods, installation and freight.	120 CNY
`units`	integer	Total purchased units.	12
`<item_id>`	map key	Item key in requisition, prices and quoted quantities.	desk; chair; cabinet
`<supplier_id>`	map key	Supplier key used in the history map.	V1

I.3 REDUNDANT MATERIAL IN INVENTORY PLANNING

`inventory_replenishment--R2` **GPT-5.5**

01 | Task and visible evidence

Task excerpt. Replenish four days of demand. Deduct reserved stock, include safety stock, and buy whole packs. Report purchases and ending surplus.
Selected records. A: on-hand 8, reserved 3, daily sales 3, safety stock 2, pack size 4. B: on-hand 5, reserved 2, daily sales 2, safety stock 1, pack size 3.
Redundant material. Read results also contain warehouse-label layouts, document-cover records, and shelf-photo indexes; these do not determine demand or procurement quantity.

02 | Recorded interaction

Agent-user interaction. No clarification question is issued. The agent proceeds using the policies and records already available.

03 | Final report payload


```
{
  "cost": 240,
  "purchase": {
    "A": 12,
    "B": 6
  },
  "ending_surplus": {
    "A": 3,
    "B": 0
  }
}
```


Business records for both items are shown. Auxiliary documents are omitted; the submitted result is complete.

04 | Outcome and audit

Result and reference cost. Correct and information-sufficient. 8 actions / 8 required; 0 information requests / 0 necessary new facts.
Interpretation. The extra documents increase the material to inspect without creating an information gap. Successful behavior does not require an extra user turn.

RECORD EXCERPT AND FIELD GUIDE

`inventory_replenishment--R2` — *input / output schema*

Selected read-tool records


```
{
  "read_items": {
    "A": {"pack": 4, "price": 10, "safety_stock": 2},
    "B": {"pack": 3, "price": 20, "safety_stock": 1}
  },
  "read_stock": {"A": {"on_hand": 8, "reserved": 3}, "B": {"on_hand":
     5, "reserved": 2}},
  "read_sales": {"A": 3, "B": 2}
}
```


Business records for both items are shown. Auxiliary documents are omitted; the submitted result is complete.

Table A12: Field guide: Redundant material in inventory planning

Key	Type	Meaning / constraint	Example
`read_items`	object	Product attributes, indexed by item ID.	A; B
`read_stock`	object	Current stock and reservations, indexed by item ID.	A; B
`read_sales`	object	Daily unit sales, indexed by item ID; not a total over the planning horizon.	A: 3; B: 2
`pack`	integer	Units per indivisible purchase pack.	A: 4
`price`	number	Price per unit in CNY.	A: 10
`safety_stock`	integer	Required safety-stock units.	A: 2
`on_hand`	integer	Physically available inventory before reservations.	A: 8
`reserved`	integer	Units already committed; subtract from on_hand.	A: 3
`cost`	number	Total cost of purchased units.	240 CNY
`purchase`	object	Purchased unit quantities by item ID, not numbers of packs.	A: 12; B: 6
`ending_surplus`	object	Units left after fulfilling four days of demand.	A: 3; B: 0
`<item_id>`	map key	Same product IDs across item, stock, sales and result maps.	A; B

I.4 CONFLICTING NOTIFICATION INSTRUCTIONS IN SCHEDULING

wb_calendar_shift--C1 **GPT-6-Astra**

01 | Task and visible evidence

Task excerpt. Move the entire S1 series 30 minutes later, preserving duration. Other meetings remain unchanged. Whether to notify attendees must follow the batch decision.
Selected records. E1 and E2 belong to S1 and initially run 10:00–11:30 on October 1 and 2. E3 belongs to another series.
Conflicting evidence. Group A: do not send update notices. Group B: send an update for every changed meeting. Neither statement has priority.

02 | Recorded interaction

Agent question. The whole S1 series will move by 30 minutes with duration preserved. The notification instructions conflict: should every changed meeting receive an update notice?
User answer. Send notifications for every changed meeting. The simulator additionally restates the already specified series scope, delay, and duration policy.

03 | Final report payload


```
{
  "calendar": [
    {
      "id": "E1",
      "start": "2026-10-01 10:30",
      "end": "2026-10-01 12:00"
    }
  ],
  "notifications": [
    {
      "event": "E1",
      "start": "2026-10-01 10:30"
    }
  ]
}
```


The input excerpt retains E1 only. The result projects E1 and its notice onto displayed fields; E2, unchanged E3, series and attendees are omitted. Full evidence retains them.

04 | Outcome and audit

Result and audit. Correct under the task grader, with 6 actions / 6 required. The information annotation is marked needs_review; this attempt is excluded from strict information-efficiency aggregation.
Interpretation. The example resolves the policy conflict, but the extra returned facts expose a simulator-scope issue. The abridged record retains that scope issue for audit.

RECORD EXCERPT AND FIELD GUIDE

`wb_calendar_shift--C1` — *input / output schema*

Selected read-tool records


```
{
  "read_calendar": [
    {
      "id": "E1",
      "series": "S1",
      "start": "2026-10-01 10:00",
      "end": "2026-10-01 11:30",
      "attendees": ["a@example.org", "b@example.org"]
    }
  ]
}
```


The input excerpt retains E1 only. The result projects E1 and its notice onto displayed fields; E2, unchanged E3, series and attendees are omitted. Full evidence retains them.

Table A13: Field guide: Conflicting notification instructions in scheduling

Key	Type	Meaning / constraint	Example
`read_calendar`	array	Original event records; the structured excerpt retains E1.	Event E1
`calendar`	array	Final event records. The original report also contains E2 and unchanged E3.	Updated E1
`notifications`	array	Notification records for changed events; the excerpt retains E1.	Notice for E1
`id`	string	Event identifier in a calendar record.	E1
`series`	string	Recurring-series identifier. Only S1 is targeted.	S1
`start`	string	Start timestamp in YYYY-MM-DD HH:MM format.	2026-10-01 10:30
`end`	string	End timestamp; shifts with start when duration is preserved.	2026-10-01 12:00
`attendees`	string list	Sorted attendee addresses; present in the full report, omitted from the displayed result projection.	a@example.org
`event`	string	Event ID referenced by a notification.	E1

J ALL TASK-FAMILY DAGS

We present workflow diagrams for all 18 task families. The diagrams follow the supplied draw.io source; the underlying task definitions and lower-bound derivations are provided in the benchmark release and Appendix C.

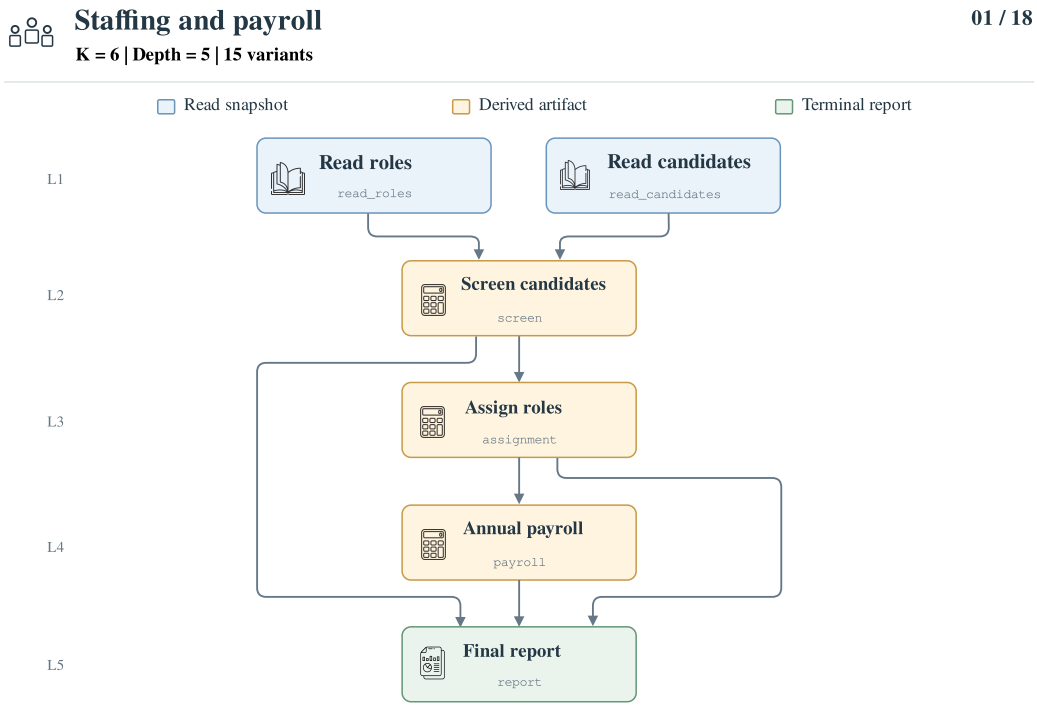

Figure 11: Workflow diagram for Staffing and payroll (workforce).

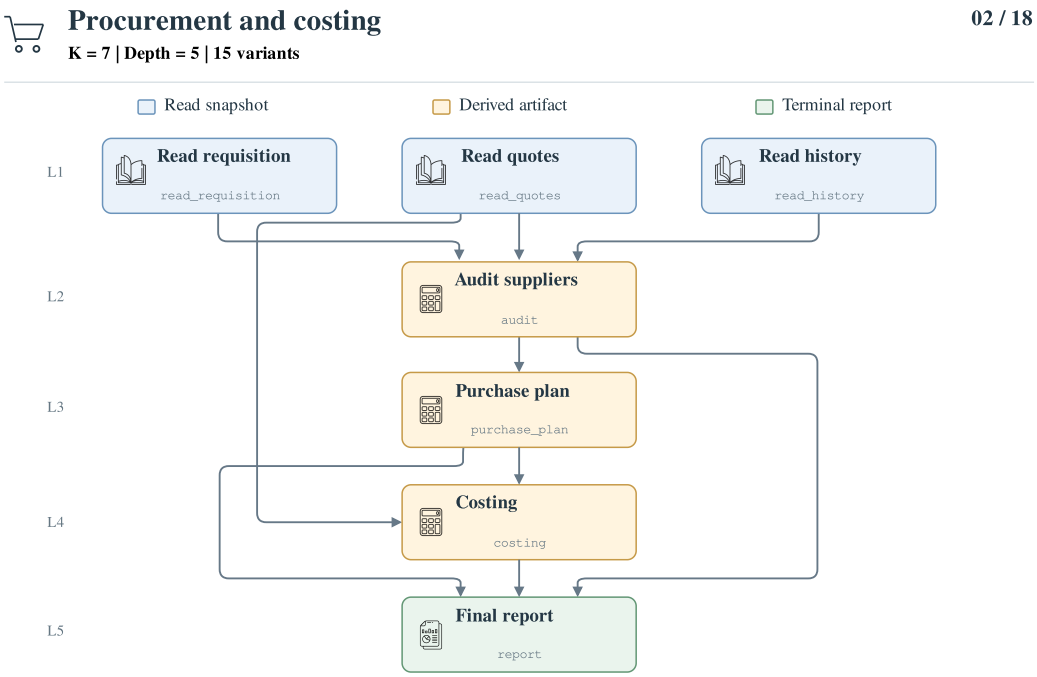

Figure 12: Workflow diagram for Procurement and costing (procurement).

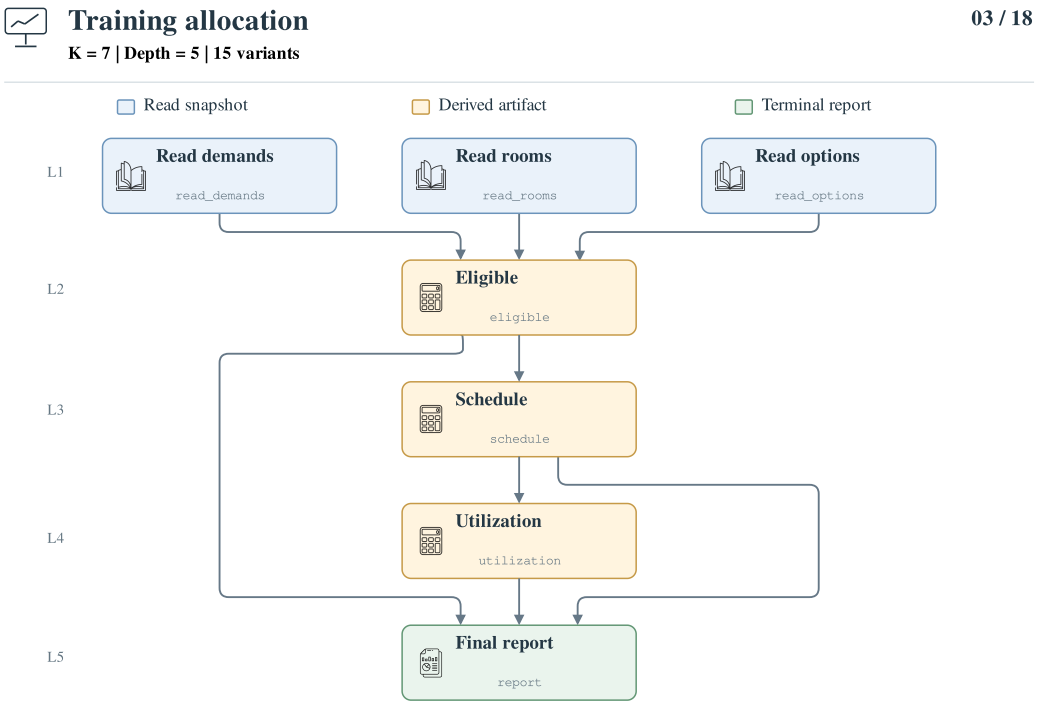

Figure 13: Workflow diagram for Training allocation (training).

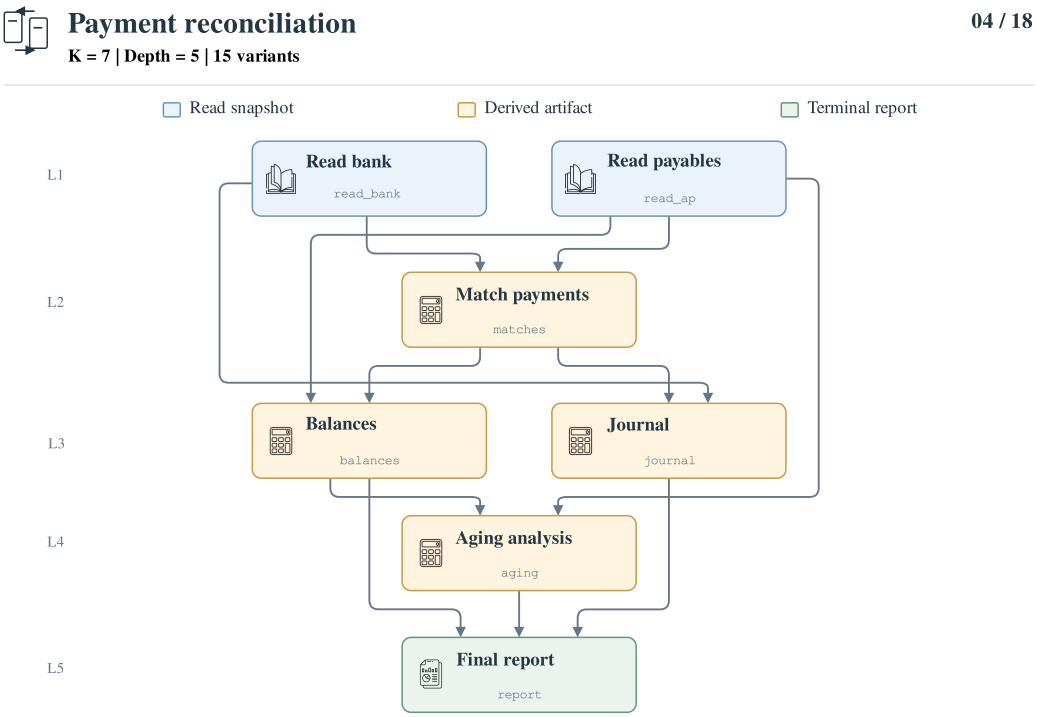

Figure 14: Workflow diagram for Payment reconciliation (reconciliation).

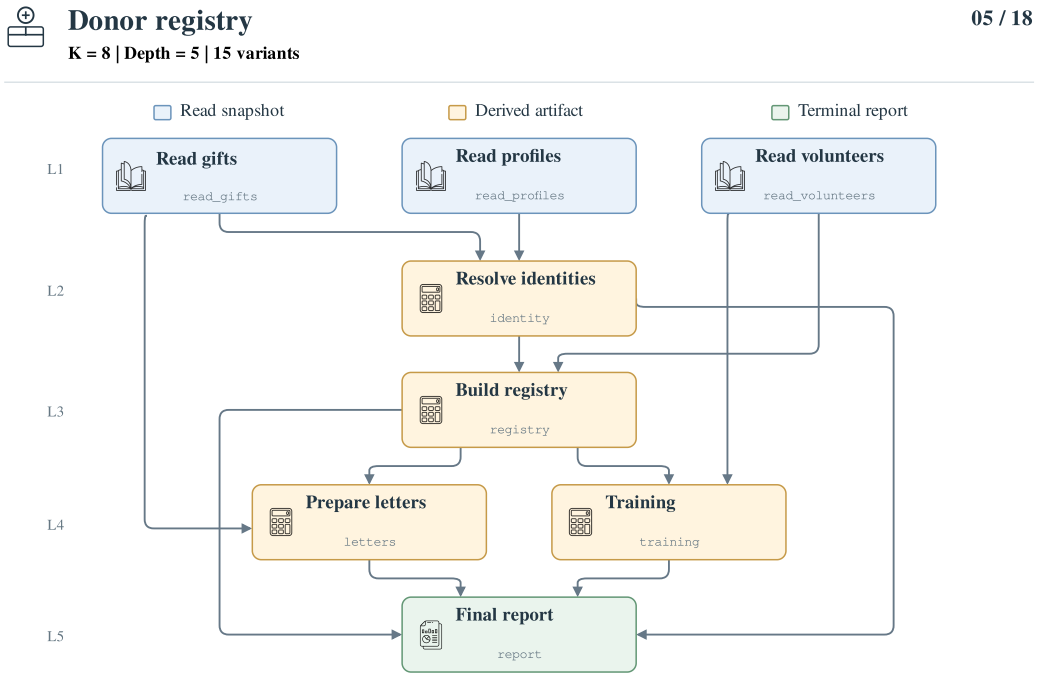

Figure 15: Workflow diagram for Donor registry (donor_registry).

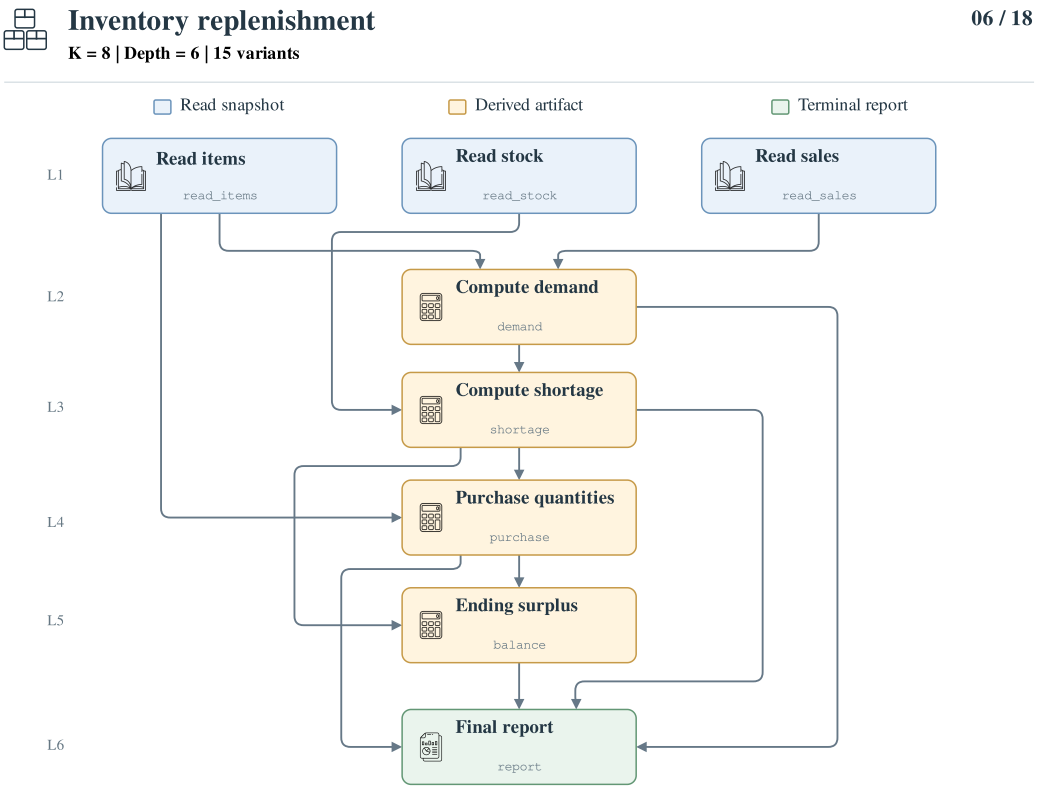

Figure 16: Workflow diagram for Inventory replenishment (inventory_replenishment).

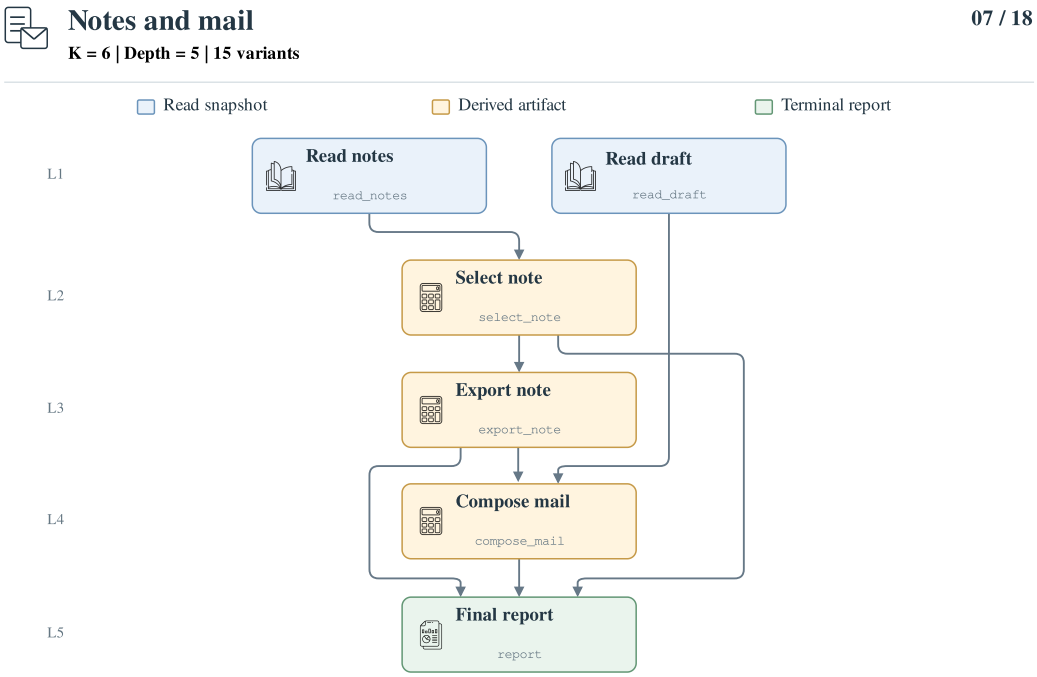

Figure 17: Workflow diagram for Notes and mail (aw_note_mail).

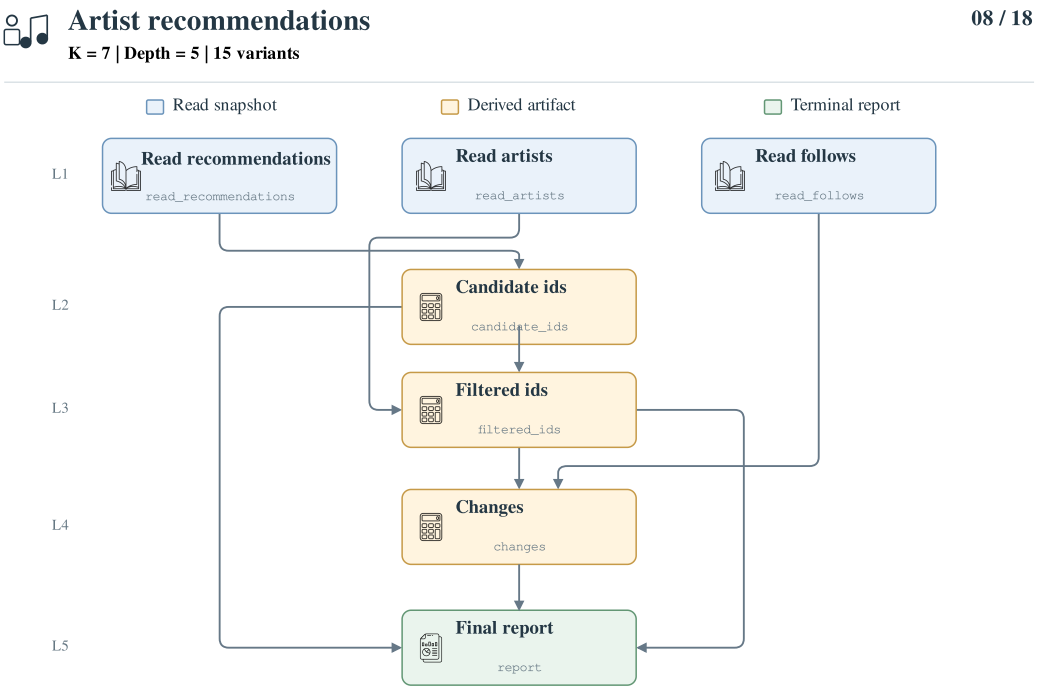

Figure 18: Workflow diagram for Artist recommendations (aw_artist_follow).

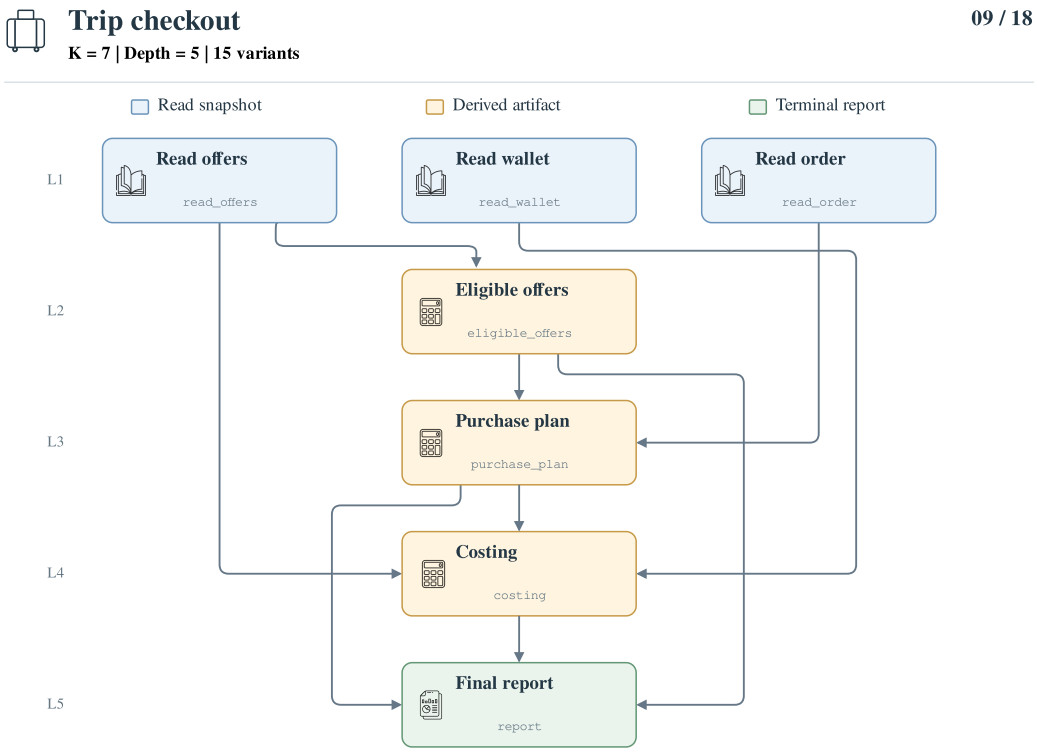

Figure 19: Workflow diagram for Trip checkout (aw_trip_checkout).

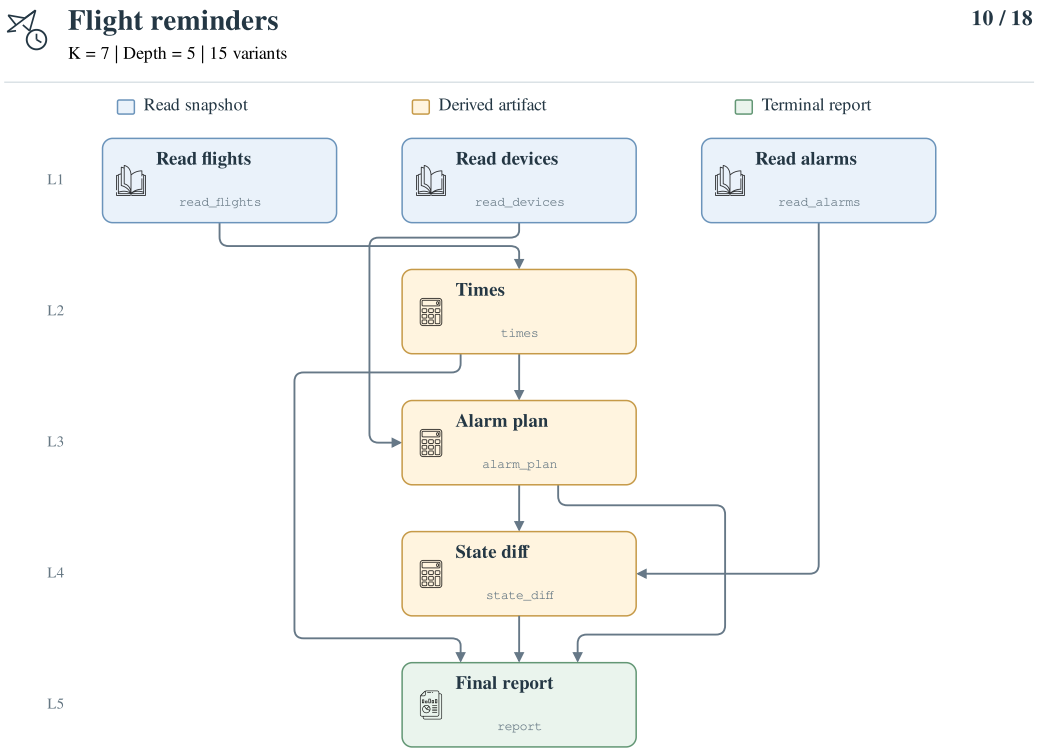

Figure 20: Workflow diagram for Flight reminders (aw_flight_alarms).

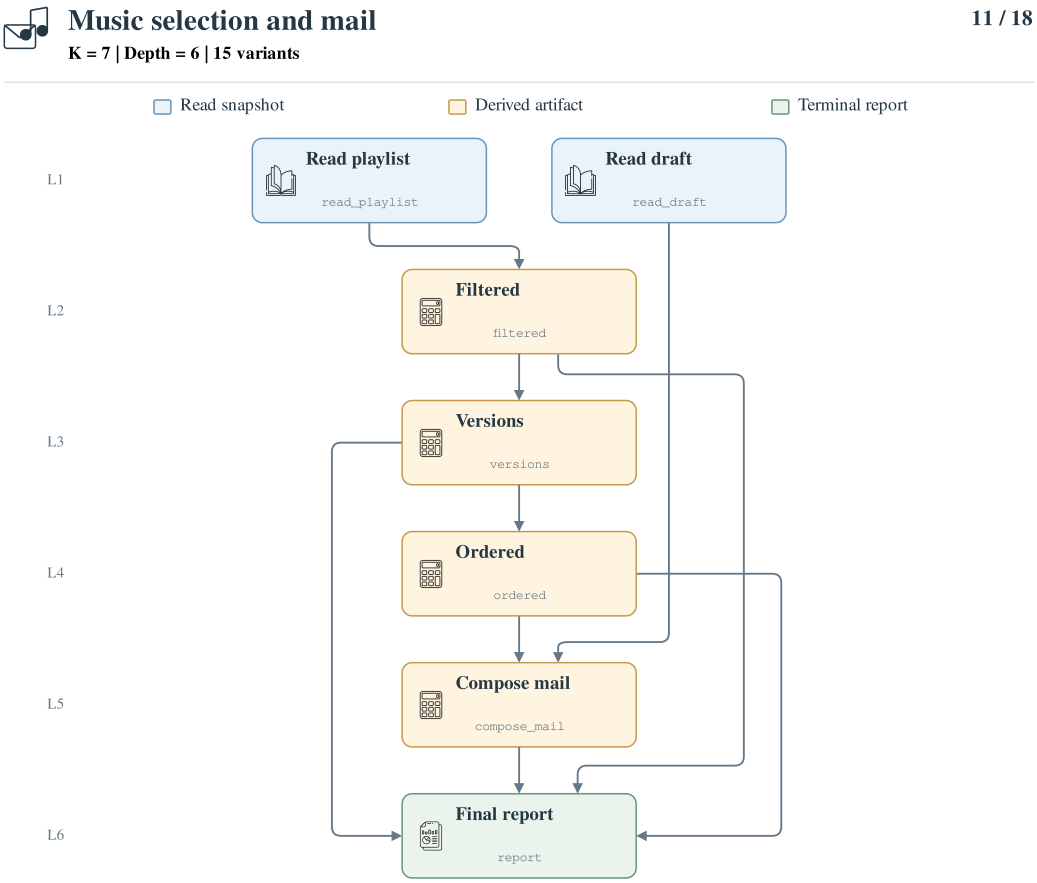

Figure 21: Workflow diagram for Music selection and mail (aw_music_mail).

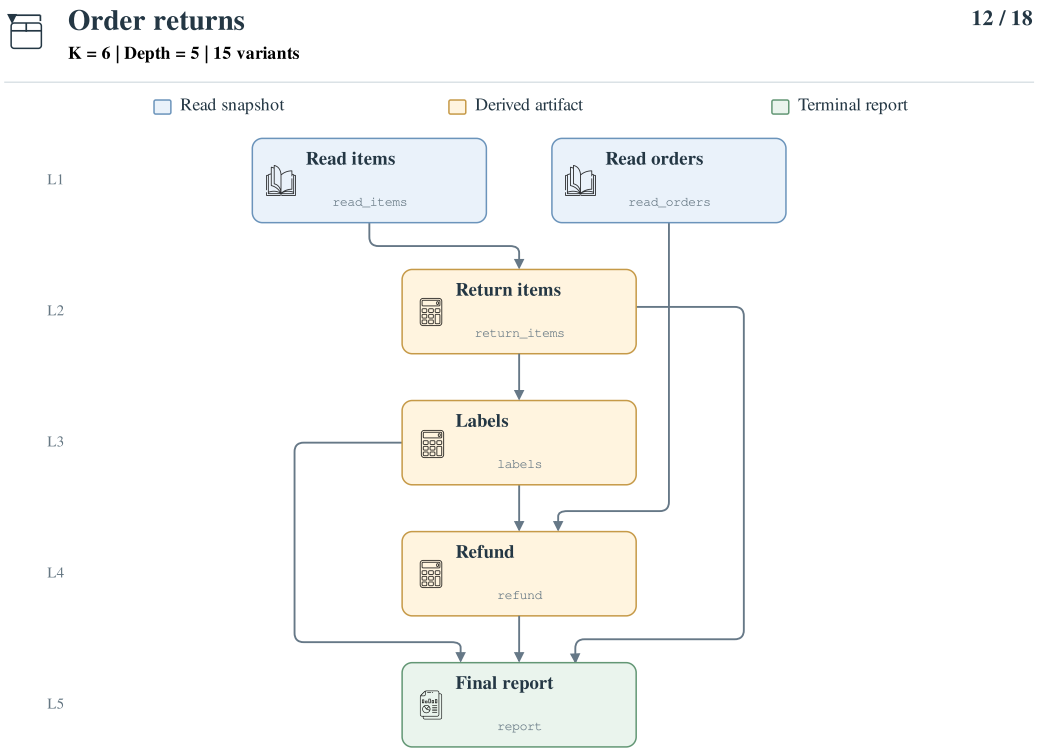

Figure 22: Workflow diagram for Order returns (aw_order_returns).

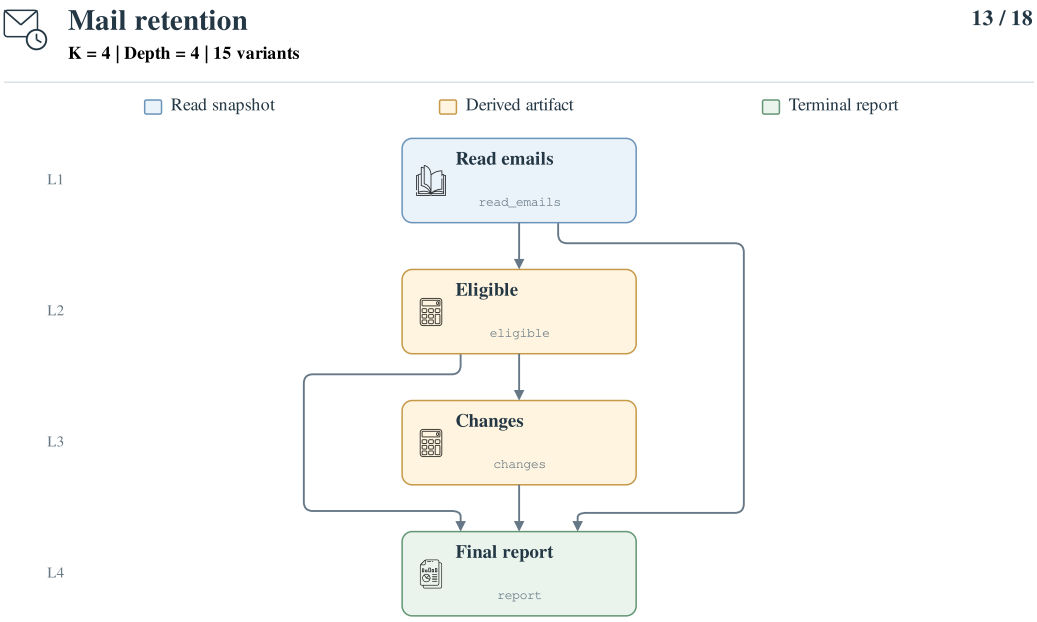

Figure 23: Workflow diagram for Mail retention (wb_mail_cleanup).

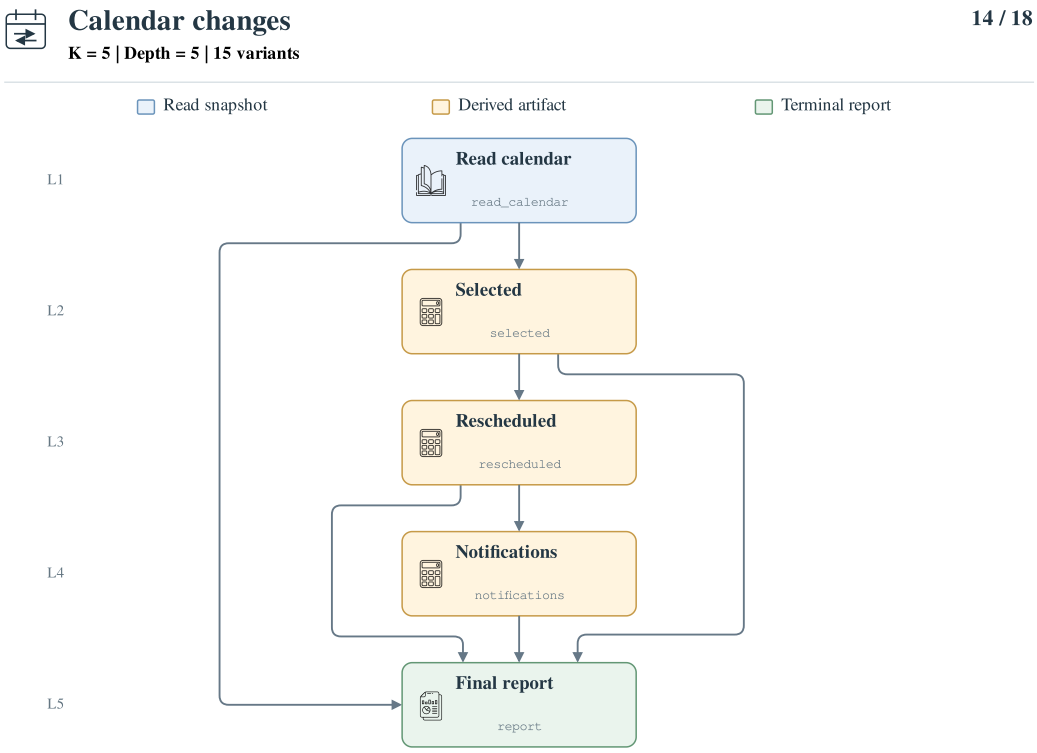

Figure 24: Workflow diagram for Calendar changes (wb_calendar_shift).

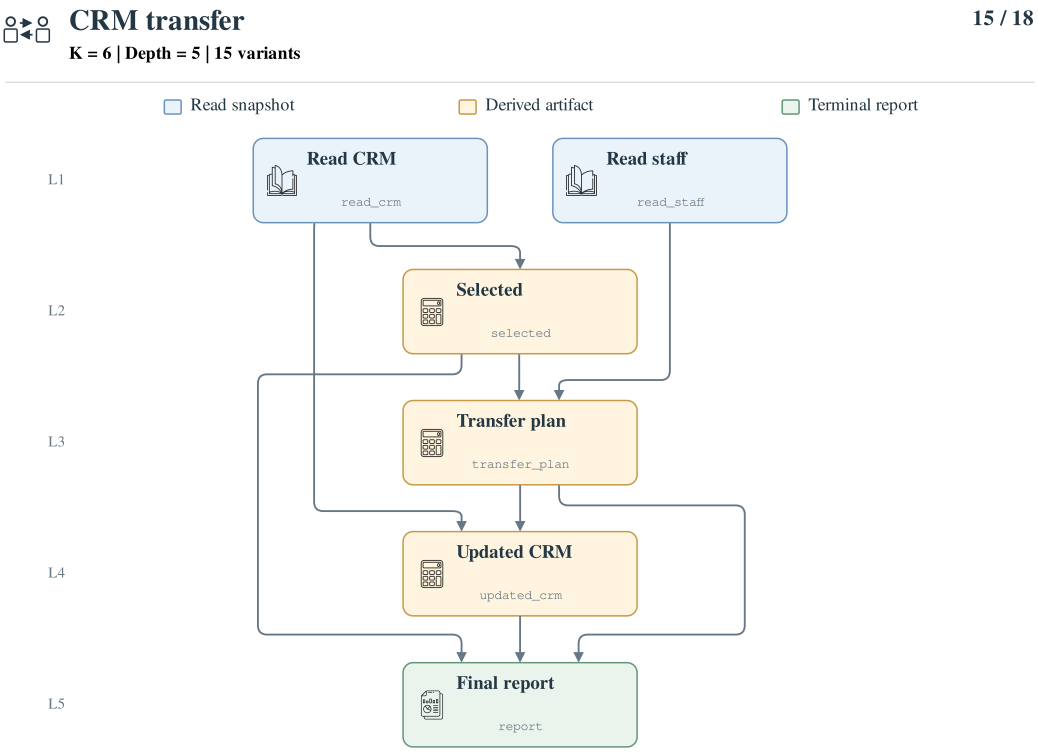

Figure 25: Workflow diagram for CRM transfer (wb_crm_transfer).

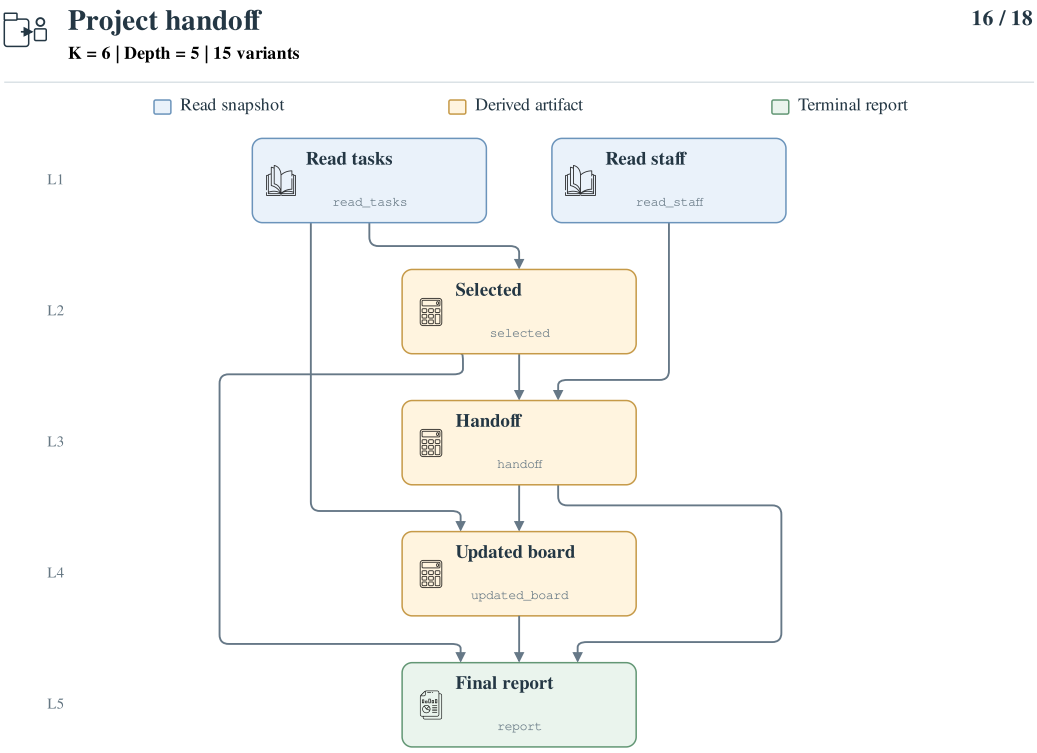

Figure 26: Workflow diagram for Project handoff (wb_project_handoff).

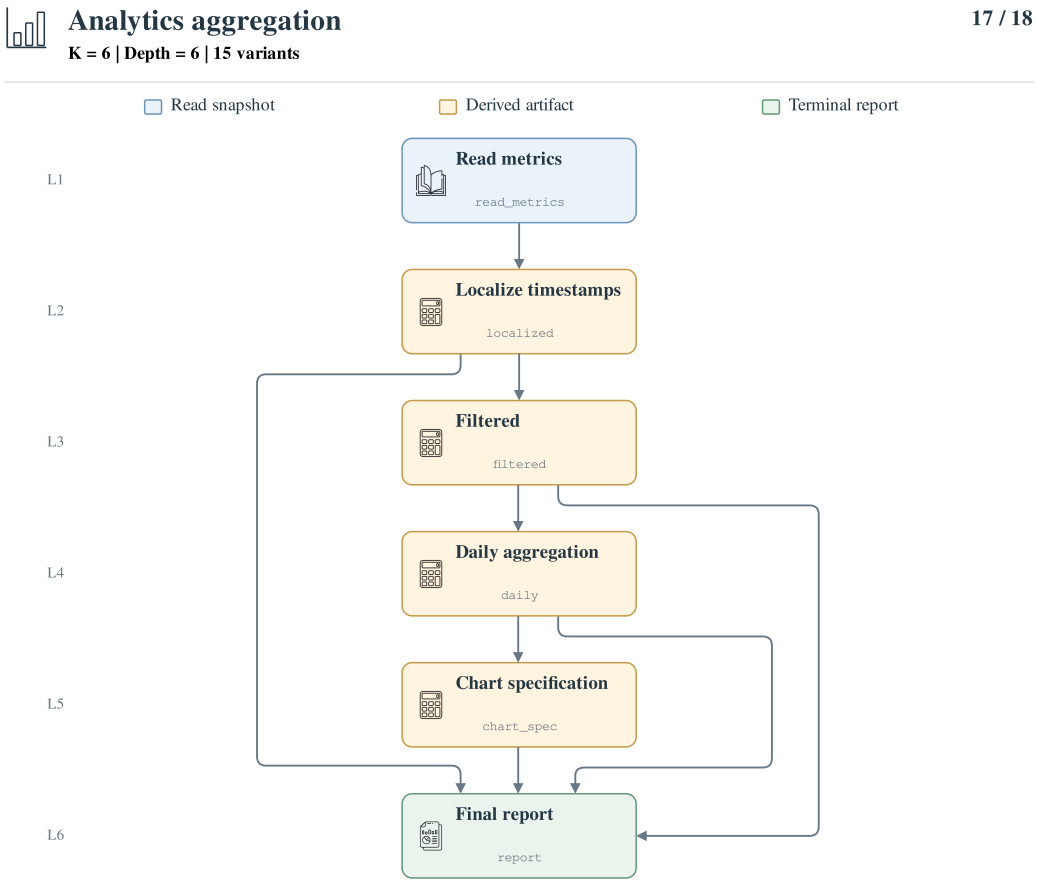

Figure 27: Workflow diagram for Analytics aggregation (wb_analytics_plot).

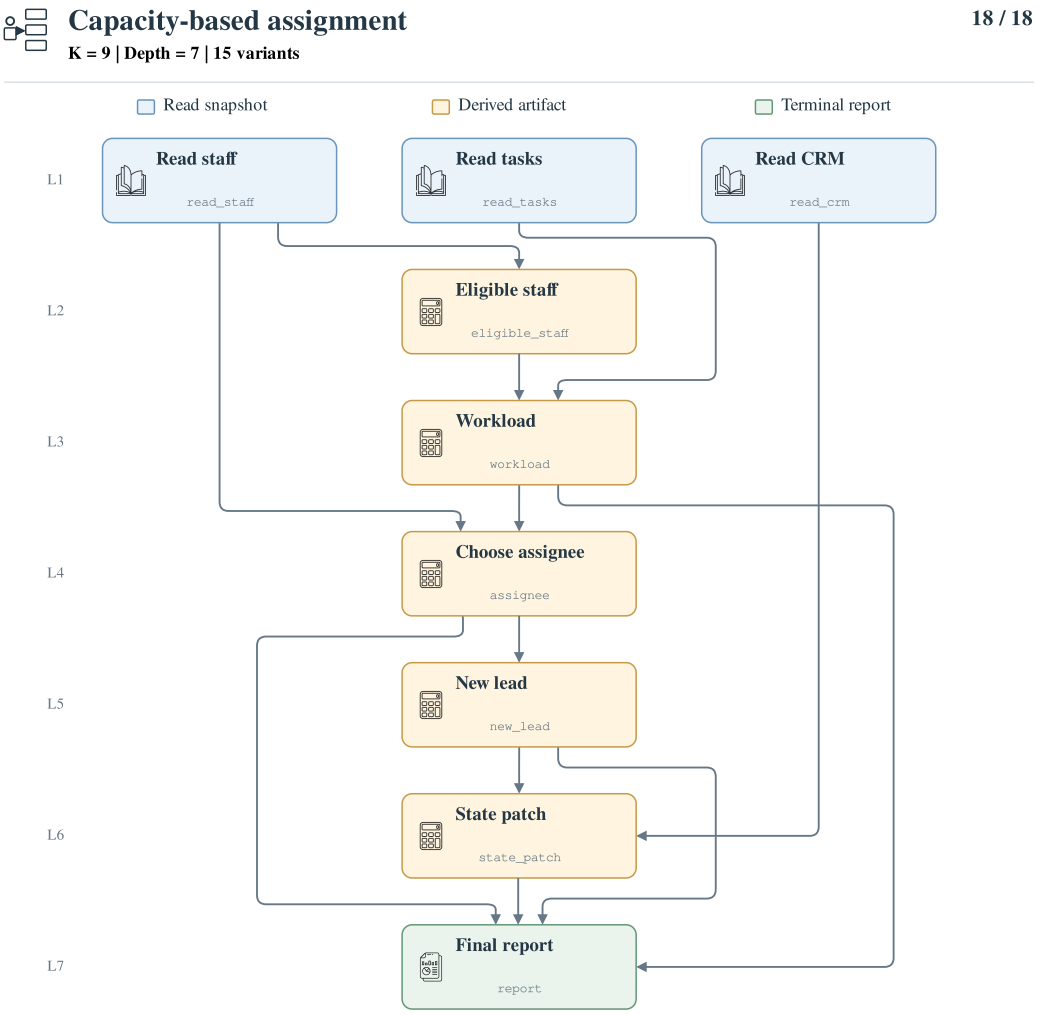

Figure 28: Workflow diagram for Capacity-based assignment (wb_capacity_lead).

K Output Tokens and Elapsed Time

These statistics use the same frozen original-attempt cohort as Tables 2–3, including unsuccessful attempts. Output tokens are summed over all evaluated-model replies within a task; they exclude the user simulator and annotation model. We use provider-reported completion tokens, including reasoning when the provider counts it there. Token means use only episodes with complete usage records. The historical GPT transports did not record usable evaluated-model token counts; these entries are marked as unavailable.

Task execution time includes model calls, pacing, simulation and tool execution, but excludes post-run information annotation. The second table additionally reports annotation time and execution-plus-annotation time. These measurements capture service-and-harness time, including pacing and simulation. P90 is the 90th percentile with linear interpolation.

Table A14: Evaluated-model output tokens. Usage n counts episodes with complete usage. Output includes provider-counted reasoning; – means unavailable.

Model	Completed	Usage n	Mean/task	Median/task	Mean/reply
Kimi-K3	270	266	1,177	1,086	154
HY4-Preview	270	270	7,445	6,771	1,015
DeepSeek-V4-Pro	270	270	11,364	4,662	1,105
Qwen3.8-Max	270	268	2,373	1,954	284
GLM-5.3	270	269	1,080	817	112
MiniMax-M2.5	270	267	7,909	4,923	419
GPT-6-Astra	270	0	–	–	–
Gemini-3.5-Flash	270	258	1,973	1,835	264
GPT-5.5	270	0	–	–	–

Table A15: Elapsed seconds per original attempt, including unsuccessful attempts. Combined time is execution plus post-run annotation.

Model	n	Exec. mean	Exec. median	Exec. P90	Annotation	Combined
Kimi-K3	270	64.6	60.7	89.5	3.2	67.8
HY4-Preview	270	227.1	192.2	390.6	6.1	233.2
DeepSeek-V4-Pro	270	121.7	62.4	278.2	3.2	124.8
Qwen3.8-Max	270	69.5	59.7	118.5	2.9	72.4
GLM-5.3	270	79.6	58.3	135.3	2.8	82.4
MiniMax-M2.5	270	126.0	73.7	268.6	1.3	127.3
GPT-6-Astra	270	50.0	40.4	56.9	2.7	52.7
Gemini-3.5-Flash	270	55.8	54.3	77.7	9.2	64.9
GPT-5.5	270	40.3	39.9	51.5	1.8	42.1