FDE-BENCH: EVALUATING END-TO-END DELIVERY FROM UNDERSPECIFIED REAL-WORLD BUSINESS REQUESTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Real-world business requests often leave key requirements unstated because frontline practitioners struggle to articulate rules they know from experience. Forward-deployed engineers (FDEs) must elicit these requirements through dialogue and iteratively optimize solutions until they meet customer acceptance criteria. Existing optimization benchmarks typically start from explicit objectives, while clarification benchmarks often introduce information gaps into existing tasks. We introduce FDE-Bench, the first benchmark built from real customer projects to evaluate agents on end-to-end delivery tasks that require interactive requirement clarification and iterative optimization. Its 49 cases span combinatorial optimization and machine learning. Clarification targets are distilled from discussions between frontline practitioners and FDEs. Agents elicit missing requirements from a simulated customer, and case-specific evaluators assess final artifacts against acceptance criteria using customer-accepted FDE solutions as quality references. Across 18 agent configurations, the highest mean delivery score is only **0.5510**, compared with an FDE reference of **1.0**. The highest delivery success rate (DSR), defined as the fraction of planned runs reaching this reference, is only **6.12%**. These results show that reliable FDE-level end-to-end delivery remains out of reach for the evaluated agents on underspecified real-world business tasks. The code and dataset for FDE-Bench are anonymously available at: https://anonymous.4open.science/r/FDE-Bench/.

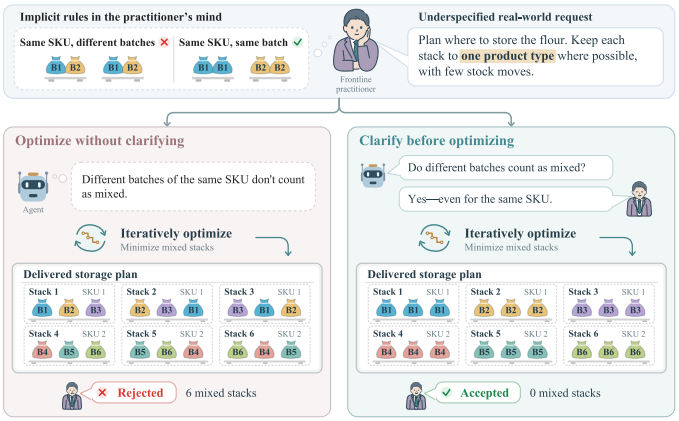
1 INTRODUCTION

In real customer projects, frontline practitioners often rely on business rules learned through experience but struggle to make them explicit in their requests (Nuseibeh & Easterbrook, 2000). Forward-deployed engineers (FDEs) must elicit these tacit requirements through dialogue and iteratively optimize solutions until they meet the customer's acceptance criteria (Zowghi & Coulin, 2005). Figure 1(a) illustrates a warehouse request to keep each stack to "**one product type**" where possible while avoiding unnecessary stock moves. The request leaves implicit that different batches of the same product (SKU) also count as mixed. Grouping only by SKU can therefore leave mixed stacks under the customer's criterion. After clarifying this rule, the agent must still iteratively refine storage assignments and moves to meet the delivery requirements.

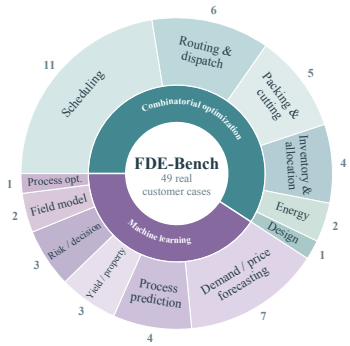
Representative benchmarks do not yet fully capture this challenge. Optimization benchmarks such as HeuriGym (Chen et al., 2026) and Frontier-Eng (Chi et al., 2026) provide explicit objectives and constraints, leaving requirement acquisition outside their scope. ClarEval (Li et al., 2026b) perturbs programming instructions to create omissions and ambiguities, while HIL-Bench (Trinh et al., 2026) introduces human-validated blockers requiring interaction in coding and SQL tasks. Their information gaps are constructed within predefined tasks; we instead ask whether agents can elicit requirements grounded in frontline practitioners' experience and iteratively optimize solutions to meet real customers' delivery standards.

We introduce FDE-Bench, a benchmark of 49 cases from real customer projects for evaluating agents from requirement clarification through iterative optimization to final delivery. As summarized in Figure 1(b), the cases cover 29 combinatorial-optimization and 20 machine-learning tasks across diverse business settings. Each case links the original request and business data, requirements distilled

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.



(a) **A warehouse case**: clarify the mixing rule, then optimize.



(b) **Overview of FDE-Bench:** 49 real customer cases across task families and subtypes.

Figure 1: **FDE-Bench: motivation and coverage.** (a) Clarifying mixing rules guides iterative warehouse optimization. (b) 49 real customer cases span two task families.

from frontline practitioner–FDE discussions, and a customer-accepted FDE solution. Agents clarify requirements with a simulated customer, then iteratively execute and improve their solutions. After submission, case-specific evaluators assess final artifacts against the project's acceptance criteria, using the accepted FDE solution as the quality reference. Separate clarification scores measure how well agents' questions cover the annotated requirements. Since performance depends on both the model and its scaffold (Kapoor et al., 2026), we build an extensible evaluation harness to compare different configurations under a shared protocol.

We evaluate 18 agent configurations spanning seven model families and six agent scaffolds. Across configurations, the highest mean delivery score is only 0.5510 against a customer-accepted FDE reference of 1.0. The highest delivery success rate (DSR)—the fraction of planned runs reaching that reference—is only 6.12%. Broader requirement coverage in agents' questions is associated with higher delivery quality. Paired information comparisons show higher scores with required clarification than with no customer answers, but even complete business information leaves all compared configurations below the FDE reference on average. These results expose gaps in both requirement acquisition and solution optimization.

Our contributions are:

- **A benchmark for end-to-end FDE delivery.** FDE-Bench is the first benchmark to couple interactive requirement clarification with iterative optimization on real customer projects, using customer-accepted FDE solutions as delivery references.
- **An extensible evaluation harness.** Backend adapters support new models and agent scaffolds; a shared protocol records interactions and evaluates submitted artifacts, with separate measures of requirement coverage and delivery quality.
- **A systematic study of current agents.** Across 18 configurations, we analyze representative delivery failures and the association between requirement coverage and delivery quality. Information ablations show that complete business information alone does not close the delivery gap.

2 Related Work

Optimization modeling and search. NL4Opt and OptiMUS study natural-language optimization modeling, while OptiBench and OptMATH provide data for evaluating or training modeling systems (Ramamonjison et al., 2023; Ahmaditeshnizi et al., 2024; Yang et al., 2025; Lu et al., 2025). MIPLIB-NL extends evaluation to industrial-scale mixed-integer problems (Li et al., 2026c). CO-Bench, HeuriGym, and ALE-Bench evaluate iterative algorithm search and optimization (Sun et al., 2026; Chen et al., 2026; Imajuku et al., 2026), while ORAgentBench and Opti-Agent-Bench extend evaluation to workflows involving problem formulation, file inspection, and execution (Li et al.,

2026a; Fu et al., 2026). These works primarily assess how agents formulate and solve given optimization tasks. FDE-Bench additionally evaluates whether agents can elicit unstated business requirements through customer interaction and use them to guide optimization.

Clarification and interactive agents. MINT evaluates multi-turn tool use with language feedback, while τ-bench studies customer interaction and policy compliance in API-based service tasks (Wang et al., 2024; Yao et al., 2025). CLAMBER focuses on identifying ambiguous information needs and generating clarifying questions (Zhang et al., 2024). HIL-Bench introduces blockers into coding and SQL tasks, while ClarEval injects ambiguities into programming instructions (Trinh et al., 2026; Li et al., 2026b). Ambig-SWE evaluates clarification in software tasks (Vijayvargiya et al., 2026), while Pan et al. (2025) study code generation with interactive feedback. SWE-RPG diagnoses implicit-requirement recovery from coding trajectories but does not evaluate interactive user feedback (Zhou et al., 2026). FDE-Bench grounds clarification targets in discussions between frontline practitioners and FDEs from real customer projects, then evaluates delivery quality after agents clarify requirements and iteratively optimize their solutions.

Executable evaluation and expert references. SWE-bench and Terminal-Bench use executable verification to assess software and command-line tasks (Jimenez et al., 2024; Merrill et al., 2026). Frontier-Eng and EngDesign evaluate engineering solutions through task-specific scoring or simulation (Chi et al., 2026; Guo et al., 2026). MLE-bench measures machine-learning engineering performance, while RE-Bench compares agents with human experts on ML research engineering tasks (Chan et al., 2025; Wijk et al., 2025). FDE-Bench also uses executable evaluation and reference solutions. Its distinguishing feature is that the initial request, clarification targets, and customer-accepted FDE reference originate from the same customer project. Agents must elicit initially unstated requirements and optimize their deliverables against that project's acceptance criteria. Appendix A.1 provides a comparison with representative benchmarks.

3 FDE-BENCH: TASKS AND CONSTRUCTION

3.1 FDE DELIVERY WORKFLOW

Each case draws on the FDE delivery workflow of a real customer project (Figure 2). Frontline practitioners provide an underspecified business request and business data. Through discussion, the FDE elicits business rules and acceptance criteria, then iteratively optimizes a solution until customer acceptance. The initial request and data, project discussions, and accepted FDE solution from the same project supply task inputs, requirement annotations, and delivery references, respectively. The original discussion transcripts remain hidden from evaluated agents.

3.2 BENCHMARK CONSTRUCTION PIPELINE

Prepare task inputs. The initial request and business data are de-identified with agent assistance and reviewed by an FDE. The process preserves the request's original meaning and level of detail, applies replacements consistently across related files, and retains numerical relationships and constraints needed for execution and evaluation. The visible inputs are `instruction.md` and `data/`; the de-identification agent does not decide requirements or acceptance criteria.

Distill requirements. The FDE responsible for the original delivery annotates requirements established in the customer–FDE discussions. We store complete business context in `information.md` and clarification targets in `gt.json`. A target is retained only if it cannot be resolved from the initial request and visible data, and its resolution can affect feasibility, the objective or target value, or customer acceptance. Examples include the meaning of a data field, the scope of a constraint, and the convention used to measure an objective. Each annotation records the requirement and the information to clarify, without prescribing question wording or dialogue order. The 49 cases contain 266 targets, or 5.43 per case on average.

Establish delivery evaluation. Complete requirements and project acceptance criteria are translated into a case-specific evaluator that checks the artifact schema, hard constraints, and case-defined quality. The customer-accepted FDE deliverable supplies the reference, normalized to 1.0. This is an achieved

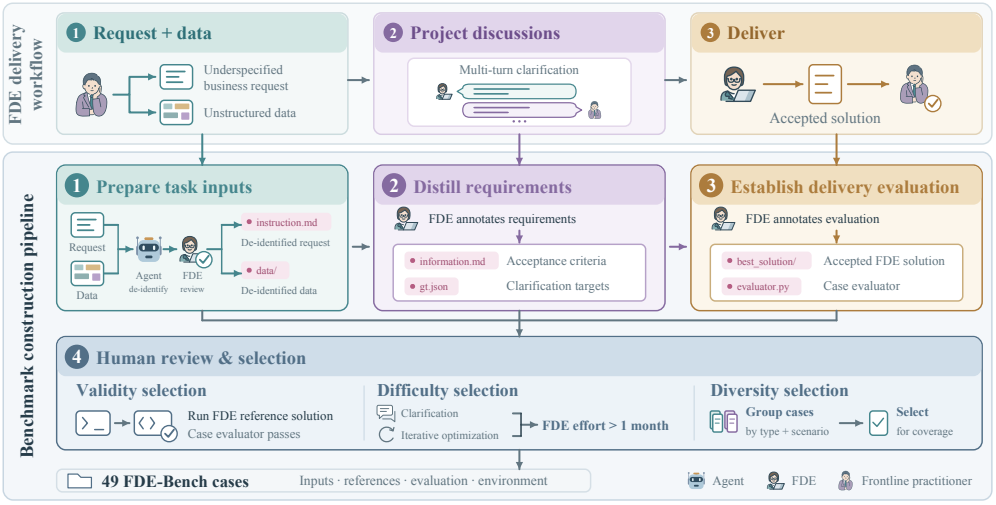


Figure 2: **FDE-Bench construction pipeline.** Real-project materials become agent-visible inputs, annotated requirements, case-specific evaluation, and a reviewed case package.

project standard rather than a claim of global optimality; different algorithms are therefore judged against a common set of requirements.

Human review and selection. Reviewers run each FDE reference solution with its evaluator and check that the evaluation matches the documented business requirements and acceptance criteria. They also review the FDE annotations using common criteria across cases, checking targets against the evaluator and reference artifact, removing duplicates, and confirming that each target has a documented customer answer. Cases with incomplete materials or inconsistent checks are excluded. To select projects involving sustained delivery work, we retain those that required more than one month of FDE effort on clarification and iterative optimization before customer acceptance. Reviewers select cases across computational task types and business settings to ensure coverage.

3.3 Task Coverage and Specification

The release contains 29 combinatorial-optimization and 20 machine-learning cases, spanning manufacturing, logistics, energy, retail, finance, and engineering settings. Primary computational subtypes are listed in the case inventory; scenario tags can overlap and are not mutually exclusive counts.

We represent each case as $\mathcal{C} = (u, D, I, G, E)$, comprising the initial request u, business data D, complete business-information file I, clarification targets G, and case-specific evaluator E. Each released case package also includes an FDE reference deliverable and its reference quality, an execution environment, and metadata. Under the main condition, `Interact-Req`, only u and D are initially exposed in the solving workspace; I, G, E, and the reference deliverable remain outside it. Section 4 specifies how agents interact with the simulated customer and how their submitted artifacts and clarification questions are evaluated.

4 Evaluation Protocol

FDE-Bench evaluates two outputs of each run: the submitted solution file (the *artifact*) and the interaction log. Case-specific evaluators score delivery, while a separate judge panel assesses clarification coverage (Figure 3). Section 5.4 compares the main condition, `Interact-Req`, with no-answer and full-information conditions.

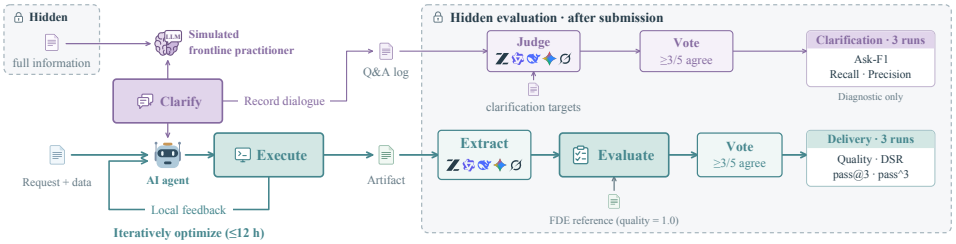


Figure 3: **FDE-Bench evaluation framework.** Agents clarify requirements and iteratively optimize deliverables. After submission, hidden evaluation extracts and scores artifacts against FDE references without online feedback; clarification coverage is assessed separately.

4.1 Evaluation Procedure

Customer interaction. Agents query a simulated customer for business information. The customer uses the request u, complete business information I, and dialogue history to answer, without access to clarification targets G. We use GLM-5.3 with a fixed prompt and record exchanges. Benchmark prompts and the native-tool, callback, and text-exchange interfaces are provided with the benchmark.

Solving and submission. The `Interact-Req` prompt instructs agents to elicit missing information before implementation, continuing until they consider the information sufficient. Within the execution budget and interaction limits, agents choose their questions, implement a candidate solution, execute it locally, inspect the resulting files and diagnostics, and repeatedly revise and re-execute the solution. They submit the final artifact only after this local optimization loop. The main campaign allows 43,200 seconds (12 hours) per solving run, including interaction, and at most 30 clarification rounds with no separate question-count cap. The hidden evaluator and artifact extractor run only after submission and provide no score or evaluator feedback during this loop.

Extraction and case evaluation. After submission, extractors convert artifacts into the case evaluator's input format while preserving the agent's solution and decisions. The evaluator checks the case's hard constraints. Violations receive a delivery score of zero; otherwise, it computes case-defined quality and normalizes it to the customer-accepted FDE reference, which scores 1.0. Up to five models—GLM-5.2, Qwen3.8-27B, DeepSeek-V4-Flash, Gemini-3.6-Flash, and Grok-4.5—extract the artifact for scoring by the same case evaluator. Remaining routes are skipped once three scores agree after rounding to four decimal places. A score is recorded when at least three routes agree. The validity outcome is taken from an evaluation supporting that score; this model-panel design is related to prior work on LLM-assisted evaluation (Zheng et al., 2023; Liu et al., 2023).

Requirement coverage. We split recorded questions into units and use the same five models to assess coverage of G. For each requirement, judges independently assess whether answering at least one question would reveal it. Judges receive the requirement and numbered questions, but not customer answers or requirement-category metadata. Coverage is accepted when at least three of five judgments agree ($\geq 3/5$); otherwise the requirement is uncovered.

4.2 Evaluation Metrics

FDE tasks require agents to elicit implicit business requirements and turn them into a deliverable that meets the customer's standard. We measure clarification and delivery separately to examine whether effective requirement acquisition translates into successful delivery.

Delivery metrics. In the main evaluation, each configuration runs three times on each of 49 cases. Quality first averages the three delivery scores within each case, then averages across cases with equal weight. Delivery Success Rate (DSR) is the fraction of the 147 runs reaching the FDE reference. Let s_{ir} denote the delivery score for repeat r of case i. We define

$$\text{Quality} = \frac{1}{49}\sum_{i=1}^{49}\left(\frac{1}{3}\sum_{r=1}^{3} s_{ir}\right), \qquad \text{DSR} = \frac{1}{147}\sum_{i=1}^{49}\sum_{r=1}^{3}\mathbf{1}[s_{ir} \geq 1].$$

Following the pass@k framing in code-generation evaluation (Chen et al., 2021), we also report pass@3 and pass^3: the fractions of cases with at least one and all three valid runs, respectively. These measure validity across repeated runs; DSR measures attainment of the FDE reference.

Clarification metrics. Missing business requirements can lead an agent to solve the wrong problem. Recall (R) measures how fully its questions cover the reference requirements, while Precision (P) measures how focused the questions are on those requirements. Their harmonic mean, Ask-F1, rewards broad coverage with focused questioning, following the standard precision–recall terminology in information retrieval (Schütze et al., 2008). For each run, let $G_{\text{cov}} \subseteq G$ denote covered requirements, Q the set of question-unit indices, and $Q_{\text{rel}} \subseteq Q$ the indices matched to at least one covered requirement. These three primary metrics are

$$R = \frac{|G_{\text{cov}}|}{|G|}, \qquad P = \frac{|Q_{\text{rel}}|}{|Q|}, \qquad \text{Ask-F1} = \frac{2PR}{P+R}.$$

Recall is question coverage: it measures whether a question could elicit a reference requirement, not whether the customer stated it or the agent used it; questions matching multiple requirements count once.

Efficient clarification should elicit business requirements with few customer exchanges and limited question burden. First-turn coverage (KQC_1) measures how much agents cover in the initial exchange, while average turns to clarify (ATC) measures how early they address the requirements eventually covered. Unmatched-question count (Redund.) captures the absolute number of questions unmatched to covered requirements. Interpreted alongside Recall, these diagnostics characterize the trade-off between requirement coverage and interaction effort. Let $G_{\text{cov},1} \subseteq G$ denote requirements covered in the first interaction turn, and $G^{\star}_{\text{cov}} \subseteq G_{\text{cov}}$ those with a recorded matching question and turn. For each requirement $g \in G^{\star}_{\text{cov}}$, let t_g be its first covering turn, counting from 1. We define

$$\text{KQC}_1 = \frac{|G_{\text{cov},1}|}{|G|}, \qquad \text{ATC} = \frac{1}{|G^{\star}_{\text{cov}}|} \sum_{g \in G^{\star}_{\text{cov}}} t_g, \qquad \text{Redund.} = |Q| - |Q_{\text{rel}}|.$$

Clarification metrics are computed per run and averaged over runs for which each metric is defined.

4.3 Evaluation Reliability

Two human experts independently audited stratified samples of retained traces, covering 1,303 simulated-customer answers, 120 artifact extractions, and 120 clarification items across all 18 configurations. Sampling procedures, confidence intervals, and full audit results are provided in Appendix A.8. For answer grounding, experts A and B marked 95.93% and 90.48% of answers as supported, respectively; their support labels agreed on 92.10% of records (Cohen's $\kappa = 0.385$ (Cohen, 1960)). For extraction fidelity, on a sample enriched for runs with extraction issues, they judged 93.33% and 91.00% of assessable extractions faithful to the submitted artifacts, respectively, with 91.67% three-category agreement (Cohen's $\kappa = 0.7656$). For clarification coverage, the model panel agreed with expert A on 94.74% ($\kappa = 0.8947$) and with B on 93.33% ($\kappa = 0.8667$), excluding uncertain labels. These results support the reliability of the evaluation pipeline on the audited subsets; the extraction κ specifically measures expert-label agreement, rather than extractor accuracy against an adjudicated gold standard. These checks reduce but do not eliminate extraction-related score risk.

5 Experiments

We evaluate 18 agent systems and ask three questions. **RQ1** (§5.2): how close do agents come to the accepted FDE deliverable? **RQ2** (§5.3): which clarification behaviors distinguish stronger systems? **RQ3** (§5.4): how does delivery quality change with access to business information?

5.1 Experimental Setup

We evaluate scaffold–model systems: general scaffolds `opencode` and `openhands`, each with seven models (GPT-6-Astra, GLM-5.3, Qwen3.8-Max, Grok-4.6, DeepSeek-V4-Pro, Kimi-K3, Gemini-3.1-Pro), and four native pairs (`codex`, `deepseek-harness`, `kimi`, `gemini`). All

main runs use `Interact-Req` with a fixed simulated customer, a 12-hour budget, and at most 30 clarification rounds; each system solves each of 49 cases three times. Paired contrasts use two-sided Wilcoxon signed-rank tests (Wilcoxon, 1945) on case means with Holm correction (Holm, 1979). Settings and bootstrap intervals are reported with the released results.

5.2 RQ1: End-to-End Delivery

Table 1: **End-to-end evaluation.** 49 cases, three runs per case, `Interact-Req`. Quality is case-macro; other metrics are percentages. Bold marks blockwise maxima before rounding.

	Delivery ↑				Clarification ↑		
Agent / model	**Quality**	**DSR**	**pass@3**	**pass^3**	**Recall**	**Precision**	**Ask-F1**
General scaffold: `opencode`							
GPT-6-Astra	**0.5459**	**5.44**	91.84	**81.63**	**83.64**	36.53	**48.82**
GLM-5.3	0.5413	1.36	91.84	77.55	66.83	**39.70**	47.46
Qwen3.8-Max	0.5387	2.72	**95.92**	73.47	70.56	35.41	45.11
Grok-4.6	0.5221	2.72	87.76	79.59	68.19	38.30	46.42
DeepSeek-V4-Pro	0.4986	1.36	91.84	71.43	55.08	37.93	42.73
Kimi-K3	0.4187	0.00	91.84	53.06	55.03	37.60	43.27
Gemini-3.1-Pro	0.3873	0.00	89.80	59.18	47.10	39.48	40.38
General scaffold: `openhands`							
GPT-6-Astra	**0.5302**	4.76	**93.88**	67.35	**84.67**	36.80	**49.50**
GLM-5.3	0.5274	2.04	91.84	67.35	67.12	36.79	45.48
Grok-4.6	0.5159	0.68	91.84	**71.43**	69.91	36.95	45.91
DeepSeek-V4-Pro	0.5104	0.68	**93.88**	**71.43**	65.24	36.41	44.92
Qwen3.8-Max	0.5071	**6.12**	**93.88**	65.31	75.37	30.89	42.24
Gemini-3.1-Pro	0.4571	0.00	87.76	65.31	45.45	**37.77**	39.25
Kimi-K3	0.4119	0.68	85.71	57.14	57.94	35.39	41.34
Native scaffolds							
`codex` + GPT-6-Astra	**0.5510**	**4.08**	**93.88**	**75.51**	**84.70**	37.87	**50.12**
`deepseek-harness` + DeepSeek-V4-Pro	0.5271	1.36	91.84	**75.51**	61.79	**39.32**	45.93
`kimi` + Kimi-K3	0.4203	0.68	89.80	59.18	55.69	36.69	42.90
`gemini` + Gemini-3.1-Pro	0.4037	0.00	87.76	57.14	44.94	38.82	39.37

FDE-level delivery remains rare. Across the 18 systems, the highest mean Quality is only 0.5510 (`codex` + GPT-6-Astra), and the highest DSR is 6.12% (Table 1). Only 8 of 49 cases have any reference-level run, and in 23 the best system averages below 0.8. Four systems, including all three Gemini-3.1-Pro systems, reach the reference in none of their 147 runs, and no ML run reaches it in any system.

Validity varies across repeated runs. Across systems, 85.7–95.9% of cases have at least one valid repeat among three attempts (pass@3). However, pass^3 trails pass@3 for every system by 8.2–38.8 percentage points (mean 23.0): for a typical system, 23% of cases are valid in some runs but not in all three. Of the 548 zero-score runs, 78.5% record a constraint or feasibility violation and 1.5% a schema error (Appendix A.7): agents submit well-formed artifacts that break business rules.

Runtime and case difficulty. Across systems, mean run time ranges from 0.3 to 5.5 hours, with no significant association with Quality (Spearman $\rho = 0.18$, $p = 0.47$) (Spearman, 1961). Case-level Quality rankings are similar across systems (mean pairwise Spearman $\rho = 0.71$).

The model matters more than the scaffold. With the scaffold fixed, Quality spans 0.159 across models on `opencode` and 0.118 on `openhands`; with the model fixed, the two general scaffolds differ by at most 0.032 for six models (Gemini-3.1-Pro: 0.387 vs. 0.457). Native scaffolds add 0.005 and 0.017 for GPT-6-Astra and DeepSeek-V4-Pro, while `gemini` trails `openhands` by 0.053.

5.3 RQ2: Clarification and Delivery

Figure 4 correlates clarification and delivery metrics across the 18 systems.

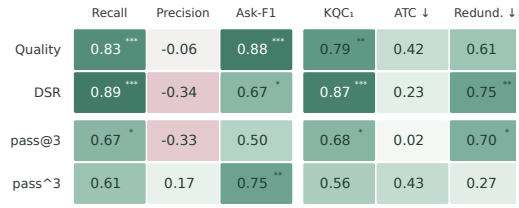


Figure 4: **Clarification–delivery correlations.** Cells report rank correlations across 18 configurations. Stars mark Holm-adjusted two-sided p-values over 24 comparisons; down arrows denote lower-is-better diagnostics.

Stronger systems cover more requirements; they do not ask more precisely. Recall ranges from 44.9% to 84.7%, while Precision stays at 30.9–39.7%; every system spends 60–69% of its question units on questions that match no covered requirement. Recall and Ask-F1 track Quality ($\rho = 0.83$ and 0.88, Holm $p < 0.001$), and Recall tracks DSR ($\rho = 0.89$), whereas Precision tracks no delivery metric (Holm $p = 1.0$). Within systems and cases, higher-Recall repeats score higher in 45% and lower in 37% of 626 cells (mean $+0.031$, Wilcoxon $p = 0.012$). Coverage of each of the four requirement domains also correlates with Quality ($\rho = 0.81$–0.83); domain-level results and question-type shares are detailed in Appendix A.5. These observational comparisons do not isolate the causal effect of clarification.

Requirement coverage is concentrated in the first round. Average turns to clarify is 1.01–1.16, and first-turn coverage is 91–99% of final Recall. Under `Interact-Req`'s instruction to clarify before implementation, most covered requirements are addressed in the initial exchange.

More questions do not consistently improve delivery. Unmatched-question count correlates with DSR ($\rho = 0.75$, Holm $p = 0.006$): systems that ask more cover more requirements and also ask more unmatched questions. On the five most requirement-dense cases, mandatory rather than optional clarification raises question count for all eight tested systems and Recall for seven, but Quality for only five; the changes range from -0.125 to $+0.196$ and none is significant (Holm $p \geq 0.50$).

Case study: asking about a rule does not ensure compliance. In case 10 (retail replenishment), `opencode` + GPT-6-Astra asks whether same-city stock must be used before supplier purchases. Its local validator reports `pass`, but the submitted plan purchases an item from a supplier while the local warehouse still holds 327 units. The hidden evaluator identifies violations of this priority rule at eight stores, and the run receives Quality 0 (Appendix A.7). This run illustrates a gap between asking about a business rule and enforcing it in the final deliverable.

5.4 RQ3: Information Ablation

Table 2 compares `Hidden` (request and data only), `Interact-Req` (main runs reused), and `Full` (complete business information, no interaction) on 20 cases and eight systems; it reports paired differences: $\Delta_{\text{req}} = \texttt{Interact-Req} - \texttt{Hidden}$ and $\Delta_{\text{remain}} = \texttt{Full} - \texttt{Interact-Req}$.

Table 2: **Information ablation.** Case-macro Quality on 20 paired cases (three runs/condition). Bold marks maxima. Intervals and adjusted tests are in Appendix A.6.

Agent / model	Hidden	Interact-Req	Full	Δ_{req}	Δ_{remain}
General scaffold: `opencode`					
`opencode` + GPT-6-Astra	**0.1803**	**0.5800**	**0.7461**	**+0.3997**	+0.1662
`opencode` + DeepSeek-V4-Pro	0.1423	0.5063	0.6821	+0.3640	+0.1758
`opencode` + Gemini-3.1-Pro	0.1218	0.2912	0.4886	+0.1694	**+0.1974**
`opencode` + Kimi-K3	0.1244	0.3694	0.4488	+0.2450	+0.0794
Native scaffolds					
`codex` + GPT-6-Astra	0.1980	**0.5980**	**0.7946**	**+0.4000**	+0.1966
`deepseek-harness` + DeepSeek-V4-Pro	**0.1998**	0.5642	0.7339	+0.3644	+0.1697
`gemini` + Gemini-3.1-Pro	0.1231	0.3763	0.5184	+0.2532	+0.1420
`kimi` + Kimi-K3	0.1498	0.3693	0.7184	+0.2195	**+0.3490**

Interaction closes 39–76% of the score gap. Under `Hidden`, systems score 0.12–0.20; `Interact-Req` adds $+0.17$–$+0.40$ (significant for all eight, Holm over 24 tests), recovering 64–76% of the `Hidden`-to-`Full` Quality gap for six systems, 46% for `opencode` + Gemini-3.1-Pro, and 39% for `kimi` + Kimi-K3. `Full` exceeds `Interact-Req` for all eight ($+0.08$–$+0.35$; significant for five), although it removes interaction.

Full information leaves a substantial delivery gap. Under `Full`, mean Quality ranges from 0.449 to 0.795, leaving all eight systems below the FDE reference of 1.0. Providing complete business information improves delivery but is insufficient to reach the reference on average.

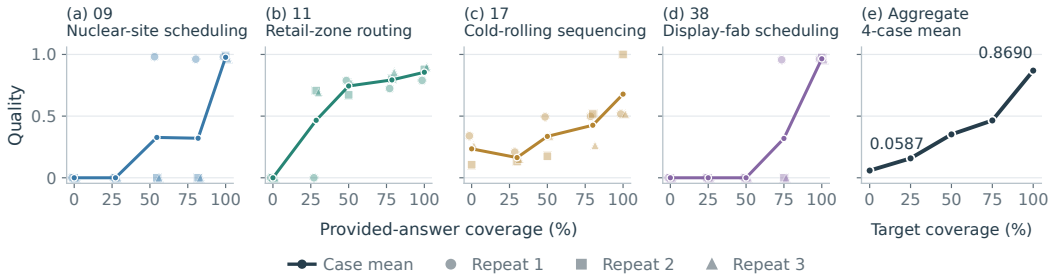


Figure 5: **Oracle information coverage.** Codex + GPT-6-Astra receives nested answer subsets; panels (a)–(d) show four cases and (e) their macro mean.

Oracle returns are nonlinear and case-dependent. We give `codex` + GPT-6-Astra nested random subsets of the registered answers on four cases, without interaction. At 100% coverage, agents receive all registered answers, not the complete business-information file used by `Full`. Mean Quality rises from 0.059 at 0% coverage to 0.352 at 50% and 0.869 at 100%; the last quarter of answers yields half of the gain. Case 38 scores zero in every run up to 50% coverage and 0.965 at full coverage, and at intermediate coverage cases 09 and 38 obtain a nonzero score in one of three repeats and score zero in the others (Figure 5). A few unresolved requirements can void an otherwise reasonable solution, which is consistent with RQ1: the highest Recall is 84.7%, yet no system exceeds 6.12% DSR.

6 Conclusion

We introduced FDE-Bench to evaluate agents' end-to-end delivery from underspecified real-world business requests. Its 49 cases use customer-accepted FDE solutions as delivery references. Across 18 configurations, the highest mean Quality is only 0.5510 against an FDE reference of 1.0. The highest delivery success rate is only 6.12%. Broader requirement coverage is associated with higher delivery quality, but even complete business information leaves all compared configurations below the FDE reference on average. Reliable FDE-level delivery therefore requires progress in both requirement clarification and solution optimization. FDE-Bench provides a shared testbed for comparing new agent systems and examining whether better requirement elicitation translates into better delivery.

7 Limitations and Future Work

FDE-Bench evaluates delivery with stable business requirements and fixed datasets. The simulated customer does not revise its requirements in response to intermediate solutions, whereas real customers may reveal or change expectations during development (Zowghi & Coulin, 2005). The benchmark therefore does not yet assess agents' ability to maintain delivery quality as production data and customer requirements evolve. Coverage is limited to 49 real customer projects in combinatorial optimization and machine learning. These cases do not cover all FDE settings.

We plan to expand FDE-Bench with additional real customer projects across more domains, including scientific and engineering applications. Selection will prioritize task types and business settings that are underrepresented in the current release. New cases will undergo the same requirement review and reference-solution checks. Future evaluations should incorporate continuously updated production data, evolving requirements, and human customers, testing both sustained delivery quality and the realism of simulated interactions. Evaluation in these settings should assess successive deliverables against the customer's current acceptance criteria and track how agents respond to requirement changes.

AI USE STATEMENT

In this research, generative AI models were not utilized for any tasks requiring mandatory disclosure, such as constructing synthetic datasets, deriving mathematical formulations, or generating proofs. The remaining required disclosure tasks are strictly inapplicable to our study. Our use of AI assistance was entirely limited to language editing and manuscript polishing to enhance readability and grammatical fluency. All AI-refined text underwent comprehensive manual review by the authors to ensure that our original scientific intent was preserved and no unintended alterations were introduced. The authors bear full responsibility for the final content of this manuscript, including all text refined with AI assistance.

REFERENCES

Ali Ahmaditeshnizi, Wenzhi Gao, and Madeleine Udell. OptiMUS: Scalable optimization modeling with (MI)LP solvers and large language models. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235, pp. 577–596. PMLR, 21–27 Jul 2024.

Jun Shern Chan, Neil Chowdhury, Oliver Jaffe, James Aung, Dane Sherburn, Evan Mays, Giulio Starace, Kevin Liu, Leon Maksin, Tejal Patwardhan, et al. Mle-bench: Evaluating machine learning agents on machine learning engineering. In *International Conference on Learning Representations*, volume 2025, pp. 50466–50494, 2025.

Hongzheng Chen, Yingheng Wang, Yaohui Cai, Hins Hu, Jiajie Li, Shirley Huang, Chenhui Deng, Rongjian Liang, Shufeng Kong, Haoxing Ren, et al. Heurigym: An agentic benchmark for llm-crafted heuristics in combinatorial optimization. In *International Conference on Learning Representations*, volume 2026, pp. 27466–27520, 2026.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Yizhe Chi, Deyao Hong, Dapeng Jiang, Tianwei Luo, Kaisen Yang, Boshi Zhang, Zhe Cao, Xiaoyan Fan, Bingxiang He, Han Hao, et al. Frontier-eng: Benchmarking self-evolving agents on real-world engineering tasks with generative optimization. *arXiv preprint arXiv:2604.12290*, 2026.

Jacob Cohen. A coefficient of agreement for nominal scales. *Educational and psychological measurement*, 20(1):37–46, 1960.

Bradley Efron. Bootstrap methods: another look at the jackknife. In *Breakthroughs in statistics: Methodology and distribution*, pp. 569–593. Springer, 1992.

Yongchang Fu, Xinjie Huang, Chengjun Dai, Chengzhe Feng, Junshao Zhang, and Hong Zhu. Opti-agent-bench: Benchmarking end-to-end optimization r&d agents on real-world business problems. *arXiv preprint arXiv:2607.10768*, 2026.

Xingang Guo, Yaxin Li, Xiangyi Kong, Yilan Jiang, Xiayu Zhao, Zhihua Gong, Yufan Zhang, Daixuan Li, Tianle Sang, Beixiao Zhu, et al. Toward engineering agi: Benchmarking the engineering design capabilities of llms. *Advances in Neural Information Processing Systems*, 38, 2026.

Sture Holm. A simple sequentially rejective multiple test procedure. *Scandinavian journal of statistics*, pp. 65–70, 1979.

Yuki Imajuku, Kohki Horie, Yoichi Iwata, Kensho Aoki, Naohiro Takahashi, and Takuya Akiba. Ale-bench: A benchmark for long-horizon objective-driven algorithm engineering. *Advances in Neural Information Processing Systems*, 38, 2026.

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, Peter Kirgis, Nitya Nadgir, Zachary Siegel, Boyi Wei, Tianci Xue, Ziru Chen, Felix Chen, Saiteja Utpala, et al. Holistic agent leaderboard: The missing infrastructure for ai agent evaluation. In *International Conference on Learning Representations*, volume 2026, pp. 98778–98849, 2026.

Jiajun Li, Mingshu Cai, Yixuan Li, Yu Ding, Ran Hou, Guanyu Nie, Xiongwei Han, and Wanyuan Wang. Oragentbench: Can llm agents solve challenging operations research tasks end to end? *arXiv preprint arXiv:2606.19787*, 2026a.

Jialin Li, Yuan Wu, and Yi Chang. Clareval: A benchmark for evaluating clarification skills of code agents under ambiguous instructions. *arXiv preprint arXiv:2603.00187*, 2026b.

Zhong Li, Hongliang Lu, Tao Wei, Yuxuan Chen, Wenyu Liu, Yuan Lan, Fan Zhang, and Zaiwen Wen. Constructing industrial-scale optimization modeling benchmark. In *Forty-third International Conference on Machine Learning*, 2026c.

Yang Liu, Dan Iter, Yichong Xu, Shuohang Wang, Ruochen Xu, and Chenguang Zhu. G-eval: Nlg evaluation using gpt-4 with better human alignment. In *Proceedings of the 2023 conference on empirical methods in natural language processing*, pp. 2511–2522, 2023.

Hongliang Lu, Zhonglin Xie, Yaoyu Wu, Can Ren, Yuxuan Chen, and Zaiwen Wen. OptMATH: A scalable bidirectional data synthesis framework for optimization modeling. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267, pp. 40769–40802. PMLR, 13–19 Jul 2025.

Mike Merrill, Alexander Shaw, Nicholas Carlini, Boxuan Li, Harsh Raj, Ivan Bercovich, Lin Shi, Jeong Shin, Thomas Walshe, E Kelly Buchanan, et al. Terminal-bench: Benchmarking agents on hard, realistic tasks in command line interfaces. In *International Conference on Learning Representations*, volume 2026, pp. 40903–40986, 2026.

Bashar Nuseibeh and Steve Easterbrook. Requirements engineering: a roadmap. In *Proceedings of the Conference on the Future of Software Engineering*, pp. 35–46, 2000.

Jane Pan, Ryan Shar, Jacob Pfau, Ameet Talwalkar, He He, and Valerie Chen. When benchmarks talk: Re-evaluating code llms with interactive feedback. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 24672–24700, 2025.

Rindranirina Ramamonjison, Timothy Yu, Raymond Li, Haley Li, Giuseppe Carenini, Bissan Ghaddar, Shiqi He, Mahdi Mostajabdaveh, Amin Banitalebi-Dehkordi, Zirui Zhou, et al. Nl4opt competition: Formulating optimization problems based on their natural language descriptions. In *NeurIPS 2022 competition track*, pp. 189–203. PMLR, 2023.

Hinrich Schütze, Christopher D Manning, and Prabhakar Raghavan. *Introduction to information retrieval*, volume 39. Cambridge University Press Cambridge, 2008.

Charles Spearman. *The proof and measurement of association between two things.* Appleton-Century-Crofts, 1961.

Weiwei Sun, Shengyu Feng, Shanda Li, and Yiming Yang. Co-bench: Benchmarking language model agents in algorithm search for combinatorial optimization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 33126–33134, 2026.

Tu Trinh, Mohamed Elfeki, Guangze Luo, Kelvin Luu, Nathan Hunt, Ernesto Hernandez, Nandan Marwaha, Yannis Yiming He, Charles Wang, Fernando Carabedo, et al. Hil-bench (human-in-loop benchmark): Do agents know when to ask for help? *arXiv preprint arXiv:2604.09408*, 2026.

Sanidhya Vijayvargiya, Xuhui Zhou, Akhila Yerukola, Maarten Sap, and Graham Neubig. Ambigswe: Interactive agents to overcome underspecificity in software engineering. In *International Conference on Learning Representations*, volume 2026, pp. 146716–146737, 2026.

Xingyao Wang, Zihan Wang, Jiateng Liu, Yangyi Chen, Lifan Yuan, Hao Peng, and Heng Ji. Mint: Evaluating llms in multi-turn interaction with tools and language feedback. In *International Conference on Learning Representations*, volume 2024, pp. 32593–32627, 2024.

Hjalmar Wijk, Tao Roa Lin, Joel Becker, Sami Jawhar, Neev Parikh, Thomas Broadley, Lawrence Chan, Michael Chen, Joshua M Clymer, Jai Dhyani, Elena Ericheva, Katharyn Garcia, Brian Goodrich, Nikola Jurkovic, Megan Kinniment, Aron Lajko, Seraphina Nix, Lucas Jun Koba Sato, William Saunders, Maksym Taran, Ben West, and Elizabeth Barnes. RE-bench: Evaluating frontier

AI r&d capabilities of language model agents against human experts. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267, pp. 66772–66832. PMLR, 13–19 Jul 2025.

Frank Wilcoxon. Individual comparisons by ranking methods. *Biometrics bulletin*, 1(6):80–83, 1945.

Zhicheng Yang, Yiwei Wang, Yinya Huang, Zhijiang Guo, Shi Shi, Xiongwei Han, Liang Feng, Linqi Song, Xiaodan Liang, and Jing Tang. Optibench meets resocratic: Measure and improve llms for optimization modeling. In *International Conference on Learning Representations*, volume 2025, pp. 24726–24759, 2025.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. In *13th International Conference on Learning Representations, ICLR 2025*, pp. 74824–74876, 2025.

Tong Zhang, Peixin Qin, Yang Deng, Chen Huang, Wenqiang Lei, Junhong Liu, Dingnan Jin, Hongru Liang, and Tat-Seng Chua. Clamber: A benchmark of identifying and clarifying ambiguous information needs in large language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 10746–10766, 2024.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in neural information processing systems*, 36:46595–46623, 2023.

Xin Zhou, Chun Yong Chong, Kisub Kim, Yun Peng, Rui Shu, Zihan Wu, Xu Han, Guowen Yuan, Zeyang Zhuang, Jounghoon Kim, Jeongjin Ju, Seongmin Ju, Taein Yoon, and David Lo. SWE-RPG: A unified issue resolution benchmark for requirement clarification, planning, and code generation for coding agents. *arXiv preprint arXiv:2608.09072*, 2026.

Didar Zowghi and Christine Coulin. Requirements elicitation: A survey of techniques, approaches, and tools. In *Engineering and managing software requirements*, pp. 19–46. Springer, 2005.

A Additional Evaluation Details

Positioning	Benchmark comparison (A.1).
Cases	Case map, scoring references, and clarification points (A.2).
Protocol	Delivery and process metrics and runtime settings (A.3).
Results	Full results (A.4), clarification types (A.5), ablations (A.6), and zero scores with a case study (A.7).
Reliability	Customer, judge, and extraction audits (A.8).
Prompts	Agent, customer, extractor, and judge templates (A.9).

A.1 Benchmark comparison

Table 3 compares FDE-Bench with representative benchmarks and systems (Section 2). Requirements denotes implicit business requirements annotated from customer–engineer discussions in real delivery projects. Reference denotes a customer-accepted project solution used as a delivery-quality baseline. These definitions concern the provenance of the requirements and reference; RE-Bench, for example, provides human expert comparisons and reference solutions, but not customer-accepted project deliverables.

Table 3: **Comparison with representative benchmarks and systems.** ✓: primary design feature; ✗: not a primary design feature. Requirements and Reference are defined above. Clarification: evaluation of clarifying questions or the use of user feedback; Execution: task outcomes verified through code or tool execution.

Benchmarks / systems	Task type	Requirements	Clarification	Execution	Reference
MIPLIB-NL (Li et al., 2026c)	Optimization modeling	✗	✗	✓	✗
OptiMUS (Ahmaditeshnizi et al., 2024)	Optimization modeling	✗	✗	✓	✗
OptiBench (Yang et al., 2025)	Optimization modeling	✗	✗	✓	✗
CO-Bench (Sun et al., 2026)	CO search	✗	✗	✓	✗
HeuriGym (Chen et al., 2026)	CO search	✗	✗	✓	✗
ALE-Bench (Imajuku et al., 2026)	CO search	✗	✗	✓	✗
ORAgentBench (Li et al., 2026a)	End-to-end OR	✗	✗	✓	✗
Opti-Agent-Bench (Fu et al., 2026)	End-to-end OR	✗	✗	✓	✗
MINT (Wang et al., 2024)	Interactive agents	✗	✓	✓	✗
τ-bench (Yao et al., 2025)	Interactive agents	✗	✓	✓	✗
CLAMBER (Zhang et al., 2024)	Clarification/HITL	✗	✓	✗	✗
HIL-Bench (Trinh et al., 2026)	Clarification/HITL	✗	✓	✓	✗
ClarEval (Li et al., 2026b)	Clarification/HITL	✗	✓	✓	✗
SWE-RPG (Zhou et al., 2026)	Requirement diagnosis	✗	✗	✓	✗
SWE-bench (Jimenez et al., 2024)	Execution agents	✗	✗	✓	✗
Terminal-Bench (Merrill et al., 2026)	Execution agents	✗	✗	✓	✗
Frontier-Eng (Chi et al., 2026)	Execution agents	✗	✗	✓	✗
EngDesign (Guo et al., 2026)	Execution agents	✗	✗	✓	✗
MLE-bench (Chan et al., 2025)	ML/R&D	✗	✗	✓	✗
RE-Bench (Wijk et al., 2025)	ML/R&D	✗	✗	✓	✗
FDE-Bench (Ours)	CO + ML	✓	✓	✓	✓

A.2 CASES AND CLARIFICATION POINTS

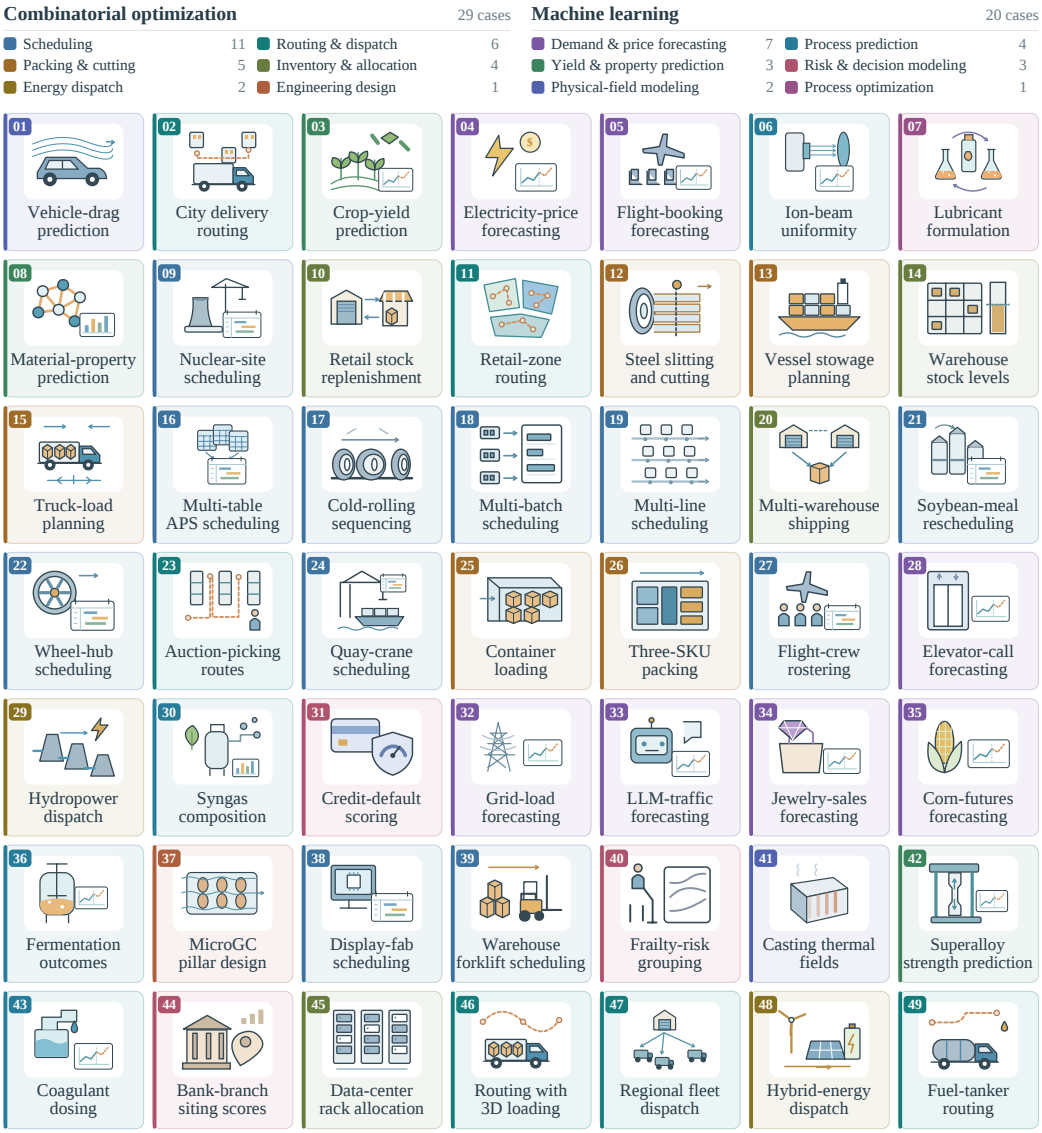


Figure 6: **Case map.** The 49 cases by task family and subtype, numbered 01–49.

Case map. Each case has one primary subtype for reporting; Figure 6 gives the counts.

Scoring references. Table 4 lists each case's raw objective, hard constraints, normalization direction, and FDE reference value R.

Table 4: **Case-level evaluator reference.** Numbering follows Figure 6.

No.	Case	Fam.	Raw objective	Hard constraints	Norm. / direction	FDE R
01	Vehicle-drag prediction	ML	mean relative C_d error (Cd_MRE)	coverage, unique finite rows; $0.05 \leq \mathrm{Cd} \leq 1.5$	R/x (min)	0.00357527
02	City delivery routing	CO	combined route cost (distance/time/penalties)	all required stops, feasible routes and valid timestamps	R/x (min)	281497
03	Crop-yield prediction	ML	yearly mean RMSE	required predictions, finite values and complete test coverage	R/x (min)	3.77281
04	Electricity-price forecasting	ML	weighted MAE/p95/p99/high-price underprediction error	complete horizon, finite values and aligned timestamps	R/x (min)	0.48004
05	Flight-booking forecasting	ML	seat-booking forecast score	complete horizon, finite nonnegative forecasts	x/R (max)	0.160653
06	Ion-beam uniformity	ML	mean WAPE-derived uniformity score	all required rows, finite predictions and valid bounds	x/R (max)	0.64572
07	Lubricant formulation	ML	weighted formulation quality (property/constraint terms)	valid formulation schema, bounds and required composition	x/R (max)	98.8783
08	Material-property prediction	ML	weighted material-property prediction quality	complete test coverage and finite predictions	x/R (max)	0.945334
09	Nuclear-site scheduling	CO	weighted construction schedule score	schema, precedence/resource feasibility and horizon limits	x/R (max)	97.2044
10	Retail stock replenishment	CO	fulfillment/lateness/cost absolute score	complete orders and feasible inventory/transfer records	x/R (max)	0.997476
11	Retail-zone routing	CO	partition plus routing composite quality	all zones/items covered and routing/partition checks pass	x/R (max)	0.837337
12	Steel slitting and cutting	CO	cut utilization and cut-count score	valid cut plan, demand coverage and capacity checks	x/R (max)	1.6645
13	Vessel stowage planning	CO	weighted vessel-stowage objective components	schema, placement, capacity and stability checks	x/R (max)	0.500878
14	Warehouse stock levels	CO	mean daily policy score minus variability penalty	valid daily policy schema and simulator checks	x/R (max)	105.697
15	Truck-load planning	CO	0.60 coverage $+0.25$ length utilization $+0.15$ balance	all items assigned once and truck/capacity checks	x/R (max)	1
16	Multi-table APS scheduling	CO	delay/utilization absolute schedule score	all jobs assigned and precedence/resource constraints	x/R (max)	0.114446
17	Cold-rolling sequencing	CO	coil-sequencing penalty	complete sequence, no duplicates and transition feasibility	R/x (min)	10718.9
18	Multi-batch scheduling	CO	multi-batch scheduling composite score	schema, assignment, precedence and capacity checks	x/R (max)	0.998438
19	Multi-line scheduling	CO	total production penalty	complete schedule and line/time feasibility	R/x (min)	637600
20	Multi-warehouse shipping	CO	lexicographic shipping score with tie-break cost	all orders served and inventory/route constraints	x/R (max)	0.674404

continued on next page

Table 4: Case-level evaluator reference (continued).

No.	Case	Fam.	Raw objective	Hard constraints	Norm. / direction	FDE R
21	Soybean-meal rescheduling	CO	weighted smoothness/safety/balance rescheduling score	valid schedule and safety/quality checks	x/R (max)	87.061
22	Wheel-hub scheduling	CO	total weighted delay	two-stage schedule complete and all hard constraints	R/x (min)	924
23	Auction-picking routes	CO	picking-route distance	all required picks served with valid route	R/x (min)	237738
24	Quay-crane scheduling	CO	$100{,}000{,}000\, n_{\mathrm{cranes}}$ + total move seconds	all hard quay-crane constraints C1–C7	R/x (min)	1.1e9
25	Container loading	CO	container stability/loading quality	all items placed once; weight/geometry/stability checks	x/R (max)	62.0141
26	Three-SKU packing	CO	validated ITEM_C count/value	valid container schema and packing/weight checks	x/R (max)	148.006
27	Flight-crew rostering	CO	crew-roster combined score	complete roster and labor/rest/qualification rules	x/R (max)	4624.98
28	Elevator-call forecasting	ML	$100/(1+\mathrm{MAE})$ minus skew penalty	complete calls, finite nonnegative predictions	x/R (max)	21.7277
29	Hydropower dispatch	CO	total cascade-hydropower energy	complete dispatch, bounds and cascade constraints	x/R (max)	160732
30	Syngas composition	ML	mean R^2 across syngas targets	complete target coverage and finite predictions	x/R (max)	0.846809
31	Credit-default scoring	ML	$0.5\,\mathrm{AUC} + 0.5\,\mathrm{KS}$	complete labels/probabilities and valid score range	x/R (max)	0.865552
32	Grid-load forecasting	ML	MAPE	complete time horizon and finite forecasts	R/x (min)	0.374465
33	LLM-traffic forecasting	ML	traffic RMSE	complete horizon and finite forecasts	R/x (min)	232222
34	Jewelry-sales forecasting	ML	0.5 quantity score $+0.5$ F1	complete SKU coverage and valid nonnegative forecasts	x/R (max)	0.193193
35	Corn-futures forecasting	ML	mean forecasting error	complete horizon and finite forecasts	R/x (min)	0.294579
36	Fermentation outcomes	ML	fermentation MAE	complete targets and finite predictions	R/x (min)	0.435077
37	MicroGC pillar design	CO	$0.6s_{\mathrm{nrmse}} + 0.2s_{\mathrm{press}} + 0.2s_{\mathrm{perim}}$	valid geometry/field outputs and CFD checks	x/R (max)	0.62696
38	Display-fab scheduling	CO	$60p_{\mathrm{ach}} + 30o_{\mathrm{ach}} + 10q_{\mathrm{score}}$	complete schedule and machine/order constraints	x/R (max)	98.6208
39	Warehouse forklift scheduling	CO	nested warehouse objective	assignment, stack/forklift and temporal constraints	R/x (min)	39.4605
40	Frailty-risk grouping	ML	$W_{\mathrm{FIT}}R^2 + W_{\mathrm{RISK}}(c-0.5)$	complete cohort outputs and valid risk/fit fields	x/R (max)	0.845504
41	Casting thermal fields	ML	mean thermal-field RMSE	complete field output and finite values	R/x (min)	0.18952
42	Superalloy strength prediction	ML	superalloy prediction error	complete target coverage and finite values	R/x (min)	13.3398
43	Coagulant dosing	ML	water-coagulant dosing error	complete horizon and finite values	R/x (min)	0.21319
44	Bank-branch siting scores	ML	branch-siting score MAE	all branches scored with valid predictions	R/x (min)	7.12993

continued on next page

Table 4: Case-level evaluator reference (continued).

No.	Case	Fam.	Raw objective	Hard constraints	Norm. / direction	FDE R
45	Data-center rack allocation	CO	rack-allocation total cost	all racks assigned and capacity/power constraints	R/x (min)	28.7197
46	Routing with 3D loading	CO	$100{,}000\, n_{\text{vehicles}}$ + routing distance	packing and route feasibility	R/x (min)	1.502e6
47	Regional fleet dispatch	CO	regional dispatch total objective	all orders assigned; violations penalized in objective	R/x (min)	1.452e6
48	Hybrid-energy dispatch	CO	8,760-hour total income	complete dispatch and operational constraints	x/R (max)	1.854e8
49	Fuel-tanker routing	CO	multi-compartment tanker distance	all demands served and vehicle/compartment constraints	R/x (min)	2.827e6

Clarification points. A clarification point is retained only if (i) it cannot be resolved from the agent-visible request and data, (ii) the customer's answer can change a feasible decision, the objective, or the evaluator's acceptance judgment, and (iii) the answer can be stated as a fact, rule, or convention that the evaluator applies to the artifact. Implementation preferences, facts visible in the workspace, and questions about the agent's method are excluded.

Splitting and merging. A candidate is split when two answers change different decisions or evaluator checks, e.g., the error metric, which changes the score, and the valid value range, which changes whether the artifact is accepted. Paraphrases and examples of the same decision are merged into one point. Table 5 lists boundary cases.

Table 5: **Boundary examples for clarification points.**

Candidate	Decision	Reason
"Is error MAE or relative error?"	Keep as one valid point	Changes the evaluator's scoring convention.
"Does same-city supplier precedence override cost minimization?"	Keep as one valid point	Changes a routing/fulfillment rule and can change feasibility.
"Are out-of-range predictions rejected?"	Keep as one valid point	Changes a hard constraint.
"What Python library should I use?"	Exclude	Implementation choice is not scored.
"How many rows are in the visible CSV?"	Exclude	The fact is already observable to the agent.
"Which metric and which value range are required?"	Split into two points	Metric changes the score; range changes acceptance.
"Should same-city orders precede transfers, and should they be grouped?"	Merge when the evaluator has one precedence rule; split otherwise	The split is determined by independent counterfactual answers, not punctuation.

Registry fields. Each point in `gt.json` has a key and four fields: the requirement description, issue type (missing, ambiguous, invalidated, or conflicting), content domain (scoring convention, constraint rule, computation convention, or data semantics), and rule source (`null` if none).

Review protocol. The FDE responsible for each original delivery annotated the requirements established in the customer–FDE discussions. A second review applied the same criteria against the evaluator and reference artifact, removed duplicates, and confirmed that each point had a customer answer; disagreements were resolved against the evaluator code.

A.3 METRICS AND RUNTIME SETTINGS

Score selection. Up to five extraction routes score each artifact, with remaining routes skipped once three scores agree after rounding to four decimal places. Extractions without source provenance are rejected; majority agreement alone does not prove that every decision was preserved (Appendix A.8). Without a majority, the score is unavailable.

Aggregation. A run is valid when its artifact satisfies the case's hard constraints (Table 4); pass@3 and pass^3 count valid repeats. Zero scores and invalid runs are included in Quality. Each main configuration contributes three scored runs on each of 49 cases, with all 147 runs included in DSR (Section 4.2). All ablation runs also have numeric scores. Clarification metrics are never imputed, and runtime, tool calls, and cost are not part of any delivery metric.

Question units. Agent messages are split at question marks into question units; a message without a question mark counts as one unit. Turn indices come from the interaction log.

Coverage judgments. Judges also receive annotated trigger examples as matching hints. Merely touching a requirement's topic does not count as coverage.

A covered point's matched questions are those cited by the agreeing judges, and Precision counts distinct matched questions.

Undefined values. With no recorded questions, Precision and Ask-F1 are undefined and Recall is zero. If every judgment in a run fails, all three metrics are unavailable; each metric is averaged only over runs where it is defined.

Runtime. Each solving run has 43,200 seconds, including interaction, and at most 30 clarification rounds, with no cap on the number of questions or on cumulative tokens. OpenHands runs at most 200 agent steps (`max_iterations`); these steps are distinct from customer exchanges. OpenCode passes a per-call output limit of 131,072 tokens with a 1,000,000-token context. The native CLIs use SDK or provider defaults. OpenCode sets neither temperature nor reasoning effort, and no run overrides the temperature. Exact runner, scaffold, and dependency pins are provided with the artifact; the controls above are the settings relevant to interpreting the experiments.

A.4 FULL RESULTS

Entries are means with 95% case-bootstrap intervals (2,000 resamples, seed 12345), following the bootstrap framework of Efron (Efron, 1992).

Delivery metrics. Table 6(a) gives delivery results with intervals. Quality is case-macro and DSR is run-level; pass@3 and pass^3 are case-level (Section 4.2). Every pair of unpaired Quality intervals overlaps, because the intervals carry the variance between cases. Paired by case, the systems form two groups. The 12 systems built on GPT-6-Astra, GLM-5.3, Qwen3.8-Max, Grok-4.6, and DeepSeek-V4-Pro score 0.499–0.551; the three Kimi-K3 systems and Gemini-3.1-Pro under `opencode` and `gemini` score 0.387–0.420. All 60 differences between the groups exclude zero under a paired case bootstrap, and none of the 76 differences within a group does. All differences above 0.08 are resolved and none of 0.04 or less, so 49 cases separate models but not scaffolds of the same model. The exception is `openhands` + Gemini-3.1-Pro (0.457), which exceeds the other two Gemini-3.1-Pro systems by 0.053 and 0.070 (both resolved). DSR does not rank systems: 15 of the 18 DSR intervals include zero.

Clarification metrics. Clarification metrics are run-level (Section 4.2; Table 6(b)). Across the 18 systems, Spearman ρ between Recall and Quality is 0.83; after subtracting each model's mean over its scaffolds, it is -0.31 ($p = 0.22$). Moving a model from `opencode` to `openhands` raises Recall for six of seven models (DeepSeek-V4-Pro: 55.1% to 65.2%) and Redund. for five (Qwen3.8-Max: 16.3 to 22.6), lowers Precision for six, and lowers Quality for five, by at most 0.032. Scaffold changes that increase coverage do not consistently improve Quality; the policy ablation also shows mixed Quality changes (Appendix A.6).

Table 6: **Complete results.** Estimates with 95% case-bootstrap intervals; bold marks blockwise best values.

(a) Delivery

Agent / model	Quality	DSR	pass@3	pass^3
General scaffold: `opencode`				
GPT-6-Astra	**0.546** [0.448, 0.647]	**5.4** [0.7, 12.2]	91.8 [83.7, 98.0]	**81.6** [71.4, 91.8]
GLM-5.3	0.541 [0.445, 0.632]	1.4 [0.0, 3.4]	91.8 [83.7, 98.0]	77.6 [65.3, 87.8]
Qwen3.8-Max	0.539 [0.444, 0.629]	2.7 [0.0, 6.1]	**95.9** [89.8, 100.0]	73.5 [61.2, 85.7]
Grok-4.6	0.522 [0.423, 0.621]	2.7 [0.7, 5.4]	87.8 [77.6, 95.9]	79.6 [67.3, 89.8]
DeepSeek-V4-Pro	0.499 [0.404, 0.593]	1.4 [0.0, 4.1]	91.8 [83.7, 98.0]	71.4 [59.2, 83.7]
Kimi-K3	0.419 [0.336, 0.503]	0.0 [0.0, 0.0]	91.8 [83.7, 98.0]	53.1 [38.8, 67.3]
Gemini-3.1-Pro	0.387 [0.309, 0.470]	0.0 [0.0, 0.0]	89.8 [81.6, 98.0]	59.2 [46.9, 73.5]
General scaffold: `openhands`				
GPT-6-Astra	**0.530** [0.427, 0.624]	4.8 [0.0, 10.9]	**93.9** [87.8, 100.0]	67.3 [53.1, 79.6]
GLM-5.3	0.527 [0.431, 0.615]	2.0 [0.0, 5.4]	91.8 [83.7, 98.0]	67.3 [55.1, 79.6]
Grok-4.6	0.516 [0.418, 0.601]	0.7 [0.0, 2.0]	91.8 [83.7, 98.0]	**71.4** [59.2, 83.7]
DeepSeek-V4-Pro	0.510 [0.418, 0.598]	0.7 [0.0, 2.0]	**93.9** [85.7, 100.0]	**71.4** [59.2, 83.7]
Qwen3.8-Max	0.507 [0.421, 0.592]	**6.1** [1.4, 11.6]	**93.9** [87.8, 100.0]	65.3 [53.0, 77.6]
Gemini-3.1-Pro	0.457 [0.360, 0.548]	0.0 [0.0, 0.0]	87.8 [77.6, 95.9]	65.3 [51.0, 77.6]
Kimi-K3	0.412 [0.331, 0.496]	0.7 [0.0, 2.0]	85.7 [75.5, 93.9]	57.1 [42.9, 71.4]
Native scaffolds				
`codex` + GPT-6-Astra	**0.551** [0.448, 0.647]	**4.1** [0.0, 9.5]	**93.9** [85.7, 100.0]	**75.5** [63.3, 87.8]
`deepseek-harness` + DeepSeek-V4-Pro	0.527 [0.427, 0.619]	1.4 [0.0, 3.4]	91.8 [83.7, 98.0]	**75.5** [63.3, 87.8]
`kimi` + Kimi-K3	0.420 [0.333, 0.502]	0.7 [0.0, 2.0]	89.8 [81.6, 98.0]	59.2 [44.9, 73.5]
`gemini` + Gemini-3.1-Pro	0.404 [0.319, 0.494]	0.0 [0.0, 0.0]	87.8 [77.6, 95.9]	57.1 [42.9, 71.4]

(b) Clarification

Agent / model	Recall	Precision	Ask-F1	KQC_1	ATC ↓	Redund. ↓
General scaffold: `opencode`						
GPT-6-Astra	**83.6** [79.5, 87.6]	36.5 [32.1, 41.1]	**48.8** [44.3, 53.7]	**78.1** [73.3, 82.6]	1.07 [1.04, 1.11]	15.6 [13.7, 17.8]
GLM-5.3	66.8 [60.7, 72.6]	**39.7** [35.3, 44.2]	47.5 [43.2, 51.8]	62.8 [57.2, 68.7]	1.05 [1.02, 1.08]	10.1 [8.7, 11.8]
Qwen3.8-Max	70.6 [65.4, 76.0]	35.4 [31.5, 39.6]	45.1 [40.8, 49.5]	68.0 [62.8, 73.3]	1.03 [1.01, 1.06]	16.3 [14.0, 19.1]
Grok-4.6	68.2 [62.3, 74.3]	38.3 [34.0, 43.0]	46.4 [42.3, 50.6]	65.2 [58.9, 71.8]	1.04 [1.02, 1.08]	8.8 [7.4, 10.5]
DeepSeek-V4-Pro	55.1 [48.7, 61.9]	37.9 [33.3, 42.8]	42.7 [38.1, 47.3]	52.9 [46.4, 59.9]	1.04 [1.01, 1.07]	8.3 [7.2, 9.6]
Kimi-K3	55.0 [49.3, 60.8]	37.6 [33.8, 41.6]	43.3 [39.6, 47.3]	53.6 [48.3, 59.5]	**1.02** [1.01, 1.04]	10.0 [8.4, 11.7]
Gemini-3.1-Pro	47.1 [41.3, 52.9]	39.5 [34.3, 44.8]	40.4 [35.8, 45.4]	43.0 [37.4, 48.7]	1.16 [1.06, 1.29]	**7.5** [6.0, 9.0]
General scaffold: `openhands`						
GPT-6-Astra	**84.7** [80.5, 88.5]	36.8 [32.7, 41.3]	**49.5** [44.8, 54.0]	**80.2** [75.2, 85.1]	1.06 [1.03, 1.10]	14.4 [13.0, 15.8]
GLM-5.3	67.1 [61.1, 73.0]	36.8 [32.4, 41.1]	45.5 [41.1, 49.9]	64.3 [58.2, 70.7]	1.04 [1.02, 1.06]	12.6 [10.7, 14.8]
Grok-4.6	69.9 [64.0, 75.9]	37.0 [32.7, 41.0]	45.9 [41.9, 49.8]	66.7 [60.4, 73.1]	1.05 [1.02, 1.08]	11.0 [9.1, 13.2]
DeepSeek-V4-Pro	65.2 [58.6, 72.0]	36.4 [31.6, 40.9]	44.9 [40.1, 49.4]	64.3 [57.6, 71.1]	**1.01** [1.00, 1.02]	11.0 [9.6, 12.6]
Qwen3.8-Max	75.4 [69.8, 80.7]	30.9 [27.0, 34.5]	42.2 [38.0, 46.5]	74.0 [68.1, 79.7]	1.02 [1.01, 1.03]	22.6 [19.9, 25.6]
Gemini-3.1-Pro	45.5 [39.0, 52.1]	**37.8** [32.6, 42.8]	39.2 [33.9, 44.5]	42.2 [36.2, 48.5]	1.06 [1.03, 1.09]	**6.3** [5.3, 7.4]
Kimi-K3	57.9 [52.4, 63.4]	35.4 [31.7, 39.1]	41.3 [37.8, 44.9]	57.1 [51.6, 62.9]	1.02 [1.01, 1.04]	11.2 [10.0, 12.5]
Native scaffolds						
`codex` + GPT-6-Astra	**84.7** [80.8, 88.7]	37.9 [33.2, 42.6]	**50.1** [45.6, 54.6]	**79.4** [74.9, 83.7]	1.06 [1.03, 1.09]	13.9 [12.2, 15.9]
`deepseek-harness` + DeepSeek-V4-Pro	61.8 [55.3, 68.1]	**39.3** [34.8, 43.8]	45.9 [41.0, 50.2]	59.9 [53.5, 66.4]	1.03 [1.01, 1.06]	9.6 [8.1, 11.4]
`kimi` + Kimi-K3	55.7 [49.3, 62.3]	36.7 [32.4, 41.1]	42.9 [38.3, 47.3]	54.2 [47.9, 60.8]	1.03 [1.01, 1.05]	9.8 [8.3, 11.6]
`gemini` + Gemini-3.1-Pro	44.9 [39.3, 50.6]	38.8 [33.6, 44.1]	39.4 [34.5, 44.2]	44.2 [38.4, 50.3]	**1.01** [1.00, 1.03]	**6.0** [5.1, 6.9]

Compute. Provider cost was available for 15 of 18 configurations and showed no detectable association with Quality; detailed accounting is included with the artifact rather than repeated here.

A.5 CLARIFICATION TYPES

Each clarification point in `gt.json` carries an issue type and a content domain (Table 7). These labels are hidden from agents and judges and are not used by the primary scorer.

Table 7: **Composition of the clarification-point registry.** 266 points in 49 cases.

Annotation family	Type	Points	Share	Cases
Issue type	Missing	207	77.8%	48
	Ambiguous	45	16.9%	30
	Invalidated	9	3.4%	8
	Conflicting	5	1.9%	4
Content domain	Scoring convention	78	29.3%	39
	Constraint rule	65	24.4%	33
	Computation convention	62	23.3%	27
	Data semantics	61	22.9%	35

Rule-source labels are too sparse (246 unlabeled) to analyze. Type-specific Recall is the share of a case's points of that type that are covered, averaged over repeats and then over cases containing the type, and correlated with Quality across the 18 configurations (Table 8(a)).

Table 8: **Issue types and delivery.** Spearman ρ with case-bootstrap 95% intervals.

(a) Type-specific Recall and Quality across 18 configurations; Holm correction over four tests

Issue type	Mean Recall	Quality ρ	95% CI	Holm p
Missing	66.3%	0.808	[0.467, 0.882]	< 0.001
Ambiguous	60.5%	0.801	[0.423, 0.884]	< 0.001
Invalidated	60.5%	0.513	[0.004, 0.707]	0.029
Conflicting	77.5%	0.641	[0.101, 0.705]	0.008

(b) Issue-type share and delivery across 49 cases

Issue type	Cases with type	Quality ρ [95% CI]	DSR ρ [95% CI]
Missing	48	0.032 [-0.253, 0.341]	-0.092 [-0.322, 0.161]
Ambiguous	30	-0.081 [-0.358, 0.209]	0.018 [-0.212, 0.260]
Invalidated	8	0.010 [-0.274, 0.283]	0.414 [0.021, 0.747]
Conflicting	4	-0.102 [-0.357, 0.216]	0.085 [-0.159, 0.457]

Table 9 breaks these Recall values down by configuration.

The ranking of systems is stable across issue types. Missing and Ambiguous Recall have Spearman $\rho = 0.95$ over the 18 configurations, and GPT-6-Astra has the highest and Gemini-3.1-Pro the lowest of both in every scaffold block. Invalidated points are the exception: Qwen3.8-Max ranks first on them in both general scaffolds (81.2% and 85.4%) although its Missing Recall is 10–17 points below GPT-6-Astra; the Invalidated and Conflicting columns rest on 8 and 4 cases. Within a case, the issue type does not decide which points are covered. In the 29 cases with both Missing and Ambiguous points, 16 of 18 systems cover Ambiguous points less often (mean gap 7.2 points), but the gap reverses in 12 of the 29 cases (95% case-bootstrap CI $[-3.8, 17.7]$).

The content domain does. Pooled over runs, agents cover 72.6% of constraint-rule points, 64.8% of computation conventions, 56.6% of scoring conventions, and 56.6% of data-semantics points, and constraint rules have the highest Recall for 16 of 18 systems. Within the same case, constraint rules are covered 14.6 points more often than data semantics (22 cases; CI $[3.9, 25.0]$) and 15.9 points more than computation conventions (22 cases; $[2.8, 28.9]$); the gap to scoring conventions ($+12.5$,

Table 9: **Type-specific Recall (%) by configuration.** Bold marks blockwise maxima.

Agent / model	Overall	Missing	Ambiguous	Invalidated	Conflicting
General scaffold: `opencode`					
GPT-6-Astra	**83.64**	**87.8**	**75.7**	64.6	**95.8**
GLM-5.3	66.83	69.0	63.1	56.2	75.0
Qwen3.8-Max	70.56	70.7	67.8	**81.2**	83.3
Grok-4.6	68.19	71.2	66.5	64.6	70.8
DeepSeek-V4-Pro	55.08	58.0	46.5	52.1	70.8
Kimi-K3	55.03	58.1	48.3	62.5	75.0
Gemini-3.1-Pro	47.10	49.9	43.9	41.7	54.2
General scaffold: `openhands`					
GPT-6-Astra	**84.67**	**87.2**	**78.3**	75.0	87.5
GLM-5.3	67.12	68.9	58.9	62.5	75.0
Grok-4.6	69.91	71.7	64.4	72.9	83.3
DeepSeek-V4-Pro	65.24	65.6	63.7	60.4	87.5
Qwen3.8-Max	75.37	77.2	68.0	**85.4**	**100.0**
Gemini-3.1-Pro	45.45	45.7	46.5	33.3	50.0
Kimi-K3	57.94	59.9	52.0	68.8	62.5
Native scaffolds					
`codex` + GPT-6-Astra	**84.70**	**87.9**	**80.0**	**70.8**	**100.0**
`deepseek-harness` + DeepSeek-V4-Pro	61.79	62.3	62.4	56.2	83.3
`kimi` + Kimi-K3	55.69	56.9	55.7	52.1	83.3
`gemini` + Gemini-3.1-Pro	44.94	45.8	46.5	29.2	58.3

26 cases) has an interval that includes zero. Agents ask about stated business rules more than about how fields are read and how quantities are computed. Covering a constraint rule also has the largest association with delivery: within the same system and case, runs that cover a constraint-rule point score 0.059 higher than runs that do not (292 system–case–point cells with mixed coverage; Wilcoxon $p = 0.006$), against 0.027–0.035 for the other domains.

Across systems, coverage of every domain tracks Quality equally (Spearman $\rho = 0.81$–0.83), and the two groups of Appendix A.4 differ by 16.2–20.7 points in every domain. Table 10 gives the domain mix of each model's matched questions; a question matched to points of several domains is split equally among them. GPT-6-Astra is the only model that directs more of its matched questions to scoring conventions (28.8%) than to constraint rules (26.7%); DeepSeek-V4-Pro, Gemini-3.1-Pro, and Grok-4.6 direct 20.1–21.7% to scoring conventions and 31.5–33.0% to constraint rules. Over the 18 systems, the scoring share correlates with Quality ($\rho = 0.61$) and the constraint share with lower Quality ($\rho = -0.60$; Holm $p = 0.028$ for both over four domains). Neither association is robust: the case-bootstrap 95% intervals include zero ($[-0.07, 0.64]$ and $[-0.69, 0.08]$), and without the three GPT-6-Astra systems the correlations fall to 0.38 ($p = 0.16$) and -0.32 ($p = 0.24$).

Table 10: **Domain mix of matched questions by model (%).** Quality is the mean over the model's systems.

Model	Constraint	Computation	Scoring	Data	Quality
GPT-6-Astra	26.7	25.4	28.8	19.1	0.542
GLM-5.3	30.7	24.5	26.6	18.3	0.534
Qwen3.8-Max	28.1	27.4	26.3	18.2	0.523
Grok-4.6	31.5	29.7	21.7	17.2	0.519
DeepSeek-V4-Pro	33.0	27.9	20.1	19.0	0.512
Kimi-K3	31.1	26.1	24.6	18.2	0.417
Gemini-3.1-Pro	33.0	29.6	20.4	17.0	0.416

Table 8(b) instead correlates each case's share of points of a type with its delivery, averaged over configurations, across all 49 cases.

No Quality interval excludes zero, so the type mix of a case does not predict its difficulty. The positive Invalidated–DSR association rests on eight cases.

A.6 ABLATIONS

Scope. The three-condition ablation uses a 20-case subset. The policy ablation uses the five cases in that subset with the most clarification points; exact case IDs and registry counts are provided with the artifact.

Information conditions. Table 11 expands Table 2. Each estimate averages 20 paired case differences. Intervals are 95% case-bootstrap intervals (2,000 resamples, seed 12345); * marks two-sided Wilcoxon signed-rank tests that remain significant after Holm correction over the 24 comparisons.

Table 11: **Paired Quality differences between information conditions.** *: Holm-adjusted $p < 0.05$.

Agent / model	Δ_{req}	Δ_{remain}	Δ_{total}
`opencode` + GPT-6-Astra	+0.3997* [0.2284, 0.5983]	+0.1662 [0.0369, 0.3167]	+0.5658* [0.3941, 0.7367]
`opencode` + DeepSeek-V4-Pro	+0.3640* [0.2367, 0.5046]	+0.1758* [0.0893, 0.2733]	+0.5398* [0.4026, 0.6741]
`opencode` + Gemini-3.1-Pro	+0.1694* [0.0824, 0.2684]	+0.1974* [0.0810, 0.3276]	+0.3668* [0.2266, 0.5130]
`opencode` + Kimi-K3	+0.2450* [0.1326, 0.3595]	+0.0794 [-0.0125, 0.1740]	+0.3244* [0.2137, 0.4438]
`codex` + GPT-6-Astra	+0.4000* [0.2211, 0.5751]	+0.1966* [0.0853, 0.3228]	+0.5966* [0.4348, 0.7473]
`deepseek-harness` + DeepSeek-V4-Pro	+0.3644* [0.2344, 0.5002]	+0.1697* [0.0689, 0.2949]	+0.5341* [0.3960, 0.6677]
`gemini` + Gemini-3.1-Pro	+0.2532* [0.1422, 0.3757]	+0.1420 [0.0189, 0.2758]	+0.3952* [0.2680, 0.5216]
`kimi` + Kimi-K3	+0.2195* [0.1044, 0.3488]	+0.3490* [0.2226, 0.4809]	+0.5685* [0.4156, 0.7132]

Δ_{req} and Δ_{total} are significant for all eight systems; Δ_{remain} is significant for five. What interaction recovers depends on how many requirements the system asks about; what remains does not. Over the eight systems, Δ_{req} rises with Recall on the same 20 cases (Spearman $\rho = 0.86$, $p = 0.007$), while Δ_{remain} has $\rho = 0.00$. The two Kimi-K3 systems show the difference: with Recall of 60.4% and 60.7% and the same `Interact-Req` Quality (0.369), the native `kimi` CLI reaches 0.718 under `Full` and `opencode` 0.449, so Δ_{remain} is $+0.349$ and $+0.079$.

Clarification policy. Both policies are run on the five policy cases with the eight ablation configurations (15 runs per configuration and policy). `Interact` lets the agent decide whether to ask; `Interact-Req` requires clarification before implementation. #Q counts question units, and ask rate is the share of runs with at least one question.

Without the requirement, agents still ask in 80–100% of runs, but they ask fewer questions: requiring clarification multiplies #Q by 1.4–3.0 in all eight systems. The added questions are worth less than the ones agents ask on their own. Measured as the change in matched questions over the change in #Q, they match a covered requirement at 13.5–45.8%, below the `Interact` Precision of the same system in all eight. Recall rises in seven systems, but by at most 2.1 points per added question. Agents ask their most useful questions first. Quality changes are mixed (Table 12(b)): no Holm-adjusted p is below 0.5.

Oracle answer coverage. We selected four cases (09, 11, 17, and 38) and ran `codex` + GPT-6-Astra without a clarification channel. We evaluate five target coverage levels, $r \in \{0, 0.25, 0.5, 0.75, 1\}$, using nested subsets of the registered answers and three repeats per case and level. The 0% level is run separately from `Hidden`; the 100% level supplies the registered answers, not the complete information file used by `Full`. Table 13 gives the values plotted in Figure 5; exact subset construction is provided with the artifact.

Answer coverage improves Quality overall, but the case-level trends are non-monotonic. Specific answers matter more than the number of answers, which also explains the high variance at intermediate coverage levels. The registered-answer endpoint differs from `Full`, and coverage of a blocking point is necessary but not sufficient for correct implementation; the table therefore provides an information-sensitivity analysis rather than a claim that more answers always improve delivery.

Table 12: **Optional versus required clarification** on five cases.

(a) Metrics by condition

Condition	Quality	Ask rate	#Q	Recall	Precision	Ask-F1
`opencode` + *GPT-6-Astra*						
`Interact`	0.3262	100.00%	15.00	63.61%	47.72%	53.84%
`Interact-Req`	0.3830	100.00%	36.27	85.24%	49.04%	61.19%
`opencode` + *DeepSeek-V4-Pro*						
`Interact`	0.4061	100.00%	10.20	37.18%	56.92%	43.73%
`Interact-Req`	0.5552	100.00%	21.13	49.76%	53.51%	50.62%
`opencode` + *Gemini-3.1-Pro*						
`Interact`	0.3304	93.33%	10.07	39.45%	53.15%	46.42%
`Interact-Req`	0.2742	100.00%	14.47	38.53%	47.24%	41.01%
`opencode` + *Kimi-K3*						
`Interact`	0.0992	80.00%	8.00	23.67%	46.33%	34.53%
`Interact-Req`	0.2955	100.00%	24.27	52.59%	43.83%	45.28%
`codex` + *GPT-6-Astra*						
`Interact`	0.4612	100.00%	16.27	69.97%	51.47%	58.61%
`Interact-Req`	0.3911	100.00%	31.27	81.03%	48.50%	59.43%
`deepseek-harness` + *DeepSeek-V4-Pro*						
`Interact`	0.4269	80.00%	8.33	33.09%	57.05%	46.32%
`Interact-Req`	0.5579	100.00%	18.60	53.26%	55.03%	53.50%
`gemini` + *Gemini-3.1-Pro*						
`Interact`	0.2536	93.33%	8.00	30.73%	47.06%	38.12%
`Interact-Req`	0.3424	100.00%	13.07	41.19%	51.12%	44.28%
`kimi` + *Kimi-K3*						
`Interact`	0.3603	100.00%	8.07	38.17%	56.13%	43.33%
`Interact-Req`	0.2351	86.67%	19.47	41.54%	44.32%	43.57%

(b) Quality change from required clarification, with Holm-adjusted p-values

Agent / model	Req−Interact	95% CI	Holm p
`opencode` + GPT-6-Astra	+0.0569	[-0.0727, 0.2129]	1.000
`opencode` + DeepSeek-V4-Pro	+0.1490	[0.0535, 0.2467]	0.875
`opencode` + Gemini-3.1-Pro	-0.0563	[-0.1866, 0.0383]	1.000
`opencode` + Kimi-K3	+0.1963	[0.0882, 0.3217]	0.500
`codex` + GPT-6-Astra	-0.0701	[-0.2065, 0.0440]	1.000
`deepseek-harness` + DeepSeek-V4-Pro	+0.1310	[-0.1608, 0.4518]	1.000
`gemini` + Gemini-3.1-Pro	+0.0888	[-0.1103, 0.3339]	1.000
`kimi` + Kimi-K3	-0.1252	[-0.5651, 0.2291]	1.000

Table 13: **Oracle Quality by answer coverage.** Mean ± SD over three repeats.

Case	0%	25%	50%	75%	100%
09 Nuclear-site scheduling	0.0000 ± 0.0000	0.0000 ± 0.0000	0.3272 ± 0.5667	0.3206 ± 0.5553	0.9770 ± 0.0109
11 Retail-zone routing	0.0000 ± 0.0000	0.4661 ± 0.4037	0.7442 ± 0.0629	0.7935 ± 0.0670	0.8558 ± 0.0582
17 Cold-rolling sequencing	0.2347 ± 0.1189	0.1637 ± 0.0399	0.3357 ± 0.1595	0.4260 ± 0.1418	0.6782 ± 0.2787
38 Display-fab scheduling	0.0000 ± 0.0000	0.0000 ± 0.0000	0.0000 ± 0.0000	0.3190 ± 0.5525	0.9650 ± 0.0093
Four-case mean	0.0587	0.1575	0.3518	0.4648	0.8690

A.7 Zero scores and a case study

Of the 2,646 main runs, 548 score zero. Of these, 509 (92.9%) are invalid and 39 (7.1%) are valid but earn no Quality; all have a complete artifact and pipeline record. Table 14 groups the recorded `error_info` into mutually exclusive signatures (precedence: constraint, fatal, schema, execution). Constraint or feasibility violations dominate (430, 78.5%), and 61 runs (11.1%) carry no explicit message.

Table 14: **Evaluator signatures of 548 zero-score runs.**

Recorded signature	Runs	Share	Interpretation
Constraint/feasibility violation	430	78.5%	Case constraints or feasibility checks are explicitly violated.
Fatal case-specific check	29	5.3%	A case-defined fatal condition is recorded, such as incomplete coverage or an out-of-horizon event.
Schema/output validation error	8	1.5%	The submitted representation fails a required schema or output check.
Evaluator/scoring execution exception	4	0.7%	The evaluator records an exception or cannot form the required score.
Quality/threshold warning	16	2.9%	A quality or threshold warning accompanies zero Quality for a valid artifact.
No explicit evaluator message	61	11.1%	The artifact is scored zero but the retained record does not identify a more specific reason.

Zero scores belong to cases more than to systems. They occur in 33 of the 49 cases, and 7 cases account for half of them. The five cases with the most zeros (21, 10, 13, 18, and 38) contribute 233 (42.5%); in each, 17 or 18 of the 18 systems score zero at least once, and in case 21, 17 systems score zero in all three repeats. Within a system and case, clarification separates failing from passing repeats only slightly: in the 202 cells with both zero and nonzero repeats, zero-score repeats have Recall 58.6% against 63.6% (lower in 89 cells, higher in 72; Wilcoxon $p = 0.021$) and ask as many questions (22.8 vs. 23.2). Seventy-two zeros (13.1%) reached the runtime limit but still produced a scored artifact; 54 of them also record a constraint violation, so a timeout does not by itself explain the zero.

Case study: case 10. We trace one run of case 10 (retail replenishment; 17 days, 73 stores, 3,055 items; `opencode` + GPT-6-Astra, `Interact-Req`, first repeat); Table 15(a) gives the evidence chain.

Table 15: **Case-10 run.**

(a) Evidence chain

Stage	Recorded evidence
Customer request	JSON replenishment plan; internal store/warehouse transfers must be exhausted before supplier purchase, and same-city stock cannot be skipped; stockout avoidance takes priority over timeliness and cost.
Agent questions	27 messages (47 scored question units) over three rounds. The agent explicitly asked about source precedence ("same-city stores and local warehouse → cross-city sources → supplier") and about how shortages and lead times should be computed.
Key rule and coverage	The complete business information specifies receive arrivals → settle sales → dispatch transfers and purchases. The registered point "daily inventory transition order" requires this exact order. The stored majority label is `covered=false`, with 0.60 judge agreement (2 covered and 3 uncovered votes). No matched question was recorded for this blocker.
Submitted plan	The plan metadata nevertheless declares the same three-step order. It contains 15,987 instructions (15,515 transfers and 472 purchases), and the agent's validator reports `pass`.
Executable evaluation	All five extractor/evaluator routes mark the plan invalid and return Quality $= 0.0$. The evaluator records 347 constraint violations, including eight same-city exhaustion violations for `ITEM_000092`.
Reference	The customer-accepted reference has value 0.9974756. For the eight affected stores, it contains no item-092 supplier purchase and routes 10.58 units through internal transfers (seven stores receive a positive quantity), in contrast to the submitted plan.

(b) Item-`092` quantities for the eight affected stores

Affected stores	Agent internal	Agent supplier	Reference internal
Eight stores (aggregate)	9.81	3.87	10.58

For example, at one affected store on 2025-03-13, the submitted plan purchased item `ITEM_000092` from `SUPPLIER_DEFAULT` despite a local-store balance of 0.0 and 327.0 units available in `WH_LOCAL`; the same hard gate violation occurs across the eight stores, summarized in Table 15(b).

The run thus shows a rule that was asked about but not enforced.

A.8 AUDITS

Audit sample and tasks. Two experts, Annotator A and Annotator B, independently labeled 120 clarification items (105 runs, 38 cases) and 120 extraction runs (43 cases), each sample covering all 18 configurations. Samples were drawn with fixed seeds, stratified by configuration and by coverage or recorded issue. A clarification item asks whether answering at least one recorded question would reveal the reference point (covered, uncovered, or uncertain). An extraction run compares the submitted artifact with its normalized form (`CORRECT`, `INCORRECT`, or `NOT_ASSESSABLE`); runs without a submitted result file are `NOT_ASSESSABLE`.

Post-hoc audit of simulated-customer answers. Annotators A and B labeled the sampled customer answers for support by the case information and consistency with earlier turns (Table 16(a)). The audit indicates that the simulator's main weakness is cross-turn inconsistency rather than direct conflict with the case.

Table 16: **Audit results.**

(a) Simulated-customer answers; counts are supported/uncertain/contradicted for support and factuality rows and not applicable/consistent/contradicted for status and consistency rows

Criterion	n	A	B	Agreement	κ
Answer support (all labels)	1303	1250/51/2	1179/117/7	92.10%	0.385
Answer support (non-uncertain)	1170	—	—	99.57%	0.285
Cross-turn status	1303	968/320/15	965/323/15	99.00%	0.974
Run factuality (all labels)	105	79/24/2	71/28/6	82.86%	0.601
Run factuality (non-uncertain)	73	—	—	91.78%	0.218
Run consistency	105	60/36/9	60/35/10	97.14%	0.948

(b) Agreement metrics between annotators and the automated five-judge panel; intervals are case-cluster bootstrap 95% intervals for κ

Comparison	n	Agreement	κ	95% CI
Clarification: A–B	114	96.49%	0.9297	[0.8571, 0.9832]
Clarification: A–panel	114	94.74%	0.8947	[0.8054, 0.9661]
Clarification: B–panel	120	93.33%	0.8667	[0.7820, 0.9424]
Extraction fidelity: A–B	120	91.67%	0.7656	[0.5835, 0.9033]

Agreement and uncertainty. Table 16(b) reports agreement and unweighted Cohen's κ with 95% case-cluster bootstrap intervals (5,000 resamples). Six clarification items that Annotator A marked uncertain are excluded from comparisons involving A; extraction agreement retains all three categories.

The panel does not systematically inflate Recall: its disagreements with Annotator A are balanced between covered and uncovered labels.

Extraction agreement and failures. We audited the retained `result.json` and extractor records of all 2,646 main runs. Numeric votes have strong majority agreement, while the remaining non-numeric failures are rare and are broken down by configuration in Table 17. Because the 120-run audit sample was enriched for recorded extraction issues, we reweighted it to the population; the resulting estimated extraction accuracy is 97.9%.

Overall extraction failure rates are low across configurations, and majority voting reduces the effect of individual extractor disagreements on final scores.

Table 17: **Extraction failures by configuration.**

Agent / model	Failures / slots	Rate	No artifact
opencode			
GPT-6-Astra	3/735	0.41%	0
GLM-5.3	9/721	1.25%	4
DeepSeek-V4-Pro	5/731	0.68%	2
Gemini-3.1-Pro	6/733	0.82%	0
Grok-4.6	2/733	0.27%	0
Kimi-K3	11/731	1.50%	4
Qwen3.8-Max	10/735	1.36%	0
openhands			
GPT-6-Astra	21/729	2.88%	0
GLM-5.3	9/711	1.27%	1
DeepSeek-V4-Pro	3/735	0.41%	0
Gemini-3.1-Pro	3/734	0.41%	1
Grok-4.6	6/733	0.82%	0
Kimi-K3	2/730	0.27%	3
Qwen3.8-Max	9/719	1.25%	3
`codex` + GPT-6-Astra	1/448	0.22%	1
`deepseek-harness` + DeepSeek-V4-Pro	2/695	0.29%	0
`gemini` + Gemini-3.1-Pro	3/734	0.41%	0
`kimi` + Kimi-K3	5/733	0.68%	2
All configurations	110/12,820	0.86%	21

A.9 Prompt templates and interaction interfaces

This section reproduces the benchmark prompts for the solving agent, simulated customer, extractor, and judge, and the question–answer interfaces. They form the benchmark layer added to each scaffold; the scaffolds' own prompts are unchanged. Double-braced fields are filled at runtime. The listings below show the exact shared and role-specific templates used in the benchmark, including the information boundaries and the no-repair extraction contract. Scaffold-specific wrappers and exact field substitutions are released with the artifact in `analysis/prompt_templates/`.

A.9.1 Solving-agent instructions

All conditions use the same base template. The request block is the heading `## The client's request` followed by the complete `instruction.md`; the data list contains the sorted filenames directly under `data/`. The workspace is the run's solving directory. Only the condition block changes.

Shared agent template

```
You are an algorithm delivery engineer. A client has sent you a business
    request and some data files.

Your job: understand what they actually need, then build a working
    solution and deliver it.

## Working directory

{{workspace}}

Data files provided by the client:
{{data_files}}

Write all outputs to the working directory.
```

Shared agent template (continued)

```
{{request_block}}

{{condition_block}}

## Delivering

Do the work -- do not merely describe how you would do it. The task is
    complete only
when the deliverable files exist in the working directory.

Produce whatever artifact the request calls for. The client did not
    specify a file
format -- infer what is appropriate from what they asked for.

Also write a short `NOTES.md` recording your key modelling decisions:
    what you
assumed, what you inferred from the data, and anything you were unsure
    about.
```

Hidden condition block

```
## Missing information

The request is written in the client's own words. It will not spell out
    every
detail you need -- data conventions, units, edge-case handling and
    similar points
are often left implicit because the client considers them obvious.

There is no one available to answer questions. Where something cannot be
determined from the request or the data, make a reasonable assumption,
    proceed,
and record the assumption in `NOTES.md`.
```

Interact condition block

```
## Missing information

The request is written in the client's own words. It will not spell out
    every
detail you need -- data conventions, units, edge-case handling and
    similar points
are often left implicit because the client considers them obvious.

The client is reachable and expects to be consulted on requirements. If
    something
cannot be determined from the request or the data, ask them. You may ask
    more
than once, and at any point in the work.

When you ask, put the questions at the end of your reply under a heading
`## Questions for the client`, one question per line, and stop there --
    the client
will answer in the next message. (Asking in plain prose also works; the
    heading
```

Interact condition block (continued)

```
just makes the exchange cleaner.)

For anything you choose not to ask about, make a reasonable assumption,
    proceed,
and record the assumption in `NOTES.md`.
```

Interact-Req condition block

```
## Missing information

The request is written in the client's own words. It will not spell out
    every
detail you need -- data conventions, units, edge-case handling and
    similar points
are often left implicit because the client considers them obvious.

The client is reachable. Before writing any code, ask the client about
    anything
you need settled, and keep asking until you have what you need. Do not
    start
the work until then.

Ask about anything the request and the data do not settle. You may ask
    across
several rounds.

When you ask, put the questions at the end of your reply under a heading
`## Questions for the client`, one question per line, and stop there --
    the client
will answer in the next message. (Asking in plain prose also works; the
    heading
just makes the exchange cleaner.)

Proceeding on an unconfirmed reading counts as a failure even if the
    guess
turns out right. Record in `NOTES.md` any assumption the client did not
    settle.
```

Full condition block

```
## Additional context

The client has already supplied the following details, covering the data
conventions, units and edge-case handling that the request itself leaves
    implicit:

{{information}}

There is no one available to answer further questions. Where something
    is still
undetermined, make a reasonable assumption, proceed, and record it in
    `NOTES.md`.
```

Oracle answer-coverage condition block

```
## Missing information

The request is written in the client's own words. It will not spell out
    every
detail you need -- data conventions, units, edge-case handling and
    similar points
are often left implicit because the client considers them obvious.

The client has already settled the following points for you:

{{answers}}

There is no one available to answer further questions. Where something
    still
cannot be determined from the request, the data, or the points above,
    make a
reasonable assumption, proceed, and record the assumption in `NOTES.md`.
```

For `Full`, `information` is the complete business-information file. For the oracle experiment, `answers` is the selected answer text, rendered as bullets in registry order, or `(none)` for zero coverage. The nested subset selection is specified in Appendix A.6; requirement identifiers and type labels are not inserted into the oracle prompt.

A.9.2 SIMULATED-CUSTOMER MESSAGES

The customer uses GLM-5.3 and the default answering style below. Each call has this system message, the accumulated user-question/assistant-answer history, and the current question as a new user message. The returned answer is appended to that history. The two knowledge fields are the initial request and complete business information; clarification annotations and evaluator code are not included. The retained implementation requests at most 10,000 output tokens per call and leaves temperature and seed unset.

Customer system prompt (English translation)

```
You are the person responsible for this business scenario, answering the
    technical team's clarification questions about the requirements.
The following is your business knowledge. Before answering, read each
    block in full and cross-check them. In particular, <information>
    often contains specific values, field names, coordinates,
    thresholds, and constraint details. When asked about a schema,
    parameter, threshold, coordinate, formula, or similar detail, first
    look for the existing answer in <information>; do not say "I will
    provide it" or decide it on the spot:

<instruction>
{{instruction}}
</instruction>

<information>
{{information}}
</information>

Note: Any of the above may contain internal technical details such as
    evaluation formulas, scoring weights, or penalty coefficients. Do
    not volunteer these to the technical team. If asked, you may provide
```

Customer system prompt (English translation) (continued)

```
    a qualitative business-level explanation, but do not give specific
    mathematical formulas or numerical weights.

Mandatory answering procedure (follow these three steps for every
    question):

Step 1: Check information first (internally; do not include this step in
    the reply).
Before selecting or writing an answer, find the corresponding text in
    <information> and answer on that basis.
You are playing the business owner: this is business knowledge in your
    head, not a document you can consult.
Do not say "checking information", "according to the
    document/materials/instructions", "original excerpt", or "not
    found", or imply that you are consulting materials. Answer directly
    in your own voice.

Step 2: Compare the options with the facts in information.
- If an option exactly matches the facts in information, select it.
- If none of the options exactly matches, select "Other (please
    describe)" and copy the ranges, values, or field names from
    information.
- Only if information does not address the question may you decide based
    on instruction.

Step 3: Do not replace specific values with vague qualifications.
Do not answer "unrestricted / not counted / unnecessary / flexible / for
    the model to decide" unless information explicitly says so. Copy
    specific values, ranges, and field names from information.

If selecting an option that says "please describe", provide the specific
    content after the option name. If there is nothing new to add,
    select a negative option rather than leaving the description empty.

Answer requirements:
- Speak as the business owner. Give only the answer itself, without
    explaining the reasoning or repeating the question.
- If there are multiple questions, answer each directly on a separate
    line.
- Do not volunteer implementation details such as evaluation methods,
    scoring formulas, or exact objective weights. If asked, you may give
    a qualitative business-level explanation, but not specific
    mathematical formulas or numerical weights.
- If none of the information sources covers the question, provide a
    reasonable business default.
```

This prompt allows business defaults for unsupported questions, so answers are not guaranteed to follow from the case information (Appendix A.8).

A.9.3 Question and answer interfaces

The harness supports native question requests, a scaffold callback, and text exchanges. These routes share the customer above but preserve each scaffold's interaction mechanism. The following examples illustrate the message shapes; angle-bracket values are placeholders, not recorded dialogue.

OpenCode native question tool. The harness receives a request containing a `questions` array and replies through `POST /question/<request-id>/reply` (or the corresponding session-scoped route in the older API). Each answer is a one-element string array, preserving free-text answers

rather than forcing them into an agent-proposed option. The customer receives each `question` string, falling back to `header` if it is empty; option metadata is recorded but is not separately appended to that string.

Native question request and reply

```
Request:
{"id": "<request-id>", "questions": [
  {"question": "<question text>", "header": "<short header>",
   "options": [{"label": "<option>", "description": "<meaning>"}]}
]}

Reply:
{"answers": [["<customer answer>"]]}
```

Text exchange and OpenHands callback. The CLI configurations use text exchanges. OpenHands obtains a reply via its user-response callback, connected to the customer by the harness. Questions under the suggested heading are parsed directly when possible; otherwise a detector determines whether the message requests information. Each detected question is answered separately, and the answers return as one user message in the form below. Recorded question units used for scoring may be split further as described in Appendix A.3.

Text-channel request and response

```
Agent message:
## Questions for the client
<question 1>
<question 2>

Customer reply delivered as a user message:
- <question 1>
  <answer 1>
- <question 2>
  <answer 2>
```

The detector sees only agent text, not hidden requirements, and is skipped when the question heading is present. Replies without a question mark are treated as non-questions; otherwise five models vote by strict majority of returned ballots on whether the agent is asking. Overlong replies keep the first 2,000 and last 8,000 characters.

Question detector system message

```
You judge whether a coding agent's reply is ASKING the client for
    information, or REPORTING/DELIVERING work.

Return JSON only:
{"asking": true|false, "questions": ["...", "..."]}

Rules:
- "asking" is true only if the agent is waiting for the client to supply
  information it cannot determine itself.
- Rhetorical questions inside an explanation or report are NOT asking.
  ("How do we balance time against fairness? We weight them 1:90." ->
    false)
```

Question detector system message (continued)

```
- Offers to continue are NOT asking. ("Shall I also plot the routes?" ->
    false)
- Split compound questions into atomic ones, one fact per entry.
  ("What speed, and is loading time included?" -> two entries)
- Keep each question in its original language, close to the agent's
    wording.
- If "asking" is false, "questions" must be [].
```

Question detector user message

```
Agent reply:

{{clipped_agent_reply}}{{hint}}
```

The detector requests at most 2,000 output tokens. On malformed JSON it retries once, prepending the following text to the original user message:

Question detector JSON-repair retry prefix

```
Your previous response violated the JSON protocol (ValueError). Return
    exactly one JSON object with keys asking and questions, and no other
    text.
```

The retained text loop sends the following continuation when an agent has stopped without a deliverable, subject to its run budget, and allows up to three such pushes in that branch. Repeated identical question batches or exhausted clarification limits also trigger a transition to solving.

Continuation and budget messages

```
Continuation:
Understood. Please now carry out the work and write the deliverable
    files into the working directory.

Text-channel clarification budget exhausted:
(No further questions can be answered. Proceed with your best judgement.)

Native-tool / callback clarification budget exhausted:
No further information is available. Proceed with your best judgement.
```

If a customer call fails, the OpenCode handler returns a placeholder telling the agent to proceed on its own judgment.

A.9.4 ARTIFACT-EXTRACTION INSTRUCTIONS

The extractor runs after the solving agent has finished. The shared prompt below inserts the case's complete `submission_schema.json` under `Target schema (authoritative)`. If that file is absent, it uses the evaluator's `EXTRACTOR_SPEC` documentation, or directs the extractor to infer the schema from evaluator source. Paths identify the retained agent workspace, case data, evaluator, and a separate normalized-output directory.

Shared extractor template

```
You are a data extractor for a benchmark evaluation pipeline.

## Your task

1. **Read `_agent_summary.md` first** (if present in the workspace). It
    tells you which file
   the agent considers its final answer. The agent's own designation
    decides which artifact
   gets graded -- you do not get to pick a different (better-scoring)
    one.
2. Read the target schema below to learn what files and fields the
    evaluator accepts.
3. Find the agent's output in the workspace. Filenames and column names
    will differ.
4. Normalize it to the schema and write the result to the OUTPUT
    directory.
5. **If the deliverable is executable code** (the schema asks for a
    `.py` defining a specific
   function or class): the evaluator imports it from the OUTPUT
    directory, so copy the entry
   file under the required name AND every workspace-local module it
    imports, directly or
   transitively. Copying only the entry file leaves it orphaned --
    ModuleNotFoundError, and a
   working solution scores 0.
6. Write `_result.json` to the OUTPUT directory as your final action.

{{schema_block}}

{{verification_block}}

**Stop as soon as the evaluator returns a score.** A 0 is a legitimate
    final answer: the
agent's solution ran and did not meet the constraints. Never edit
    values, add rows, drop
rows, or swap in a different solution file to raise the number. You
    normalize
*representation* -- filenames, column names, layout, encoding. The
    decisions are the agent's
and must survive unchanged. If no score can be obtained, the artifact is
    wrong; say so and
stop rather than making something up.

**Forbidden transformations.** The following transformations would
    change the
agent's substantive answer and are therefore forbidden.

1. **Nudging values inside a stated tolerance.** Do not adjust a number
    within an evaluator
   tolerance (or any other slack) so that a failing constraint passes.
    Tolerance exists to
   absorb *the agent's* rounding, not to give the extractor a search
    space. If the agent's
   numbers land outside a limit, that is the answer: report the 0.
2. **Writing an adapter when the agent's interface does not match the
    evaluator's.** When a
   deliverable is code and its signature or data model disagrees with
    what the evaluator
```

Shared extractor template (continued)

```
   passes in (e.g. the policy expects a custom object, the evaluator
    hands it a dict), do
   **not** author a shim, a wrapper class, or a re-implementation. Copy
    the agent's file
   as-is, let it fail, and record the mismatch in `notes`.
3. **Dropping rows that "look wrong".** Removing rows the agent emitted
    -- even ones that
   seem like annotations, placeholders, or infeasible leftovers --
    changes the answer. If you
   drop or add **any** row, state the before/after counts and the reason
    in `notes`.

4. **"Fixing" an identifier the agent spelled wrong.** Whether you may
    repair a bad name
   depends on one question: **does the task specify the naming?**
   - **Task is silent on naming** (e.g. the agent labels a plate
    `plate1` where the evaluator
     keys on `P001`): map it, provided the target is **unique** in the
    reference data. This
     is ordinary identifier normalization. If the mapping is ambiguous
    or has no match,
     do not guess -- leave it and say so.
   - **Task states the naming rule** (e.g. `information.md` says feature
    names must be
     real columns of the training data, in `{trait}_D{day}` form): a
    name that breaks the
     rule is a **substantive error by the agent**, not a formatting
    slip. Copy it as-is and
   let it score 0.
   Check `information.md` / `instruction.md` for a naming rule
    **before** you repair any
   name. Also check the schema's `role`: repairing is only ever in scope
    for `identifier`
   fields, never for `decision` fields -- those are the answer itself.

5. **Using the evaluator's own formula to generate a value the evaluator
    checks.** When the
   schema documents how a field relates to others (`sub_i_long must equal
   mother_cut_weight / (mother_width x 7.65) x 10000`), that formula is
    there so you can
   understand the field -- **not so you can compute it**. Filling the
    field from the formula
   makes the consistency check pass by construction: the evaluator ends
    up verifying its own
   arithmetic instead of the agent's. A derived value that the evaluator
    validates is
   load-bearing even when it is not itself a decision variable.
   The test is not "is this field a decision?" -- it is **"does the
    evaluator look at it?"**
   If yes, the value must come from the agent, even when you can compute
    a better one, and
   even when the agent's is provably inconsistent. An inconsistency you
    found is a finding to
   report in `notes`, not a defect to repair.

## When the agent's designated file is incomplete

Rule 1 says the agent's own designation decides what gets graded. That
    assumes the
```

Shared extractor template (continued)

```
designated artifact exists and is coherent. When it is **not**:

- **The designation names a state the agent never finished.** If the
    designated artifact is
  incomplete or internally contradictory, fall back to the most recent
    complete state that
  passes the agent's own validation, set `status` to `partial`, and say
    in `notes` which state
  you used and why the designated one was left out. This is not "picking
    the better-scoring
  file" -- it is selecting the only file that is a coherent answer.
- **Extra rows beyond the graded set.** Rows the agent produced for keys
    that are not in the
  evaluation set should be dropped, with the before/after counts
    disclosed. That is different
  from dropping rows the agent *did* produce for graded keys, which is
    forbidden.

## Verify order-bearing fields before you write

When a field encodes a sequence (a visit order, a permutation, or a
    concatenated route), its
order **is** the decision. Reconstructing it from a grouped source can
    silently scramble the
group order even when all members and counts match.

Before writing an order-bearing field, re-derive it from the source a
    second way and compare
element by element. If the schema gives you a companion structure (a
    `trips` array with per-
trip stop counts, say), slice your sequence by those counts and check
    each slice against the
source group it should match.

Rule of thumb: if an edit would change the score, it is not
    normalization. Reformatting,
renaming, reshaping, re-encoding never change a score -- those are yours
    to make freely.

## Normalization rules

- **Semantic-first mapping**: the same column name does not guarantee
    the same meaning.
  Check the schema's field definition (units, scope, total vs. net)
    before renaming.
- **Column renaming / format conversion**: fuzzy-match agent columns to
    schema fields;
  convert wide/long layouts as needed.
- **Missing rows**: only fill a missing combination with zero when the
    schema defines that
  dimension's absence AS zero. Do NOT invent rows for tasks the agent
    did not produce --
  leave them out and set status to "partial".
- **Never** re-run the agent's solver, re-optimize, or re-compute its
    decisions.
- **Prefer the highest-precision artifact.** When the agent expresses
    the same decisions in
  several files -- say a minute-truncated CSV and a full-precision JSON
    -- take the lossless
```

Shared extractor template (continued)

```
  one, even if `_agent_summary.md` names the other. Precision loss is
    not a formatting
  choice; it silently changes the answer being graded.
- **Never take anything from the Data directory as the agent's output.**
    `data/` holds the
  problem input, including any baseline or seed policy shipped with the
    task. Submitting one
  of those grades the benchmark's own reference instead of the agent --
    if the workspace has
  no real deliverable, say so and set status to "failed".

## Paths & permissions

| Directory | Path | Read | Write |
|-----------|------|------|-------|
| Workspace | `{{workspace}}` | YES | **NO** |
| Data      | `{{data_dir}}` | YES | **NO** |
| Evaluator | `{{evaluator}}` | YES | **NO** |
| Output    | `{{output_dir}}` | YES | **YES** |

All writes go to the Output directory only.

## _result.json format

{
  "status": "success",
  "notes": "what was found and what transformations were applied",
  "files_found": ["relative paths of agent output files discovered"],
  "files_normalized": ["filenames written to the output directory
    (excluding _result.json)"]
}

Status: "success" = all required files written; "partial" = found output
    but it is
incomplete or self-contradictory (describe what is missing); "failed" =
    no relevant agent
output found. When the agent's output is genuinely incomplete, "partial"
    is the correct
outcome -- do NOT reconstruct or re-solve to reach "success".
```

The `verification_block` uses the first block below for ordinary cases. Case 13, whose evaluator launches expensive solver subprocesses, uses the second block and is scored after the extractor exits.

Standard extractor verification block

```
## Verify by running the evaluator (required)

A schema cannot enumerate every shape an agent might produce, so do not
    guess whether your
normalization is right -- check it:

```bash
python3 {{evaluator}} --data-dir {{data_dir}} --submission-dir
    {{output_dir}}
```
```

Standard extractor verification block (continued)

```
React to *structural* failures only:

| What you see | What it means | What to do |
|---|---|---|
| `ModuleNotFoundError` | a module the entry file imports is missing |
    copy that module in too |
| `AttributeError: must define <fn>` | wrong entry file | try the next
    candidate |
| `missing <file>` / `KeyError` / missing column | filename or column
    mapping is off | fix the mapping |
| valid JSON with any score, **including 0** | the evaluator ran to
    completion | **stop -- you are done** |
```

Executable-case verification exception (case 13)

```
## Heavyweight executable case

Do not run `{{evaluator}}` or the Agent solver. Copy the selected Agent
    Python
source byte-for-byte to the required output filename. Do not rewrite its
    CLI,
output shape, algorithm, or decisions: incompatibility is an Agent
    result.
The scoring stage evaluates it after you exit.
```

The extractor's `status` describes the extraction only; the evaluator alone scores the artifact (Appendix A.3).

A.9.5 CLARIFICATION-COVERAGE JUDGE MESSAGES

Each judge receives the same system message and, for each registered requirement, the user template below. The numbered question list is the complete run-level list after splitting. Calls use temperature zero.

Coverage judge system message

```
You grade whether an agent's clarification questions uncovered a
    specific hidden requirement. Judge strictly and reply with JSON only.
```

Coverage judge user message

```
A client holds the hidden requirement below. An agent asked the questions
below to clarify the task. Decide what the client would reveal in reply.

HIDDEN REQUIREMENT (id: {{requirement_id}})
{{requirement_description}}
{{optional_trigger_block}}

AGENT'S QUESTIONS (each on its own line, numbered)
{{numbered_questions}}
```

Coverage judge user message (continued)

```
The client answers cooperatively and concretely: when a question points
    at a
topic they hold information about, they state the actual values or rules
    rather
than a generic description. They do NOT volunteer information the agent
    never
asked about.

Judge two things:

1. ATTEMPTED -- does any question point at this requirement's topic,
    however
   loosely phrased?
2. COVERED -- answering that question, would the client state the
    specific
   requirement above?
   - Yes if the question names the specific quantity, rule, field or
    entity this
     requirement is about, so the client knows exactly which convention
    to state.
     The agent need not already know the value, nor use the same
    wording; asking
     what the convention is counts.
   - No if the question merely names a topic, module or data source and
    asks what
     to "watch out for", what the "rules" are, or what is "special" --
    even when
     that topic is the right one. Such a question does not tell the
    client which
     of the many conventions in that area is wanted, so a cooperative
    client
     would ask back rather than recite this requirement.
     Examples that must be judged No: "anything special about the data?",
     "any rules I should know about billing?", "any pitfalls when
    reading these
     sheets?". Naming the right domain does not make a question specific.

Reply with JSON only:
{"attempted": true|false, "covered": true|false,
  "question_indices": [<1-based indices that cover it>]}
```

Optional trigger-example block

```
QUESTIONS THAT WOULD SURFACE IT (reference examples, not exhaustive --
a differently-worded question that targets the same fact also counts)
{{indented_trigger_bullets}}
```

Only `covered` enters Recall, and the returned indices determine Precision. A malformed reply is retried once with the prefix below.

Coverage judge JSON-repair retry

```
Your previous response violated the JSON protocol (ValueError). Return
    exactly one JSON object matching the required schema. Do not include
    analysis, Markdown, or any text before or after it.

{{original_judge_user_message}}
```