ORWORKERBENCH: BENCHMARKING LLMS FROM DATA TO DECISIONS UNDER UNCERTAINTY

Anonymous authors
Paper under double-blind review

ABSTRACT

Operations Research increasingly provides the decision layer for AI systems that must allocate resources, schedule work, manage inventory, or act under uncertainty. Recent optimization benchmarks show strong performance from frontier LLMs and agents, but many begin from a *specified optimization problem*, where the stochastic quantities required by the model are already provided. In real operational settings, those quantities often have to be inferred from finite observations before optimization can begin. We introduce ORWORKERBENCH, a benchmark for evaluating this observation-to-decision setting. Each underlying decision problem is evaluated under matched conditions: in *NL2Code*, the relevant stochastic parameters are given explicitly, while in *Workflow*, the agent must infer them from 100 observations before constructing and executing the optimization procedure. Our benchmark contains 147 NL2Code tasks and 441 Workflow tasks across 21 human-verified executable protocols. We additionally use neuron-guided synthesis to reduce behavioral redundancy during benchmark construction; compared with embedding-based selection, it produces more diverse activation patterns across four held-out models. Our evaluation of six proprietary and two open-weight LLMs reveals a substantial observation-to-decision gap. The strongest models achieve over 99% tolerance accuracy when the stochastic quantities are supplied, yet incur roughly 7% mean capped error when those quantities must instead be recovered from observations. Moreover, even the strongest systems perform close to a simple same-data reference that estimates the required stochastic quantities from the same observations and passes them to a verified solver. These results show that strong performance on specified optimization problems does not yet translate into reliable decision making from finite observations.

1 INTRODUCTION

Operations Research (OR) provides the decision layer for systems that schedule work, route vehicles, manage inventory, plan capacity, and allocate resources. Recent work has shown that LLMs can translate natural-language requirements into optimization models and executable solver code (AhmadiTeshnizi et al., 2024; Huang et al., 2025a). Real operational tasks, however, often do not begin with all stochastic quantities specified. Instead, an agent must infer decision-relevant information from finite observations before optimization can begin. For example, an inventory agent may receive historical demand records and stocking costs rather than a known demand distribution. It must identify which characteristics of demand affect the decision, estimate them from the observations, formulate the optimization problem, and compute the resulting action. Reliable performance therefore depends on the entire observation-to-decision workflow, rather than on optimization alone.

Existing evaluations span natural-language optimization modeling, broader optimization paradigms, executable agent workflows, and data-driven optimization from historical samples (Ramamonjison et al., 2022; Zhang et al., 2026; Li et al., 2026; Zhu et al., 2025). One remaining gap is a controlled evaluation of the capability gap between solving a fully specified optimization problem and solving the same problem when its decision-relevant stochastic quantities must first be inferred from finite observations. Without such a matched comparison, aggregate benchmark performance cannot reveal how much error arises from interpreting and estimating the uncertain inputs, rather than from formulating and solving the optimization problem itself.

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

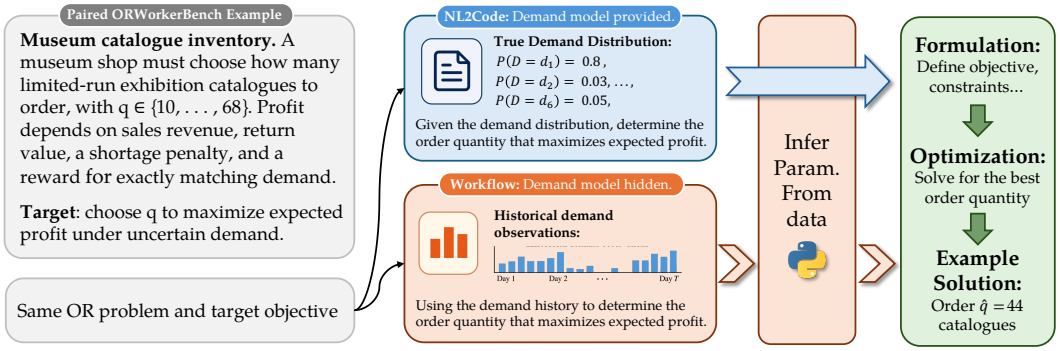


Figure 1: A concrete paired example from ORWorkerBench. The underlying optimization problem and target objective are fixed across the two conditions. In NL2CODE, the stochastic demand model is provided directly; in WORKFLOW, the agent instead observes historical demand observations and must infer the decision-relevant uncertainty before solving the same optimization problem.

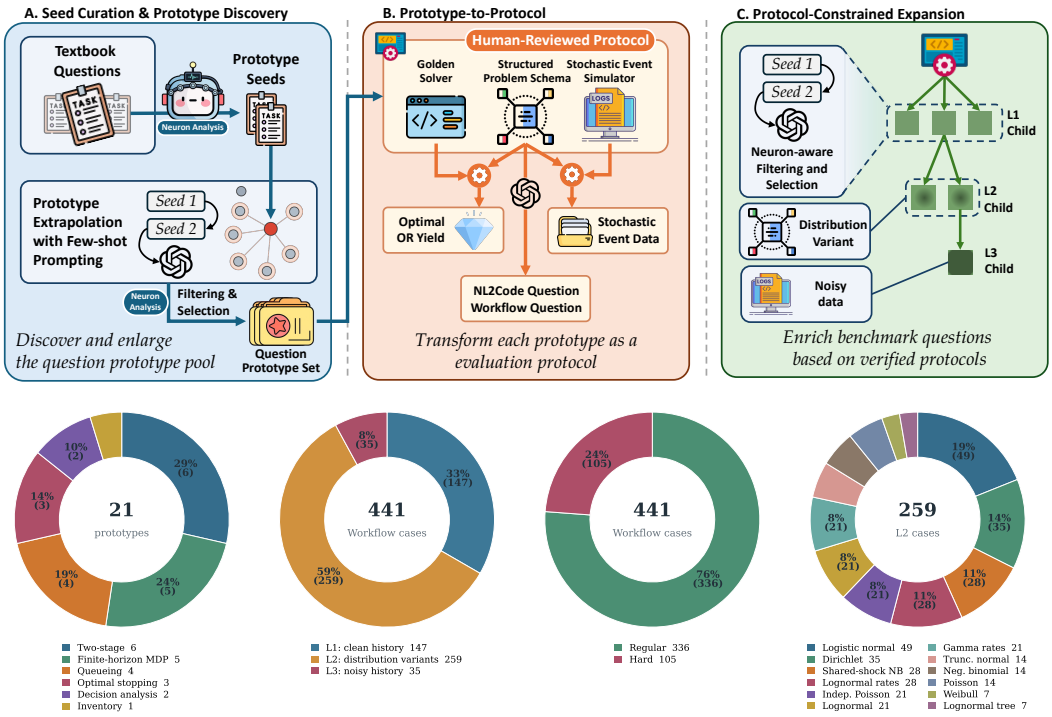
Evaluating this capability gap requires evaluating the same underlying decision problem once with the relevant stochastic quantities provided directly and once with those quantities inferred from finite observations. To understand where performance is lost, the benchmark should also retain executable ground truth and provide a reproducible reference baseline that uses exactly the observations available to the agent.

To this end, we introduce ORWORKERBENCH, in which each decision problem is evaluated under two settings. First, in *NL2Code*, the parameter-specified setting, the stochastic parameters are stated explicitly. Second, in *Workflow*, the observation-driven setting, the agent receives a finite event history and must infer the quantities needed for optimization. Both questions are generated from the same underlying OR question and are scored against the same optimum, so they differ only in whether the parameters are given or must be inferred. Each Workflow question is also compared with an empirical baseline for measuring excess error. Overall, the benchmark includes 147 NL2Code and 441 Workflow tasks across 21 human-verified protocols and spans diverse hidden distributions and observation-noise levels. Figure 1 shows a paired example from the benchmark.

Constructing such paired cases requires several elements: a valid natural language question containing decision context, matching stochastic observations, and verified reference solutions. We organize construction around problem **prototypes**, each formalized as a reusable **protocol** consisting of a schema, simulator, and verified solver. Once prototypes are validated by human reviewers, each protocol generates paired instances and variants. To address the limited diversity of LLM-generated data (Jung et al., 2025), we adopt a neuron-guided generation and selection pipeline that uses activation patterns from solution traces to guide prototype discovery and instance expansion (Figure 2). Our ablation study shows greater activation-mode diversity than traditional embedding-based selection across four held-out models unused in the synthesis process, suggesting that the resulting questions engage a broader range of model knowledge.

Our design choices of the benchmark allow us to study three research questions: **RQ1:** To what extent does LLM performance on specified optimization problems carry over when the stochastic quantities required for optimization must be inferred from finite observations? **RQ2:** How much of the error in observation-driven optimization can be explained by finite-sample estimation, and how much additional error is introduced by the agent while interpreting the data, constructing the model, and executing the solution procedure? **RQ3:** Do stronger models and specialized optimization tools substantially reduce the observation-to-decision gap?

The experiments reveal a substantial gap between the parameter-specified and observation-driven settings. The strongest models nearly saturate the parameter-specified tasks, including the Hard split, yet performance drops sharply when the stochastic quantities required for the same decision problems must instead be recovered from observations. Moreover, most systems perform close to the naive empirical baseline reference, indicating that much of the remaining difficulty lies upstream of, or at the interface with, optimization: identifying and estimating the decision-relevant stochastic quantities and carrying them reliably into the downstream optimization procedure. Stronger models



(a) Problem Type **(b)** Variant Levels **(c)** Benchmark Difficulty **(d)** Distribution Types

Figure 2: **ORWorkerBench construction and composition.** Top: neuron-guided prototype selection (A), human protocol review (B), and expansion into L1–L3 instances (C). Bottom: 21 protocols from six OR families yield 147 NL2Code tasks and 441 Workflow cases.

narrow this gap, while specialized tool access does not consistently improve objective accuracy and introduces additional execution failures.

Overall, our main contributions are:

- **An observation-to-decision optimization benchmark.** We introduce a benchmark for evaluating optimization agents that must infer decision-relevant stochastic information from finite observations before solving the downstream problem.
- **Matched observation-to-decision evaluation.** Each problem is evaluated in paired NL2Code and Workflow settings, allowing us to directly measure how much performance drops when stochastic parameters must be inferred rather than provided.
- **Neuron-guided benchmark synthesis.** We use internal activation patterns to complement semantic and solution-trace diversity, helping reduce benchmark behavioral redundancy.
- **Empirical study of frontier LLMs.** Across eight modern LLMs, strong performance on specified optimization problems does not reliably carry over to observation-driven decision making.

2 Related Work

LLMs for Optimization Modeling. Recent work uses LLMs to translate natural-language requirements into optimization models and solver code through multi-agent collaboration (AhmadiTeshnizi et al., 2024; Xiao et al., 2024), fine-tuning (Jiang et al., 2025; Huang et al., 2025a), reinforcement learning (Chen et al., 2025; Zhao et al., 2026), and reusable optimization knowledge (Yang et al., 2026; Kong et al., 2026b). These methods are evaluated on benchmarks such as NL4Opt (Ramamonjison et al., 2022), MAMO (Huang et al., 2025b), IndustryOR (Huang et al., 2025a), OptiBench (Yang et al., 2025), and NLP4LP (AhmadiTeshnizi et al., 2025), which primarily measure modeling and solving from specified problem descriptions.

Benchmark	Problem Type		Uncertainty		Evaluation Paradigm		Data Source
	Determ.	Stoch.	Stated	Latent	Single-turn	Multi-turn	
NL4Opt (2022)	✓	✗	✗	✗	✓	✗	Human
MAMO (2025b)	✓	✗	✗	✗	✓	✗	Literature, LLM
OptiBench (2025)	✓	✗	✗	✗	✓	✗	Literature, LLM
IndustryOR (2025a)	✓	✗	✗	✗	✓	✗	Human
NLP4LP (2025)	✓	✗	✗	✗	✓	✗	Human, Literature
OptiVerse (2026)	✓	✓	✓	✗	✓	✗	Literature
FrontierOR (2026a)	✓	✗	✗	✗	✗	✓	Literature
OR-Space (2026)	✓	✗	✗	✗	✗	✓	Human, LLM
ORAgentBench (2026)	✓	✗	✗	✗	✗	✓	Human, Literature, LLM
DAOpt / OptU (2025)	✗	✓	✗	✓	✓	✗	LLM
ORWorkerBench (ours)	✗	✓	✓	✓	✓	✓	Literature, LLM

Table 1: Positioning of ORWorkerBench. Stated vs. latent indicates whether stochastic parameters are given or must be inferred from data. Ours is the only benchmark covering both conditions.

More recent benchmarks broaden this setting: OptiVerse covers a wider range of optimization paradigms (Zhang et al., 2026), ORAgentBench evaluates executable workflows over operational artifacts (Li et al., 2026), and DAOpt studies optimization from historical samples (Zhu et al., 2025). Our ORWorkerBench differs by providing matched parameter-specified and observation-driven versions of the same decision problem, together with a same-data reference baseline for measuring the observation-to-decision gap. Table 1 summarizes these distinctions.

Data Generation with Model-Internal Signals. Synthetic-data pipelines commonly encourage diversity through generation, filtering, evolution, and embedding-based selection (Wang et al., 2023; Xu et al., 2024; Liu et al., 2024; Jung et al., 2025). However, semantic diversity does not necessarily imply diversity in the model behaviors being exercised. Recent work therefore explores gradients and neuron activations as model-dependent signals for data selection (Xia et al., 2024; Chen et al., 2026b). We build on this idea by using neuron activations from solution traces to complement semantic diversity during benchmark synthesis.

3 ORWorkerBench: Design and Construction

ORWorkerBench measures the gap between solving a stochastic optimization problem when its parameters are given and solving the same problem when those parameters must first be inferred from observations. To support this comparison, the benchmark must satisfy three requirements: matched task conditions, executable ground truth, and sufficient diversity to avoid repeatedly testing the same problem behaviors. Figure 2 summarizes the construction pipeline, while Figure 1 illustrates the intuition behind the paired evaluation.

3.1 Task Formulation and Metrics

Each parent instance defines two matched task conditions. In **NL2Code**, the stochastic parameters required for optimization are stated explicitly. In **Workflow**, those parameters are hidden and the evaluated model instead receives a finite observation history from which it must infer the quantities needed for the same decision problem. Models used for benchmark generation, neuron-guided selection, and construction-time quality checking are distinct from those evaluated on the benchmark.

The clean Workflow condition, which we denote **L1**, shares the same latent instance and target objective as its NL2Code counterpart. We additionally construct two robustness variants. **L2** changes the hidden stochastic distribution while preserving the optimization structure, and **L3** introduces corrupted observations for the corresponding latent problem. Appendix B.3 provides full definitions.

Task interface and executable ground truth. In both conditions, the evaluated model receives a natural-language description of the decision problem and may construct and execute code before returning a final scalar objective value. We organize problems into executable benchmark protocols, each defining a problem family with a structured problem schema, an event simulator, and a verified reference solver.

Let z denote a complete instance containing deterministic inputs and latent stochastic parameters. For protocol p, the simulator G_p produces an observation history D, and the reference solver S_p computes the optimal scalar objective $y^\star$:

$$D = G_p(z; \xi), \qquad y^\star = S_p(z), \tag{1}$$

where ξ denotes simulator randomness. All Workflow tasks use an observation budget of 100 records. The evaluated agent's returned objective y is scored against $y^\star$.

Evaluation metrics. Because ORWORKERBENCH evaluates scalar objective values with verified solver outputs, we score predictions directly rather than using an LLM judge. We report capped relative objective error

$$E(y, y^\star) = \min\left(1, \frac{|y - y^\star|}{\max(|y^\star|, 10^{-12})}\right), \tag{2}$$

and **tolerance accuracy**, the fraction of cases satisfying

$$E(y, y^\star) \leq 0.01. \tag{3}$$

Across repeated evaluations, we report the mean and standard deviation of each metric over four runs with different seeds. We additionally report lower-90% error, computed after excluding the largest 10% of case-level errors, to reduce the influence of extreme failures.

Same-data reference. Workflow introduces an additional source of difficulty that is absent from NL2Code: the hidden stochastic quantities must be estimated from 100 observations. Even a perfect optimizer can therefore differ from the true optimum simply because the observed sample is finite.

To measure this effect, we construct a fixed **same-data reference**. For each protocol, a deterministic estimator uses exactly the same observation history D available to the evaluated agent to estimate the hidden stochastic quantities. We then pass these estimates to the verified reference solver:

$$\hat{z} = g_p(D), \qquad y_{\text{ref}} = S_p(\hat{z}). \tag{4}$$

Thus, y_{ref} answers a simple question: *how well could we do using these same observations if estimation followed a fixed procedure and the downstream optimization were solved correctly?*

We compare the evaluated model with this reference using

$$\Delta E = E(y_{\text{model}}, y^\star) - E(y_{\text{ref}}, y^\star). \tag{5}$$

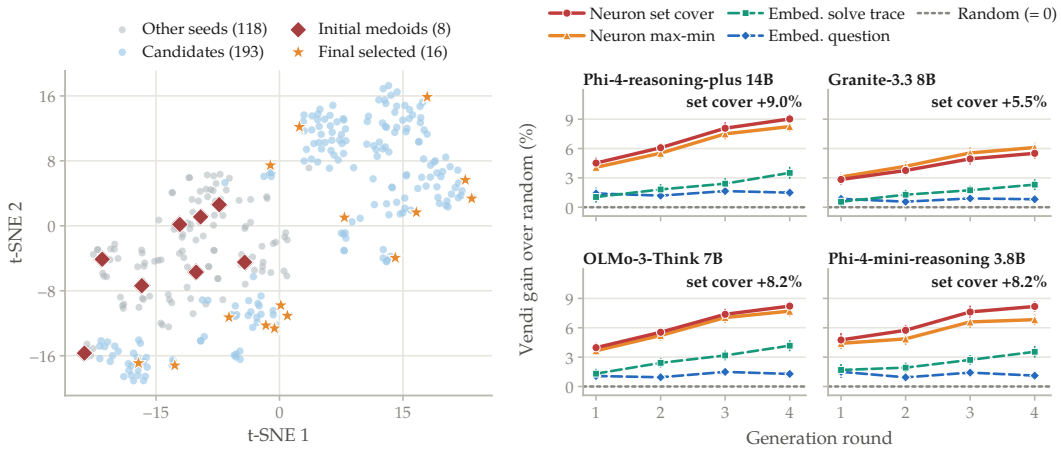
A positive ΔE means that the evaluated model introduces additional error beyond the fixed estimate-then-optimize pipeline, while a negative value means its objective error is lower than that reference.

3.2 NEURON-GUIDED TASK CONSTRUCTION

We use neuron activations to select diverse problem families and expand cases within each family. For each candidate, GPT-5.5 (OpenAI, 2026a) generates a solution trace, which we feed into Qwen3.5-2B (Qwen Team, 2026), a fixed open-weight proxy model. We extract its activated neurons and favor candidates that activate neurons not covered by previously selected tasks.

Selecting and extrapolating problem prototypes. We begin with stochastic optimization questions from OptiVerse (Zhang et al., 2026) and use K-Medoids (Kaufman & Rousseeuw, 1987) to select representative seeds based on their activation patterns. We then generate additional question candidates and greedily select those that activate the most previously uncovered neurons, updating the covered set after each selection. This process expands the candidate pool beyond the initial seeds, as illustrated in Figure 3a. Each selected question then serves as a problem prototype and is formalized into a reusable benchmark protocol for structured benchmark case generation, validation, and evaluation, allowing one human-reviewed protocol to support multiple benchmark cases.

Building executable protocols. Each prototype is formalized as a reusable protocol comprising (1) a schema that defines the required parameters and separates public inputs from hidden stochastic parameters, (2) an event simulator that generates observations, and (3) a golden solver that computes the reference objective. Two PhD-level reviewers validate these components, check Workflow questions for hidden-parameter leakage, and assess both question variants for clarity and natural wording. The reviewed protocols support automated case generation and validation, while neuron-guided selection broadens activation coverage across and within prototypes.



(a) t-SNE of neuron activation representations of generated candidates and selected prototypes.

(b) Mean Vendi gain of each rule over random selection within 48 paired blocks; whiskers show the SEM.

Figure 3: Neuron-guided construction and ablations. (a) Prototype extrapolation extends coverage beyond the seed pool. (b) Under four held-out models, neuron-based rules raise the Vendi score 5.5–9.0% over random selection, well above the embedding baselines.

Expanding instances with quality checks. We then perform structured expansion within each benchmark protocol, using its schema to guide an LLM in generating new instances and paired questions. Programmatic assertions verify schema validity, while a separate LLM judge checks whether the questions match the structured instance, whether public and hidden parameters are correctly separated, and whether Workflow questions reveal hidden information. The judge agrees with human annotations on 132 of 135 labels (97.78%). Neuron-guided selection retains candidates that activate the most previously uncovered neurons within each protocol.

3.3 Quality and Diversity Validation

The final benchmark contains 147 NL2Code and 441 Workflow tasks (L1: 147, L2: 259, L3: 35) across 21 protocols (16 Regular, 5 Hard), with 100 observations per Workflow task. We evaluate the two construction goals separately: whether the tasks satisfy quality checks, and whether neuron guidance yields more diverse activation patterns than embedding-based selection.

Task quality. We assess task quality through protocol review and instance-level checks. Human review retained 21 of 24 proposed protocols. To assess the reliability of LLM-based checks, we compare the judge's decisions with human annotations, finding agreement on 132 of 135 labels across 45 cases (97.78%). A separate audit covers all 441 released Workflow tasks, with the primary judge rating 438 as satisfying all five quality criteria. Appendix C reports the criteria, results, and independent-model checks.

Activation diversity under held-out models. We use activation diversity as a proxy for reasoning diversity to assess whether neuron-guided selection helps the benchmark elicit different reasoning behaviors. We compare five selection rules. Neuron set cover is the rule used in our pipeline, selecting candidates that add the most previously uncovered neurons. Neuron max-min applies the same greedy max-min (k-center) algorithm as the question-embedding and solution-trace-embedding baselines, differing only in the representation used for selection. Random acceptance serves as the fifth rule. All rules share candidate batches and generation budgets across 16 protocols and three seeds. Four held-out models, unused in selection, measure activation diversity using the Vendi score (Friedman & Dieng, 2023).

Figure 3b shows that both neuron-based rules outperform the embedding and random baselines across all four held-out models. The max-min comparison supports the benefit of neuron representations with the selection algorithm held fixed, while set cover tests the rule used in our pipeline,

improving Vendi scores by 5.5–9.0% over random selection. Consistent gains under both rules suggest that neuron-guided selection helps the benchmark elicit a broader range of reasoning behaviors beyond the proxy model.

4 EXPERIMENTS

With ORWorkerBench, we examine the observation-to-decision capability gap through three research questions. **RQ1** measures how much performance degrades when stochastic quantities must be inferred from finite observations rather than provided directly. **RQ2** asks how much of the resulting Workflow error can already be explained by finite-sample estimation and how much additional error is introduced by the agent. **RQ3** examines whether stronger models and specialized optimization tools substantially reduce this gap.

4.1 EVALUATION SETUP

Models. We evaluate both frontier proprietary models and open-weight models. For proprietary models, we use cloud-based APIs to evaluate GPT-4o (OpenAI, 2024), GPT-5.2 (OpenAI, 2025), GPT-5.6 Luna (OpenAI, 2026b), Terra (OpenAI, 2026d), and Sol (OpenAI, 2026c), as well as GPT-6 Astra (OpenAI, 2026e). For open-weight models, we self-host Muse-Glimmer-30B (Meta Superintelligence Lab, 2026) and Gemma-4-31B (Google DeepMind, 2026) with the vLLM framework (Kwon et al., 2023). Reasoning effort is set to the highest available level, while all other sampling parameters follow the default settings specified by the respective model developers.

Evaluation settings. We evaluate NL2Code and Workflow as defined in Section 3.1. The aggregate Workflow comparison also includes unpaired observation variants.

Our main evaluation uses multi-turn interaction, allowing models to inspect the data and iteratively execute code with feedback before returning a final objective value. Following previous OR benchmarks, we additionally evaluate NL2Code in a single-turn setting, where models generate solver code from the natural-language problem without intermediate execution feedback.

For multi-turn evaluation, we use two execution environments. The Python-only environment provides Python execution without optimization packages or external solvers, whereas the Tool-use environment additionally provides OR modeling and scientific computing tools.

Metrics. We use the evaluation metrics, four-run reporting procedure, and same-data reference defined in Section 3.1. The reference estimator averages the raw observations to estimate the hidden stochastic parameters, then passes these estimates to the verified solver.

4.2 RQ1: FROM SPECIFIED OPTIMIZATION TO OBSERVATION-DRIVEN DECISIONS

We first evaluate LLM performance when the stochastic parameters are explicitly specified. As shown in Table 2, the strongest models are already near saturation in NL2Code. Under both multi-turn execution environments, Sol and Astra achieve a mean relative error of only 0.06%, with over 99% tolerance accuracy. This establishes that strongest models can reliably solve optimization problems when the required stochastic quantities are given.

	Python-only		**Tool-use**	
Model	Acc. ↑	Err. ↓	Acc. ↑	Err. ↓
GPT-5.6 Luna	98.64	0.08	98.64	0.12
GPT-5.6 Terra	98.30	0.21	97.62	0.51
GPT-5.6 Sol	**99.32**	**0.06**	**99.32**	**0.06**
GPT-6 Astra	**99.32**	**0.06**	**99.32**	**0.06**
Muse-Glimmer-30B	96.26	3.12	95.92	3.30
Gemma-4-31B	97.62	0.81	94.39	3.11

Table 2: Multi-turn NL2Code performance with stated stochastic parameters.

We then turn to the observation-driven setting, where these quantities must be inferred from finite observations before optimization. Table 3 shows a sharp increase in error. The strongest models incur around 7% overall relative error: Astra achieves 7.35% under Python-only and 7.41% with Tool-use, while Sol reaches 7.54% under both settings.

The increase is especially pronounced on the Hard split. Workflow errors for Sol and Astra rise to roughly 12–13% when the stochastic quantities must be inferred from observations. Strong performance on specified optimization problems therefore does not carry over to observation-driven workflows.

		Python-only			Tool-use		
Model	Split	Acc. (%) ↑	L90 Err. (%) ↓	Mean Err. (%) ↓	Acc. (%) ↑	L90 Err. (%) ↓	Mean Err. (%) ↓
Same-data reference	Regular	21.73	–	5.82	21.73	–	5.82
	Hard	15.24	–	13.30	15.24	–	13.30
	Overall	20.18	–	7.60	20.18	–	7.60
GPT-4o	Regular	14.66 ±0.66	11.17 ±1.07	19.35 ±1.12	9.52 ±1.06	23.35 ±1.81	33.58 ±1.83
	Hard	1.90 ±0.78	47.55 ±3.08	54.94 ±2.08	1.67 ±1.63	65.07 ±22.92	64.24 ±5.53
	Overall	11.62 ±0.60	18.47 ±1.11	27.83 ±1.12	7.65 ±1.02	31.50 ±3.79	40.88 ±2.32
GPT-5.2	Regular	20.68 ±0.62	3.99 ±0.07	6.58 ±0.25	20.24 ±0.73	4.11 ±0.11	6.65 ±0.52
	Hard	15.00 ±2.62	10.51 ±2.40	18.28 ±2.83	14.52 ±0.91	11.19 ±2.02	19.25 ±2.23
	Overall	19.33 ±0.60	4.82 ±0.25	9.37 ±0.78	18.88 ±0.50	4.97 ±0.17	9.65 ±0.70
GPT-5.6 Luna	Regular	20.61 ±0.70	3.94 ±0.06	6.33 ±0.21	20.01 ±1.01	4.05 ±0.06	5.95 ±0.11
	Hard	17.14 ±1.10	7.56 ±1.07	15.13 ±1.95	16.43 ±1.43	7.82 ±0.40	15.37 ±0.88
	Overall	19.78 ±0.50	4.46 ±0.10	8.42 ±0.41	19.16 ±0.92	4.65 ±0.11	8.19 ±0.18
GPT-5.6 Terra	Regular	20.91 ±1.01	3.96 ±0.10	6.15 ±0.36	21.50 ±0.89	3.93 ±0.02	5.95 ±0.26
	Hard	**18.57** ±2.52	6.95 ±0.18	14.09 ±0.55	18.10 ±1.90	7.06 ±0.89	14.32 ±1.71
	Overall	20.35 ±1.29	4.40 ±0.11	8.04 ±0.33	20.69 ±1.09	4.37 ±0.04	7.94 ±0.54
GPT-5.6 Sol	Regular	22.17 ±0.38	**3.81** ±0.08	**5.76** ±0.11	**22.84** ±0.85	3.84 ±0.09	5.79 ±0.21
	Hard	**18.57** ±1.23	**6.30** ±0.58	13.21 ±1.01	**18.57** ±0.95	**6.06** ±0.59	13.11 ±1.38
	Overall	**21.32** ±0.32	**4.20** ±0.08	7.54 ±0.19	**21.83** ±0.77	**4.20** ±0.09	7.54 ±0.41
GPT-6 Astra	Regular	**22.25** ±0.37	3.85 ±0.04	**5.76** ±0.05	22.10 ±0.15	**3.82** ±0.03	**5.74** ±0.04
	Hard	16.90 ±0.91	6.69 ±0.15	**12.44** ±0.31	17.86 ±1.63	6.73 ±0.37	**12.77** ±0.79
	Overall	20.98 ±0.23	4.28 ±0.03	**7.35** ±0.08	21.09 ±0.32	4.25 ±0.05	**7.41** ±0.19
Muse-Glimmer-30B	Regular	20.54 ±0.24	4.01 ±0.06	6.43 ±0.24	20.68 ±1.02	4.07 ±0.06	7.07 ±0.82
	Hard	11.19 ±1.80	9.79 ±3.95	34.87 ±3.61	11.43 ±1.10	11.93 ±3.24	31.86 ±3.40
	Overall	18.31 ±0.43	4.52 ±0.18	13.20 ±0.85	18.48 ±0.68	4.86 ±0.10	12.97 ±0.68
Gemma-4-31B	Regular	20.09 ±0.93	4.00 ±0.04	7.11 ±0.91	19.94 ±0.54	4.18 ±0.18	7.38 ±1.09
	Hard	10.48 ±2.06	20.23 ±3.48	31.77 ±5.15	11.67 ±1.96	16.32 ±1.63	27.98 ±3.13
	Overall	17.80 ±0.43	5.48 ±0.21	12.98 ±1.47	17.97 ±0.22	5.33 ±0.25	12.28 ±1.36

Table 3: Workflow performance at budget 100. Numbers are average ± standard deviation over four runs, and bold marks the best model mean within each split and setting. L90 Err. averages the smallest 90% of uncapped errors among valid completed answers.

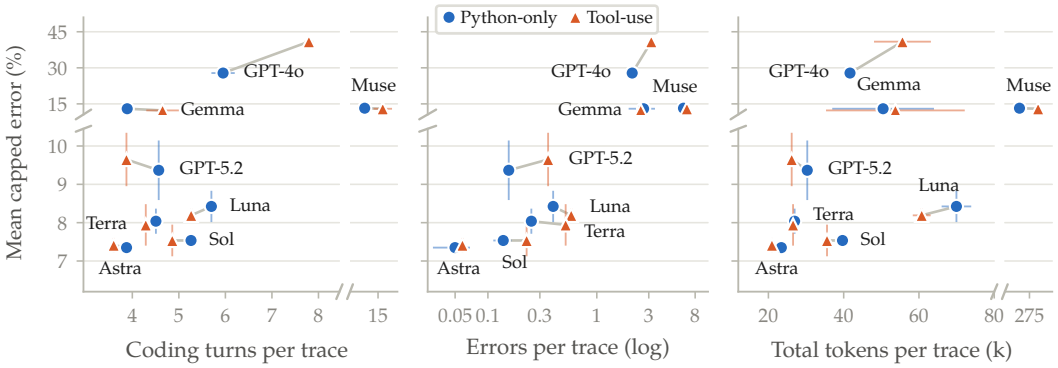
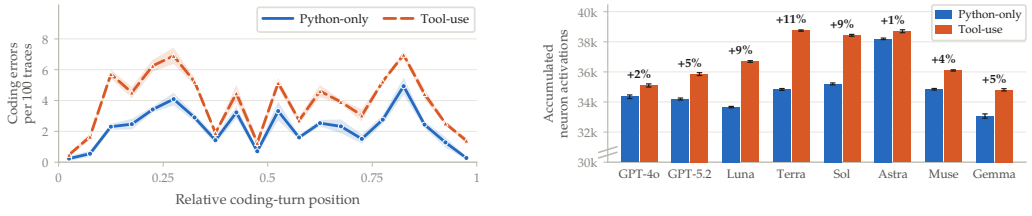


Figure 4: Capped error against execution behaviour for all eight models. Points are four-run means with one-SD bars, joined from Python-only circles to Tool-use triangles.



(a) Coding errors in the six proprietary models' four-run traces.

(b) Numbers of accumulated neuron activations in the first 1,024 assistant tokens, four runs pooled.

Figure 5: Execution diagnostics from the four-run traces. (a) Coding errors per 100 traces by relative coding-turn position, with one-SD bands over runs. (b) Proxy neurons accumulated over the first 1,024 assistant tokens, on an axis starting at 30k, with the Tool-use change labeled.

4.3 RQ2: How Much Error Is Explained by Finite-Sample Estimation?

The Workflow setting introduces two potential sources of error: estimating the hidden stochastic quantities from finite observations, and using those estimates correctly inside the downstream optimization procedure. To separate these effects, we compare each agent with the same-data estimate-then-optimize reference.

As shown in Table 3, the same-data reference incurs 7.60% overall capped error even though the downstream optimization is solved by the verified solver. This shows that a substantial part of the observation-to-decision gap arises before agentic optimization begins: finite-sample estimation alone can move the resulting objective away from the latent optimum.

Most agents add further error on top of this estimation effect, with overall errors ranging from 7.94% to 40.88% across the remaining settings. Only Sol and Astra obtain slightly lower point estimates than the reference. Thus, the strongest models operate close to the error floor induced by this simple same-data pipeline, while weaker models incur substantial additional end-to-end error.

4.4 RQ3: Can Stronger Models and Tools Close the Capability Gap?

The newest frontier models substantially outperform weaker baselines and reduce Workflow error, but finite-data estimation still limits their performance, leaving an observation-to-decision gap. We next ask whether specialized optimization tools provide further benefit. Across the evaluated settings, tool access does not consistently improve objective accuracy and can introduce execution and protocol failures. Neither stronger models nor tool use fully resolves the gap.

More coding turns and tokens are not reliably associated with higher accuracy. Figure 4 compares capped error with coding turns, coding errors, and total tokens (including reported reasoning tokens). Astra has the lowest error with the fewest turns, errors, and tokens; Muse-Glimmer-30B uses the most turns and tokens but incurs nearly twice Astra's error. Among proprietary models, GPT-4o and Luna also spend many turns or tokens without matching gains. All six use more turns on Hard than Regular tasks, suggesting that longer trajectories often reflect difficulty rather than resolve it.

Tool access does not consistently reduce error on Workflow tasks: no model improves beyond run-to-run variation, and the mean across the six proprietary models rises from 11.42% to 13.60%. For all six, error-bearing traces rise from 23.77% to 37.19% on average despite five taking fewer coding turns (Table 13 in Appendix E.3). Figure 5a separately shows more failed executions across coding turns. Early failures often involve mishandling nested records with libraries named in the prompt, such as `pandas`. Coding-error rates among traces that import specialized packages vary by model: with Tool-use, Astra imports a package in 62% of its traces yet has errors in only 6%, versus about 40% for GPT-5.2, Luna, and Terra.

Prior single-turn work associates incorrect responses with broader activation in the generating model (Chen et al., 2026a), motivating an exploratory diagnostic of our trajectories. Figure 5b shows that, among matched, valid pairs whose traces both reach 1,024 assistant tokens, Tool-use increases proxy-neuron coverage over that span for all eight models.

5 Conclusion

We introduced ORWorkerBench to examine the gap between solving specified optimization problems and making decisions from observational data. The neuron-guided construction pipeline expands activation coverage, while human-verified protocols provide paired information conditions, reproducible simulators, and executable ground truth. Evaluations across eight LLMs show that even leading models such as GPT-6 Astra struggle to translate uncertain observations into high-quality decisions. Using additional OR tools, some models slightly improve their objective error, but all six proprietary models exhibit more error-bearing traces with tools. These findings support direct observation-to-decision evaluation and distinguish final decision quality from execution reliability.

Reproducibility Statement

The benchmark data is uploaded as supplementary materials for reproducibility. The benchmark is organized around executable protocol artifacts: each protocol contains the schema, stochastic simulator, frozen solver, and verification logic used to construct and score instances. Tables 11, 3, and 13 are backed by preserved proprietary-model records, trajectory fingerprints, four-run aggregates, and reproducible table-generation scripts. The single-turn entries in Table 11 include the explicitly mapped recovery outcomes described in Appendix E.1; original artifacts remain unchanged, and there is no best-answer selection. Multi-turn results continue to use original outcomes without selective-recovery substitution. The two open-weight models are integrated into the result tables with their supplied NL2Code and archived Workflow aggregate provenance. The appendix documents synthesis, difficulty variants, prompts, execution accounting, and explicitly historical neuron diagnostics.

AI Use Statement

Generative AI tools were used in the research workflow for benchmark candidate generation, LLM-assisted validity checking, and model evaluation. Generated benchmark candidates were not accepted as ground truth by default: protocol schemas, simulators, and frozen solvers were subject to human verification, and generated child cases were checked against the protocol contract. The authors reviewed the AI-assisted analyses, text, and artifacts and take responsibility for the final content of this work.

References

Ali AhmadiTeshnizi, Wenzhi Gao, and Madeleine Udell. OptiMUS: Scalable optimization modeling with (MI)LP solvers and large language models, 2024. URL https://arxiv.org/abs/2402.10172.

Ali AhmadiTeshnizi, Wenzhi Gao, Herman Brunborg, Shayan Talaei, Connor Lawless, and Madeleine Udell. OptiMUS-0.3: Using Large Language Models to Model and Solve Optimization Problems at Scale, August 2025. URL https://arxiv.org/abs/2407.19633v3. arXiv:2407.19633 [cs.AI].

Nora Belrose, Igor Ostrovsky, Lev McKinney, Zach Furman, Logan Smith, Danny Halawi, Stella Biderman, and Jacob Steinhardt. Eliciting latent predictions from transformers with the tuned lens. *arXiv preprint arXiv:2303.08112*, 2023.

Yixin Cao, Jiahao Ying, Yaoning Wang, Xipeng Qiu, Xuanjing Huang, and Yugang Jiang. Model utility law: Evaluating llms beyond performance through mechanism interpretable metric. *arXiv preprint arXiv:2504.07440*, 2025.

Kang Chen, Yaoning Wang, Kai Xiong, Zhuoka Feng, Minshen Yu, Wenhe Sun, Haotian Chen, and Yixin Cao. Do LLMs signal when they're right? evidence from neuron agreement. In *Forty-third International Conference on Machine Learning*, 2026a. URL https://openreview.net/forum?id=ZVRgcQq4D2.

Yitian Chen, Jingfan Xia, Siyu Shao, Dongdong Ge, and Yinyu Ye. Solver-informed RL: Grounding large language models for authentic optimization modeling. In *Advances in Neural Information Processing Systems*, volume 38, 2025. doi: 10.52202/085713-3539. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/98555b92e5dedbdad5921049281995bc-Abstract-Conference.html.

Zhuowei Chen, Liwei Chen, Christian Schunn, Raquel Coelho, and Xiang Lorraine Li. Neuron-aware active few-shot learning for LLMs. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 33047–33062, San Diego, California, United States, July 2026b. Association for Computational Linguistics. ISBN 979-8-89176-390-6. doi: 10.18653/v1/2026.acl-long.1526. URL https://aclanthology.org/2026.acl-long.1526/.

Cody Coleman, Christopher Yeh, Stephen Mussmann, Baharan Mirzasoleiman, Peter Bailis, Percy Liang, Jure Leskovec, and Matei Zaharia. Selection via proxy: Efficient data selection for deep learning. *arXiv preprint arXiv:1906.11829*, 2019.

Dan Friedman and Adji Bousso Dieng. The Vendi Score: A Diversity Evaluation Metric for Machine Learning. *Transactions on Machine Learning Research*, 2023. URL https://mlanthology.org/tmlr/2023/friedman2023tmlr-vendi/.

Google DeepMind. Gemma 4 31B, 2026. URL https://huggingface.co/google/gemma-4-31B. Model card.

Chenyu Huang, Zhengyang Tang, Shixi Hu, Ruoqing Jiang, Xin Zheng, Dongdong Ge, Benyou Wang, and Zizhuo Wang. ORLM: A customizable framework in training large models for automated optimization modeling. *Operations Research*, 73(6):2986–3009, 2025a. doi: 10.1287/opre.2024.1233. URL https://doi.org/10.1287/opre.2024.1233.

Xuhan Huang, Qingning Shen, Yan Hu, Anningzhe Gao, and Benyou Wang. LLMs for mathematical modeling: Towards bridging the gap between natural and mathematical languages. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pp. 2678–2710. Association for Computational Linguistics, 2025b. doi: 10.18653/v1/2025.findings-naacl.146. URL https://aclanthology.org/2025.findings-naacl.146/.

Caigao Jiang, Xiang Shu, Hong Qian, Xingyu Lu, Jun Zhou, Aimin Zhou, and Yang Yu. LLMOPT: Learning to define and solve general optimization problems from scratch. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=9OMvtboTJg.

Jaehun Jung, Seungju Han, Ximing Lu, Skyler Hallinan, David Acuna, Shrimai Prabhumoye, Mostofa Patwary, Mohammad Shoeybi, Bryan Catanzaro, and Yejin Choi. Prismatic synthesis: Gradient-based data diversification boosts generalization in llm reasoning. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (eds.), *Advances in Neural Information Processing Systems*, volume 38, pp. 90649–90685. Curran Associates, Inc., 2025. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/82a34882f560db982d1257b5af461605-Paper-Conference.pdf.

Leonard Kaufman and Peter J. Rousseeuw. Clustering by means of medoids. In Yadolah Dodge (ed.), *Statistical Data Analysis Based on the L1-Norm and Related Methods*, pp. 405–416, Amsterdam, 1987. North-Holland.

Alaa Khaddaj, Logan Engstrom, and Aleksander Madry. Small-to-large generalization: Training data influences models consistently across scale. In *International Conference on Learning Representations*, volume 2025, pp. 33729–33768, 2025.

Minwei Kong, Chonghe Jiang, Ao Qu, Wenbin Ouyang, Zhaoming Zeng, Xiaotong Guo, Zhekai Li, Junyi Li, Yi Fan, Xinshou Zheng, Xi Jing, Yikai Zhang, Zhiwei Liang, Seonghoo Kim, Runqing Yang, Zijian Zhou, Sirui Li, Han Zheng, Wangyang Ying, Ou Zheng, Chonghuan Wang, Jinglong Zhao, Hanzhang Qin, Cathy Wu, Paul Pu Liang, Jinhua Zhao, and Hai Wang. FrontierOR: Benchmarking LLMs' capacity for efficient algorithm design in large-scale optimization, 2026a. URL https://arxiv.org/abs/2605.25246.

Minwei Kong, Ao Qu, Xiaotong Guo, Wenbin Ouyang, Chonghe Jiang, Han Zheng, Yining Ma, Dingyi Zhuang, Yuhan Tang, Junyi Li, Shenhao Wang, Haris Koutsopoulos, Hai Wang, Cathy Wu, and Jinhua Zhao. AlphaOPT: Formulating optimization programs with self-improving LLM experience library. In *Proceedings of the 32nd ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.2*, pp. 2389–2400. ACM, 2026b. doi: 10.1145/3770855.3817974. URL https://doi.org/10.1145/3770855.3817974.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, pp. 611–626. ACM, 2023. doi: 10.1145/3600006.3613165. URL https://doi.org/10.1145/3600006.3613165.

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?, 2026. URL https://arxiv.org/abs/2606.19787.

Wei Liu, Weihao Zeng, Keqing He, Yong Jiang, and Junxian He. What makes good data for alignment? a comprehensive study of automatic data selection in instruction tuning. In B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun (eds.), *International Conference on Learning Representations*, volume 2024, pp. 22353–22373, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/6091f2bb355e960600f62566ac0e2862-Paper-Conference.pdf.

Meta Superintelligence Lab. Muse Glimmer-30B, August 2026. URL https://huggingface.co/meta-models/Muse-Glimmer-30B. Model card.

OpenAI. GPT-4o, 2024. URL https://developers.openai.com/api/docs/models/gpt-4o. OpenAI API model documentation.

OpenAI. GPT-5.2, 2025. URL https://developers.openai.com/api/docs/models/gpt-5.2. OpenAI API model documentation.

OpenAI. GPT-5.5, 2026a. URL https://developers.openai.com/api/docs/models/gpt-5.5. OpenAI API model documentation.

OpenAI. GPT-5.6 Luna, 2026b. URL https://developers.openai.com/api/docs/models/gpt-5.6-luna. OpenAI API model documentation.

OpenAI. GPT-5.6 Sol, 2026c. URL https://developers.openai.com/api/docs/models/gpt-5.6-sol. OpenAI API model documentation.

OpenAI. GPT-5.6 Terra, 2026d. URL https://developers.openai.com/api/docs/models/gpt-5.6-terra. OpenAI API model documentation.

OpenAI. GPT-6 Astra, 2026e. URL https://developers.openai.com/api/docs/models/gpt-6-astra. OpenAI API model documentation.

Qwen Team. Qwen3.5: Towards native multimodal agents, February 2026. URL https://qwen.ai/blog?id=qwen3.5.

Rindranirina Ramamonjison, Timothy Yu, Raymond Li, Haley Li, Giuseppe Carenini, Bissan Ghaddar, Shiqi He, Mahdi Mostajabdaveh, Amin Banitalebi-Dehkordi, Zirui Zhou, and Yong Zhang. NL4Opt competition: Formulating optimization problems based on their natural language descriptions. In *Proceedings of the NeurIPS 2022 Competitions Track*, volume 220 of *Proceedings of Machine Learning Research*, pp. 189–203. PMLR, 2022. URL https://proceedings.mlr.press/v220/ramamonjison23a.html.

Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A. Smith, Daniel Khashabi, and Hannaneh Hajishirzi. Self-instruct: Aligning language models with self-generated instructions. In Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki (eds.), *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 13484–13508, Toronto, Canada, July 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.acl-long.754. URL https://aclanthology.org/2023.acl-long.754/.

Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyan Jiang, et al. Mmlu-pro: A more robust and challenging multi-task language understanding benchmark. *Advances in Neural Information Processing Systems*, 37:95266–95290, 2024.

Mengzhou Xia, Sadhika Malladi, Suchin Gururangan, Sanjeev Arora, and Danqi Chen. LESS: Selecting influential data for targeted instruction tuning. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 54104–54132. PMLR, 21–27 Jul 2024. URL https://proceedings.mlr.press/v235/xia24c.html.

Ziyang Xiao, Dongxiang Zhang, Yangjun Wu, Lilin Xu, Yuan Wang, Xiongwei Han, Xiaojin Fu, Tao Zhong, Jia Zeng, Mingli Song, and Gang Chen. Chain-of-experts: When LLMs meet complex operations research problems. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=HobyL1B9CZ.

Can Xu, Qingfeng Sun, Kai Zheng, Xiubo Geng, Pu Zhao, Jiazhan Feng, Chongyang Tao, Qingwei Lin, and Daxin Jiang. Wizardlm: Empowering large pre-trained language models to follow complex instructions. In B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun (eds.), *International Conference on Learning Representations*, volume 2024, pp. 30745–30766, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/82eec786fdfbbfa53450c5feb7d1ac92-Paper-Conference.pdf.

Haochen Yang, Ke Zhao, Mengyuan Ma, Xingyu Lu, Xiangfeng Wang, and Hong Qian. OptSkills: Learning generalizable optimization skills from problem archetypes via cluster-based distillation, 2026. URL https://arxiv.org/abs/2605.29829v2.

Zhicheng Yang, Yiwei Wang, Yinya Huang, Zhijiang Guo, Wei Shi, Xiongwei Han, Liang Feng, Linqi Song, Xiaodan Liang, and Jing Tang. OptiBench meets ReSocratic: Measure and improve LLMs for optimization modeling. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=fsDZwS49uY.

Xinyu Zhang, Boxuan Zhang, Yuchen Wan, Lingling Zhang, YiXing Yao, Bifan Wei, Yaqiang Wu, and Jun Liu. OptiVerse: A Comprehensive Benchmark towards Optimization Problem Solving, April 2026. URL http://arxiv.org/abs/2604.21510. arXiv:2604.21510 [cs.CL].

Yanzhao Zhang, Mingxin Li, Dingkun Long, Xin Zhang, Huan Lin, Baosong Yang, Pengjun Xie, An Yang, Dayiheng Liu, Junyang Lin, Fei Huang, and Jingren Zhou. Qwen3 Embedding: Advancing text embedding and reranking through foundation models. *arXiv preprint arXiv:2506.05176*, 2025. URL https://arxiv.org/abs/2506.05176.

Ke Zhao, Zixiang Di, Hong Qian, Xiang Shu, Yaolin Wen, Qitao Shi, Bingdong Li, Xingyu Lu, Xiangfeng Wang, Jun Zhou, Ke Tang, and Yang Yu. MiniOpt: Reasoning to model and solve general optimization problems with limited resources, 2026. URL https://arxiv.org/abs/2606.25832.

Chenyu Zhou, Xinyun Lu, Jiangyue Zhao, Jianghao Lin, Dongdong Ge, and Yinyu Ye. OR-Space: A full-lifecycle workspace benchmark for industrial optimization agents, 2026. URL https://arxiv.org/abs/2605.28158.

WenZhuo Zhu, Zheng Cui, Wenhan Lu, Sheng Liu, and Yue Zhao. DAOpt: Modeling and evaluation of data-driven optimization under uncertainty with LLMs, 2025. URL https://arxiv.org/abs/2511.11576.

A Limitations and Future Work

ORWorkerBench focuses on stochastic OR problems with reproducible simulators and verified reference solutions. Its 21 protocols cover a limited range of operational settings, leaving large-scale, long-horizon, partially observed, and nonstationary problems for future study. Broader evaluations should also include real operational data and varying observation budgets.

Our evaluation measures the accuracy of estimated optimal objective values, which does not directly establish the quality of the resulting decisions. The same-data reference provides a reproducible comparison, but relies on simple estimators and is not a statistical lower bound. Future work should incorporate stronger estimation baselines and evaluate the feasibility and performance of agents' decisions under the true underlying distributions.

Neuron activation diversity is a proxy for reasoning diversity. The gains across held-out models support its use for benchmark construction, but further evidence is needed to connect these gains to differences in reasoning strategies and task coverage. Combining activation-based selection with independent assessments of problem structure and reasoning behavior would strengthen this connection.

B BENCHMARK SYNTHESIS DETAILS

Section 3 presents the benchmark design at a high level; this section supplies the representation, controlled variants, selection procedure, and release counts needed to reproduce that construction.

B.1 ANALYSIS MODEL AND NEURON PATTERN EXTRACTION

The neuron-guided construction in Section 3.2 requires a common representation in which traces from different generators can be compared. We use Qwen3.5-2B with thinking disabled as that analysis model. During construction, a trace is represented by its first 300 solution tokens in chunks of 10 tokens, retaining the top $K = 5{,}000$ layer–neuron pairs per chunk; the settings used for evaluation trajectories are given in Appendix E.5.

To make the coverage rule in Section 3.2 executable, we define the neuron pattern of each trace explicitly. For a trace $x = (q, \mathbf{y})$ with generated tokens $\mathbf{y} = (y_1, \ldots, y_T)$, we replay x through the analysis model $\mathcal{M}_{\text{neu}}$, which places every trace in one neuron coordinate system regardless of the model that generated it. At token y_t and layer ℓ, the MLP activation is

$$\mathbf{a}_t^\ell = \sigma\left(\mathbf{h}_t^\ell \mathbf{W}_{\text{in}}^\ell\right),$$

where $\mathbf{h}_t^\ell$ is the MLP input and $\mathbf{W}_{\text{in}}^\ell$ is the up-projection matrix. Following early-unembedding analyses (Belrose et al., 2023), the contribution of neuron i to the generated token is

$$s_{t,\ell,i} = a_{t,i}^\ell \left\langle \mathbf{w}_{\text{out},i}^\ell, \mathbf{e}_{y_t} \right\rangle,$$

where $\mathbf{w}_{\text{out},i}^\ell$ is the neuron's down-projection vector and $\mathbf{e}_{y_t}$ is the unembedding vector of y_t. Following Cao et al. (2025), the trace is divided into fixed-length chunks $\mathcal{C}_1, \ldots, \mathcal{C}_C$, the top-$K$ pairs of each chunk are retained, and the neuron pattern of x is their union,

$$\mathcal{N}(x) = \bigcup_{c=1}^{C} \text{TopK}_{(\ell,i)} \left\{ \max_{t \in \mathcal{C}_c} s_{t,\ell,i} \right\},$$

encoded as a sparse binary vector over the neurons of $\mathcal{M}_{\text{neu}}$.

B.2 CHOICE OF PROXY MODEL

Because most evaluated models do not expose their internal activations, we analyze all solution traces with a fixed open-weight proxy. To this end, following prior work that uses small proxy models for data selection and attribution (Coleman et al., 2019; Khaddaj et al., 2025), we test whether a small proxy preserves the similarity structure on which selection and trajectory analysis rely. We replay the same MMLU-Pro solution traces (Wang et al., 2024) through a larger reference model and a smaller proxy, extract each trace's neuron pattern with both, and compute the pairwise Jaccard distances between traces under each model. Table 4 reports, per domain, the Spearman rank correlation between the two models' pairwise distances: a high value means that the proxy orders pairs of traces by similarity as the reference model does. Same-family proxies agree closely with their reference, and Qwen3.5-2B reaches an overall correlation of 0.9655 with Qwen3.5-9B; cross-family agreement is lower and varies with the pair. These results support using Qwen3.5-2B as a lightweight proxy for benchmark construction and trajectory analysis.

B.3 CONTROLLED VARIANTS AND HARD SPLIT

The paired-task design in Section 3.1 distinguishes structural difficulty from uncertainty in the observations; this subsection specifies how those two axes are controlled. The Regular and Hard splits differ in problem structure, whereas L1, L2, and L3 differ in the hidden distribution and in the quality of its observations. Appendix B.5 gives the resulting counts.

Hard split. To test whether the observation-to-decision gap persists under greater optimization complexity, five protocols are redesigned with shared resources, cross-stage dependencies, nonanticipativity, or coupled state transitions (Table 5). Each receives a new solver and simulator, and its instances are regenerated through the same pipeline. Every protocol has seven parent instances: its

Reference–Proxy Model	Qwen9B–Qwen4B	Qwen9B–Qwen2B	Qwen9B–Llama3B	Qwen9B–Llama1B	Llama8B–Llama3B	Llama8B–Llama1B	Llama8B–Qwen4B	Llama8B–Qwen2B	OLMo7B–Qwen4B	OLMo7B–Qwen2B
Biology	0.9699	0.9376	0.5584	0.5282	0.9273	0.9372	0.7131	0.7039	0.9268	0.8884
Business	0.9954	0.9882	0.8510	0.8510	0.9702	0.9710	0.8259	0.8194	0.9421	0.9312
Chemistry	0.9925	0.9835	0.7887	0.7775	0.9610	0.9715	0.8446	0.8559	0.9471	0.9508
Computer science	0.9797	0.9597	0.7942	0.8212	0.9279	0.9361	0.7647	0.7524	0.9255	0.9269
Economics	0.9762	0.9472	0.8016	0.7414	0.9216	0.9437	0.5128	0.5018	0.9398	0.9371
Engineering	0.9912	0.9766	0.7144	0.7513	0.9397	0.9328	0.7511	0.7605	0.9792	0.9758
Health	0.9661	0.9481	0.6980	0.6818	0.9307	0.9323	0.4892	0.5322	0.9228	0.9235
History	0.9704	0.9403	0.7490	0.7621	0.9040	0.9259	0.4105	0.3219	0.8843	0.8702
Law	0.9773	0.9287	0.5696	0.5715	0.9419	0.9673	0.3993	0.3410	0.9559	0.9509
Math	0.9896	0.9579	0.7247	0.7023	0.9383	0.9254	0.7284	0.6625	0.9212	0.9163
Other	0.9787	0.9432	0.7515	0.7393	0.8734	0.8950	0.5646	0.5692	0.8793	0.8724
Philosophy	0.9576	0.9278	0.8236	0.7951	0.9455	0.9586	0.4400	0.4207	0.9590	0.9589
Physics	0.9906	0.9777	0.8100	0.7942	0.9505	0.9548	0.6756	0.6686	0.9432	0.9445
Psychology	0.9283	0.8925	0.7851	0.7281	0.9479	0.9605	0.4652	0.4140	0.9178	0.8994
Overall Spearman	0.9877	0.9655	0.7890	0.7839	0.9138	0.9253	0.7055	0.6882	0.9203	0.9202

Table 4: Spearman rank correlation between the pairwise neuron-pattern distances of MMLU-Pro traces under a reference model and under a proxy model, by domain. Qwen denotes the Qwen 3.5 series (Qwen Team, 2026) and Llama the Llama 3 series.

problem prototype and the six instances added by within-protocol expansion. The Workflow tasks below are derived from these parents; the five Hard protocols therefore contribute 35 parents, and the 16 Regular protocols contribute 112.

Family	**Regular**	**Hard**
Crop allocation	one-shot allocation	early signal, then nonanticipative recourse over 9 leaf scenarios
Sequential testing	one project	6 correlated projects sharing test and capital budgets
Perishable inventory	one product	3 products sharing order and storage capacity
Maintenance MDP	one machine	4 machines sharing one repair crew
Queue staffing	fixed staffing for one queue	4 regions over 5 shifts with redeployable staff

Table 5: The five protocols rebuilt for the hard split. Every redesign introduces a coupling between decisions that a single stage or a single resource cannot express.

L1: clean observations from the original distribution. L1 anchors the paired comparison in Section 4.2 by changing only whether the stochastic parameters are stated or must be estimated. The simulator draws a clean 100-record history from the parent's latent instance. The Workflow task shares its deterministic inputs, stochastic parameters, and target optimum with the paired NL2Code task, which states the stochastic parameters explicitly.

L2: alternative hidden distributions. L2 tests whether agents can transfer the same public problem-solving procedure to a changed latent distribution. It replaces the hidden stochastic specification while keeping the public decision problem and the solver interface. Each Regular parent receives two variants and each Hard parent one, drawn from the source families in Table 6. A seeded materializer converts the source into the representation the protocol supports, such as a finite probability mass function, a weighted scenario set, a transition table, or queueing rates; the materialized instance is the ground truth, from which a new history is sampled and the optimum recomputed. Hard protocols use structure-specific transformations, namely lognormal perturbations of scenario probabilities, transition rows, yields, and rates, and a shared-shock negative-binomial demand model for perishable inventory.

L3: corrupted observations. L3 isolates robustness to imperfect logs from changes in the underlying decision problem. Each Hard L2 instance therefore receives an L3 task with the same canoni-

L2 source family	Regular	Hard	Total
Dirichlet	35	0	35
Gamma rates	21	0	21
Independent Poisson	21	0	21
Logistic normal	35	14	49
Lognormal	21	0	21
Lognormal rates	21	7	28
Lognormal scenario tree	0	7	7
Negative binomial	14	0	14
Poisson	14	0	14
Shared-shock negative binomial	21	7	28
Truncated normal	14	0	14
Weibull	7	0	7
Total	224	35	259

Table 6: L2 source families and task counts at the 100-record budget. A source family specifies how a hidden instance is constructed.

cal instance, hidden parameters, question, and optimum but a newly sampled 100-record history, to which four operations are applied in order.

1. **Repeated logging.** 20 records are replaced by copies of other records; lineage fields identify the source of each copy.
2. **Missing entries.** 15% of eligible observation cells are set to `null`.
3. **Numeric corruption.** 20% of the remaining numeric cells receive zero-mean Gaussian noise with standard deviation $0.05 \max(|v|, 1)$ for an original value v, clipped at zero and rounded for integer fields.
4. **Categorical corruption.** 5% of the remaining categorical cells are replaced by another observed label of the same field.

To keep the corruption process auditable, identifiers, time-step counters, actions, and lineage fields are never modified. Because corruption follows repeated logging, copies of one record can differ, as in Appendix F.2. The released files record the seeds, rates, and operation counts.

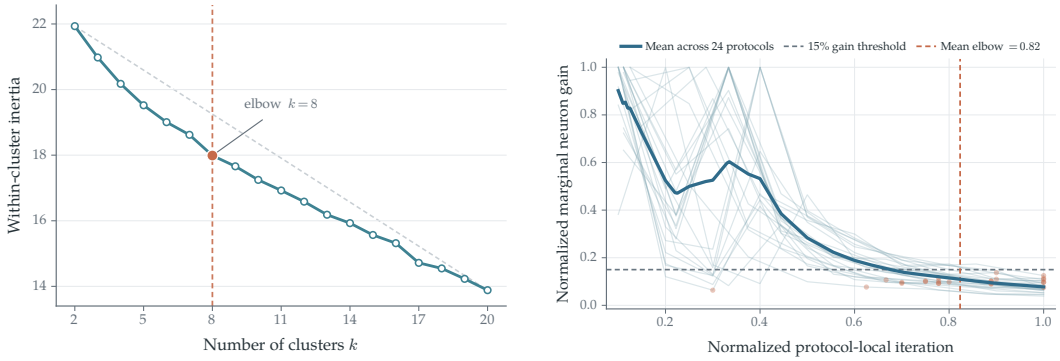
B.4 Construction Models and Selection Procedure

Section 3.2 summarizes prototype selection, generation, expansion, and review; this subsection records the models and stopping rules used at each stage. Table 7 lists the model roles.

Role	Model	Use in construction
Prototype discovery	GPT-5.5 (OpenAI, 2026a)	Propose candidate questions and generate the solution traces used for selection.
Instance expansion	GPT-5.5	Generate schema-constrained instances and paired questions; obtain their solution traces.
Activation extraction	Qwen3.5-2B (Qwen Team, 2026)	Replay traces in a fixed neuron coordinate system with thinking disabled.
Protocol assistance and semantic review	GPT-5.5	Assist protocol construction and separately check canonical instances, hidden parameters, and paired questions.

Table 7: Model roles in benchmark construction. Semantic review checks task specifications; frozen solvers and verifiers establish the reference objectives.

Initial prototypes. The first stage of prototype discovery selects representative starting points before any generation occurs. Of the 134 curated OptiVerse seeds, 126 have solution traces and enter clustering. Their activation sets span a 133,041-dimensional sparse space, which we reduce to



(a) Initial seed clustering. Within-cluster distortion over the sparse activation representations of the 126 clustered seeds.

(b) Within-protocol expansion. Normalized marginal new-neuron coverage against expansion iteration, averaged over protocols.

Figure 6: Neuron coverage drives both selection stages. (a) The elbow that fixes the number of clusters whose medoids become the initial prototypes. (b) The per-iteration coverage gain that determines where within-protocol expansion stops.

50 dimensions by truncated SVD before running K-Medoids for $k \in \{2, \ldots, 20\}$. The number of clusters is the elbow of the inertia curve, the point farthest from the chord joining its two endpoints after min–max normalization, which gives $k = 8$ (Figure 6a). The seed nearest each cluster center becomes an initial prototype.

Prototype generation. Starting from those representative seeds, the second stage searches for problem families that add new activation coverage. Exploration runs for 20 rounds. In each round the generator receives three examples from the frozen prototype pool, candidates that pass format and task checks are solved, and their traces are converted to neuron patterns. The marginal coverage of a candidate x against the selected pool $\mathcal{P}$ is

$$\Delta(x \mid \mathcal{P}) = \left| \mathcal{N}(x) \setminus \bigcup_{s \in \mathcal{P}} \mathcal{N}(s) \right|, \tag{6}$$

and candidates with the largest gains are accepted greedily, with coverage updated after each acceptance. A final global pass starts from the eight initial prototypes and adds sixteen of the 209 unique solved candidates, giving the 24 candidate prototypes that enter protocol review (Section 3.2).

Within-protocol expansion. Once a prototype has been formalized as an executable protocol, the third stage diversifies instances without changing the protocol contract. For each reviewed protocol, the generator proposes five candidates per iteration, conditioned on the core prototype, the schema, and one to three accepted examples; the checks are those of Section 3.2, and the prompt is given in Appendix B.6. Up to three candidates per iteration are accepted greedily by Equation 6, with the pool restricted to the protocol. Expansion stops at the first iteration at which two consecutive gains, normalized by the protocol's largest gain, fall to 0.15 or below, or at the iteration cap. Figure 6b shows the mean gain curve. Protocols stop between three and ten iterations, and the mean normalized stopping point is 0.82, so most of a protocol's budget still adds coverage. A final greedy pass retains six children per protocol, so each released protocol has seven parents.

Semantic review. Because schema and solver checks cannot establish that the two natural-language questions express the intended task, the final stage adds semantic review. Separate GPT-5.5 calls check whether the canonical instance matches the public problem, whether the hidden parameters are complete and consistent with NL2Code, and whether the two questions describe the same problem while Workflow conceals the hidden information. On 45 cases with human annotations, 132 of the 135 judgments agree (97.78%). This is the figure quoted in Section 3.3.

B.5 BENCHMARK COMPOSITION

Table 8 gives the composition of the release at the 100-record budget. Each of the 21 protocols contributes seven parent instances: one problem prototype and six expanded instances. This yields 112 Regular and 35 Hard parents in total. Each parent instance defines one NL2Code task and one L1 Workflow task; Regular parents additionally receive two L2 Workflow tasks and Hard parents one L2 and one L3 task, giving 147 NL2Code and 441 Workflow tasks.

Split	L1	L2	L3	Total
Regular	112	224	–	336
Hard	35	35	35	105
Total	147	259	35	441

Table 8: Workflow tasks by split and observation condition at the 100-record budget. Each of the 147 L1 tasks has a paired NL2Code task.

B.6 INSTANCE EXPANSION PROMPT

The within-protocol stage above depends on a constrained generation interface, so we include its prompt to make the expansion contract inspectable. The prompt receives the protocol's JSON Schema, its core prototype, and one to three accepted examples, and returns a canonical instance together with both question variants in one call, so the NL2Code and Workflow texts are co-generated. The three capitalized tokens are replaced at dispatch time by those inputs serialized as JSON. The listing is condensed.

```
# Protocol-Constrained Benchmark Expansion Prompt

You are generating one new verified-oracle benchmark datapoint for a fixed decision
optimization or stochastic optimization (DO/SO) protocol.

You are given only three inputs: a canonical JSON Schema, one core prototype, and 1-3 same-
protocol few-shot examples. Generate a NEW datapoint that expands reasoning coverage within
the same schema rather than copying the examples.

## Hard Requirements

- Return only valid JSON. No markdown fences and no prose outside JSON.
- Generate exactly one new datapoint.
- Stay inside the fixed schema: do not change the hidden/public split, schema contract,
objective type, timing structure, or solver/verifier assumptions implied by the schema.
- The new datapoint must ask for one scalar numerical value.
- The NL2Code question must be self-contained: include all public and hidden data needed to
compute the scalar answer directly.
- The Workflow question must describe the same datapoint and public business context, but
must hide the hidden stochastic/scenario parameters.
- Do not reveal the scalar answer, optimal decision, oracle solution, certificate, or solver
 reasoning in either question.
- Avoid near-duplicate domains, values, labels, and story framing from the core prototype
and few-shot examples.
- Keep probabilities, costs, capacities, rewards, horizons, units, and objective definition
internally consistent.

## Canonical Instance Contract

- `canonical_instance` is the structured source of truth used by the solver/verifier.
- `canonical_instance` must validate against the provided JSON Schema.
- Preserve the solver-compatible object structure shown in the Core Prototype JSON.
- Put all public and hidden data needed by the solver/verifier inside `canonical_instance`.
- Hidden stochastic/scenario parameters must be stored inside `canonical_instance.
hidden_params`.
- Copy that exact nested object to the top-level `hidden_params` field for validation and
Workflow leakage checks.

## Required Output JSON

{
  "canonical_instance": {},
  "hidden_params": {},
  "nl2code_question": "self-contained scalar value-based NL2Code question",
```

```
  "workflow_question": "same instance as a Workflow question, with hidden stochastic/
scenario parameters hidden"
}

## Canonical JSON Schema

__CANONICAL_SCHEMA_JSON__

## Core Prototype JSON

__CORE_PROTOTYPE_JSON__

## Few-Shot Examples JSON

__FEW_SHOT_EXAMPLES_JSON__

## Generation Instruction

Generate one new schema-valid benchmark datapoint. It should be natural as a DO/SO problem,
require one scalar numeric answer, and add useful reasoning variation while remaining valid
for the same schema/solver/verifier. Return exactly one JSON object.
```

B.7 Activation Diversity under Held-out Models

Section 3.3 uses held-out activation diversity to test whether neuron guidance adds value beyond ordinary embedding-based diversification. We use Qwen3-Embedding-0.6B (Zhang et al., 2025) for this experiment. This subsection gives the controlled comparison behind that claim. The ablation varies only the acceptance rule of the generation loop. Five rules run for four iterations with the generator, the protocols, the seeds, and the per-iteration candidate budget fixed, and rules that share a (protocol, seed) block also share its candidate batches. Three rules use the same greedy max-min (k-center) algorithm, which at each step picks the candidate least similar to anything already selected, and differ only in the similarity: Jaccard over the (layer, neuron) pairs activated in the proxy model, or cosine over embeddings of the question text or of the solution trace. The remaining rules are the set-cover rule of our pipeline, which picks the candidate that adds the most uncovered neurons, and uniform random acceptance. Rules are compared within 48 blocks (16 protocols, 3 seeds). Four held-out models of 3.8B to 14B parameters score every pool with the Vendi score, the effective number of activation modes.

The resulting held-out comparison supplies the evidence summarized in the main paper. Under all four scorers, both neuron rules finish above the three baselines (Figure 3b). Because the raw scores differ by scorer, roughly 1.6 to 5.1, the figure plots each rule's gain over random acceptance within blocks: after four iterations set cover gains 5.5% to 9.0% and neuron max-min 6.1% to 8.2%, against 2.3% to 4.2% for solution-trace embeddings and 0.8% to 1.5% for question embeddings. The max-min comparison isolates the representation: against the solution-trace embedding rule, neuron max-min adds 0.183 Vendi and wins 43 of 48 blocks and 16 of 16 protocols (permutation $p = 10^{-4}$), and set cover adds 0.213 Vendi and wins 43 of 48 blocks. Against random acceptance both neuron rules win all 48 blocks.

C Quality Audit of the Frozen Workflow Benchmark

The task-quality result in Section 3.3 requires evidence beyond construction-time checks, so this section audits the frozen artifacts independently and at full benchmark scale.

Accordingly, we audit the specifications and scoring contracts of all 441 Workflow tasks, 336 Regular and 105 Hard, with LLM judges.

C.1 Rubric and Judges

To make the quality claim in Section 3.3 measurable, Table 9 defines five criteria, each scored from 0 to 4; a criterion is satisfied at a score of at least 3, and every score must cite a reason and an artifact location. The rubric treats hidden uncertainty, finite samples, and noisy observations as intended task properties. A public view contains the Python-only instructions, the question, and all

100 observations and scores R1, R3, R4, and R5; a private view adds the canonical instance, hidden parameters, schema, and reference artifacts and scores R2.

Criterion	Requirement	Mean /4	Score ≥ 3
R1: Clarity	Decisions, objectives, timing, constraints, units, and observations are specified.	3.87	440/441
R2: Fidelity	The public task agrees with its canonical model and scoring contract.	3.97	438/441
R3: Stochastic OR	Observations inform a meaningful stochastic optimization problem.	4.00	441/441
R4: Coherence	The operational setting, parameters, assumptions, and tradeoffs are coherent.	4.00	441/441
R5: Assessability	Available information supports a defensible attempt without evidenced leakage or shortcuts.	3.98	438/441

Table 9: Primary rubric scores on all 441 Workflow tasks. A criterion is satisfied at a score of at least 3.

To test whether the audit depends on a single judge or pass, GPT-6 Astra assesses all 441 tasks, Gemini 2.5 Pro independently assesses a prespecified random sample of 89 tasks stratified by protocol and level plus the 3 tasks flagged by the primary judge, and a second Astra pass re-assesses 45 tasks.

C.2 Results

With the rubric and comparison protocol fixed, the results support the main paper's task-quality summary. All 441 tasks pass 16 deterministic checks covering artifact integrity, schemas, observation budgets, parameter consistency, and full solver replay. The primary mean score is 99.10 of 100 (median 100), and 438 tasks (99.32%) satisfy all five criteria (Table 9). On the random sample, Gemini rates 92.13% of the tasks as satisfying all criteria against 100% for Astra, with exact per-criterion agreement between the two judges of 83% to 100%, and the repeated Astra pass agrees with the first on 84% to 100% of the tasks per criterion (Table 10). The three tasks that the primary judge rates below the threshold are rated below it by Gemini as well.

		Mean score /100		All criteria satisfied	
Subset	n	Primary	Comparator	Primary	Comparator
Random	89	99.27	97.81	100.00%	92.13%
Flagged by primary	3	71.67	68.33	0.00%	0.00%
Astra repeat	45	98.78	99.00	97.78%	100.00%

Table 10: Matched quality assessments. The comparator is Gemini for the first two rows and a second Astra assessment for the last.

D Evaluation Prompts

The evaluation setup in Section 4.1 compares information and execution conditions whose differences are encoded in their prompts; this section records those interfaces for reproducibility.

D.1 Single-turn NL2Code Prompt

The single-turn NL2Code results in Section 4.2 isolate formulation and solution without interactive execution. In this condition, the model receives the self-contained question with stated stochastic parameters and returns one response, with no data file and no execution.

```
You are solving one decision optimization or stochastic optimization benchmark problem.
```

```
Use only the problem statement supplied by the user. You have no external tools or hidden
data. Solve the problem independently.

You may include reasoning, but your response must end with exactly one line in this form:

FINAL_ANSWER: <number>

`<number>` must be one finite base-10 scalar. Do not include commas, currency symbols,
units, equations, uncertainty ranges, or commentary on the final line. Do not emit more
than one FINAL_ANSWER marker.
```

D.2 MULTI-TURN NL2CODE PROMPTS

To separate the effect of interaction from the effect of hidden information, multi-turn NL2Code runs the same parameter-specified questions through the Workflow harness. The initial inspection of `data.json` is retained for protocol compatibility, and the model may execute and revise Python code for up to 50 rounds with 900 seconds per execution.

Python-only.

```
Solve the user's operational decision problem through an iterative Python-based analysis.

The evaluator provides exactly one execution capability: a persistent Python
interpreter. There is no separate operations-research tool, optimizer service,
or solver API.

Dedicated optimization solvers are unavailable and prohibited. Do not import,
install, call, or invoke OR-Tools (ortools), Gurobi (gurobipy),
scipy.optimize, CVXPY (cvxpy), PuLP (pulp), Pyomo (pyomo),
CPLEX/docplex, MOSEK, HiGHS/highspy, or any other optimization-solver package.
Do not launch command-line solvers or external processes. Attempts are rejected
by the Python execution policy.

General-purpose Python is allowed. You may use the standard library and
installed data-analysis or numerical packages for parsing, statistics, linear
algebra, and simulation, but not their dedicated optimizer interfaces.
Implement the decision algorithm yourself with an appropriate bounded method,
such as enumeration, dynamic programming, Bellman recursion, custom search, or
simulation-based policy evaluation.

Every stochastic parameter this problem needs is stated in the problem text itself. There
is nothing to estimate: take the probabilities, payoffs, costs, and any other quantities
directly from the question.

Build the decision model from the parameters stated in the question and solve it. Use
analytical methods when reliable and computational optimization, numerical integration,
simulation, or sensitivity analysis when necessary. Verify feasibility, numerical
stability, and convergence before reporting the result.

Work iteratively using one executable Python block per response. The Python environment is
state-persistent, so imports, variables, models, and intermediate results remain available
across executions.

Each Python execution has a hard wall-clock limit of 900 seconds. Choose a bounded
algorithm expected to finish comfortably within this limit. A timed-out Python execution
is considered a failed action.

The initial inspection does not complete the task. Before returning a final answer, use
subsequent Python actions to build the decision model from the stated parameters and solve
the requested optimization problem. The final scalar must be an objective value actually
computed by Python. Never use zero or any other number as a placeholder, sentinel, guess,
or fallback; zero is permitted only when the completed computation produces zero.

When the computation is complete, respond with exactly one line in the following format:

FINAL_ANSWER: <number>

The `<number>` value must be a single finite base-10 scalar. Do not include commas, units,
symbols, equations, ranges, uncertainty intervals, or additional commentary. Include
exactly one FINAL_ANSWER marker.
```

Tool-use.

```
Solve the user's operational decision problem through an iterative Python-based analysis.

Every stochastic parameter this problem needs is stated in the problem text itself. There
is nothing to estimate: take the probabilities, payoffs, costs, and any other quantities
directly from the question.

Build the decision model from the parameters stated in the question and solve it. Use
analytical methods when reliable and computational optimization, numerical integration,
simulation, or sensitivity analysis when necessary. Verify feasibility, numerical
stability, and convergence before reporting the result.

Work iteratively using one executable Python block per response. The Python environment is
state-persistent, so imports, variables, models, and intermediate results remain available
across executions. The environment uses Python 3.12 and includes the standard library,
NumPy, SciPy, pandas, scikit-learn, NetworkX, SymPy, PuLP, gurobipy, and CVXPY.

Continue with one Python block per response until the modeling, optimization, and
validation are complete.

Each Python execution has a hard wall-clock limit of 900 seconds (15 minutes). Choose a
bounded algorithm expected to finish comfortably within this limit, preferably within 5
minutes. A timed-out Python execution is considered a failed action.

The initial inspection does not complete the task. Before returning a final answer, use
subsequent Python actions to build the decision model from the stated parameters and solve
the requested optimization problem. The final scalar must be an objective value actually
computed by Python. Never use zero or any other number as a placeholder, sentinel, guess,
or fallback; zero is permitted only when the completed computation produces zero.

After the initial inspection, you must continue with additional Python
actions that build the decision model and solve the optimization problem.

Do not emit FINAL_ANSWER until a successful Python execution has computed
and printed the final objective value.

Never use zero or any other number as a placeholder or fallback.
A zero answer is permitted only when the completed Python optimization
actually computes zero.

When the computation is complete, respond with exactly one line in the following format:

FINAL_ANSWER: <number>

The `<number>` value must be a single finite base-10 scalar. Do not include commas, units,
symbols, equations, ranges, uncertainty intervals, or additional commentary. Include
exactly one FINAL_ANSWER marker.
```

D.3 WORKFLOW PROMPT

The central observation-to-decision comparison in Section 4 uses the Workflow prompt, which hides the stochastic parameters and points the model to `data.json` in a persistent Python working directory. The Python-only variant below names the prohibited solver packages, and the execution policy rejects any attempt to import them; the Tool-use variant differs only in its capability contract, which permits the scientific and OR modeling packages. Each response contains exactly one Python block, the first response must inspect the data schema, and placeholder answers are forbidden. The token `__PYTHON_TIMEOUT_SECONDS__` is set to 900 at dispatch. Both paradigms end with a single line `FINAL_ANSWER: <number>`; a response without a parseable marker is scored as invalid and counts as 100% capped error and zero accuracy.

```
Solve the user's operational decision problem through an iterative Python-based analysis.

The evaluator provides exactly one execution capability: a persistent Python
interpreter. There is no separate operations-research tool, optimizer service,
or solver API.

Dedicated optimization solvers are unavailable and prohibited. Do not import,
install, call, or invoke OR-Tools (ortools), Gurobi (gurobipy),
scipy.optimize, CVXPY (cvxpy), PuLP (pulp), Pyomo (pyomo),
CPLEX/docplex, MOSEK, HiGHS/highspy, or any other optimization-solver package.
Do not launch command-line solvers or external processes. Attempts are rejected
by the Python execution policy.

General-purpose Python is allowed. You may use the standard library and
```

```
installed data-analysis or numerical packages for parsing, statistics, linear
algebra, and simulation, but not their dedicated optimizer interfaces.
Implement the decision algorithm yourself with an appropriate bounded method,
such as enumeration, dynamic programming, Bellman recursion, custom search, or
simulation-based policy evaluation.

A historical dataset is available as data.json in the Python working directory. Use this
dataset to estimate the uncertain quantities required by the decision model. Treat the
observations as a finite, noisy sample.

Begin by loading data.json and inspecting its schema, dimensions, variable types, missing
values, and basic summary statistics without printing the full dataset. Identify how the
available fields relate to the operational decision, the public parameters, constraints,
and the requested objective.

You can try to construct a statistically justified model for the uncertainty.

Combine the fitted stochastic model with the stated operational parameters and solve the
resulting decision problem. Use analytical methods when reliable and computational
optimization, numerical integration, simulation, or sensitivity analysis when necessary.
Verify feasibility, numerical stability, and convergence before reporting the result.

Work iteratively using one executable Python block per response. The Python environment is
state-persistent, so imports, variables, fitted models, and intermediate results remain
available across executions.

The first response must contain only one Python block that opens data.json and inspects
its structure without displaying the entire dataset. Do not include explanatory prose or a
final result in that response.

Continue with one Python block per response until the data analysis, stochastic modeling,
optimization, and validation are complete.

Each Python execution has a hard wall-clock limit of __PYTHON_TIMEOUT_SECONDS__ seconds.
Choose a bounded algorithm expected to finish comfortably within this limit. A timed-out
Python execution is considered a failed action.

The initial inspection does not complete the task. Before returning a final answer, use
subsequent Python actions to estimate the uncertain quantities and solve the requested
optimization problem. The final scalar must be an objective value actually computed by
Python. Never use zero or any other number as a placeholder, sentinel, guess, or fallback;
zero is permitted only when the completed computation produces zero.

When the computation is complete, respond with exactly one line in the following format:

FINAL_ANSWER: <number>

The `<number>` value must be a single finite base-10 scalar. Do not include commas, units,
symbols, equations, ranges, uncertainty intervals, or additional commentary. Include
exactly one FINAL_ANSWER marker.
```

E Additional Evaluation Results

This section follows the order of the main evaluation: it first expands the parameter-specified and observation-driven evidence for RQ1, then documents the execution and neuron-level analyses used in RQ3.

E.1 Parameter-Specified NL2Code Results

Section 4.2 uses NL2Code to establish whether each model can solve the optimization problem before asking it to infer hidden parameters; Table 11 provides the complete single-turn and multi-turn results.

E.2 Results by Observation Condition

After the paired NL2Code–L1 comparison in Section 4.2, separating Workflow results by L1–L3 shows whether distribution shift or corrupted observations account for the remaining difficulty. Table 12 reports the three conditions defined in Appendix B.3: L1 contains the 147 parents of both

		Single-turn		Multi-turn			
				Python-only		Tool-use	
Model	Split	Acc. (%) ↑	Mean Err. (%) ↓	Acc. (%) ↑	Mean Err. (%) ↓	Acc. (%) ↑	Mean Err. (%) ↓
GPT-4o	Regular	18.30 ±1.15	38.25 ±2.74	69.42 ±3.45	7.74 ±2.61	65.62 ±0.89	9.90 ±1.43
	Hard	0.00 ±0.00	84.13 ±4.42	25.71 ±4.67	43.92 ±4.37	17.86 ±1.43	51.15 ±3.03
	Overall	13.95 ±0.88	49.18 ±1.77	59.01 ±2.74	16.36 ±2.12	54.25 ±0.86	19.72 ±1.18
GPT-5.2	Regular	98.44 ±0.45	0.74 ±0.45	98.66 ±0.52	0.31 ±0.44	97.99 ±0.85	0.68 ±0.44
	Hard	0.00 ±0.00	91.59 ±3.11	98.57 ±1.65	1.43 ±1.65	97.14 ±4.04	2.86 ±4.04
	Overall	75.00 ±0.34	22.38 ±0.72	98.64 ±0.56	0.57 ±0.64	97.79 ±0.65	1.20 ±0.73
GPT-5.6 Luna	Regular	95.31 ±1.84	0.48 ±0.30	98.88 ±0.45	0.09 ±0.03	98.44 ±1.34	0.15 ±0.16
	Hard	26.43 ±2.74	14.02 ±5.29	97.86 ±2.74	0.04 ±0.05	99.29 ±1.43	0.01 ±0.03
	Overall	78.91 ±1.47	3.71 ±1.18	98.64 ±0.96	0.08 ±0.04	98.64 ±0.96	0.12 ±0.12
GPT-5.6 Terra	Regular	94.64 ±3.18	0.42 ±0.24	98.21 ±1.03	0.20 ±0.17	98.66 ±0.89	0.09 ±0.04
	Hard	37.86 ±4.29	25.24 ±0.61	98.57 ±2.86	0.21 ±0.40	94.29 ±4.04	1.86 ±2.04
	Overall	81.12 ±2.63	6.33 ±0.30	98.30 ±1.30	0.21 ±0.22	97.62 ±1.42	0.51 ±0.48
GPT-5.6 Sol	Regular	**99.11** ±0.00	**0.07** ±0.00	**99.11** ±0.00	**0.07** ±0.00	**99.11** ±0.00	**0.07** ±0.00
	Hard	**60.71** ±6.34	**4.05** ±2.37	**100.00** ±0.00	**0.00** ±0.00	**100.00** ±0.00	**0.00** ±0.00
	Overall	**89.97** ±1.51	**1.02** ±0.56	**99.32** ±0.00	**0.06** ±0.00	**99.32** ±0.00	**0.06** ±0.00
GPT-6 Astra	Regular	**99.11** ±0.00	**0.07** ±0.00	**99.11** ±0.00	**0.07** ±0.00	**99.11** ±0.00	**0.07** ±0.00
	Hard	97.14 ±2.33	2.86 ±2.33	**100.00** ±0.00	**0.00** ±0.00	**100.00** ±0.00	**0.00** ±0.00
	Overall	98.64 ±0.56	0.74 ±0.56	**99.32** ±0.00	**0.06** ±0.00	**99.32** ±0.00	**0.06** ±0.00
Muse-Glimmer-30B	Regular	98.21 ±1.17	0.23 ±0.18	**99.11** ±0.00	**0.07** ±0.00	98.88 ±0.45	0.08 ±0.02
	Hard	7.14 ±4.58	62.76 ±7.57	87.14 ±1.65	12.86 ±1.65	86.43 ±2.74	13.57 ±2.74
	Overall	76.53 ±1.21	15.12 ±1.83	96.26 ±0.39	3.12 ±0.39	95.92 ±0.56	3.30 ±0.64
Gemma-4-31B	Regular	64.29 ±2.90	4.46 ±0.93	98.21 ±1.03	0.59 ±0.59	95.09 ±2.96	2.65 ±1.80
	Hard	0.71 ±1.32	47.68 ±4.50	95.71 ±1.65	1.54 ±1.65	92.14 ±4.88	4.58 ±5.25
	Overall	49.15 ±2.29	14.75 ±0.94	97.62 ±0.88	0.81 ±0.62	94.39 ±1.79	3.11 ±0.95

Table 11: NL2Code performance with stated stochastic parameters on 112 Regular and 35 Hard tasks, reported as mean ± standard deviation over four runs. Accuracy uses a 1% tolerance, mean error is capped at 100%, and failed or invalid outcomes stay in the denominator.

splits, L2 their 259 alternative-distribution variants, and L3 the 35 corrupted Hard histories; the same-data reference row uses the same case sets.

Model	L1 Err. ↓	L2 Err. ↓	L3 Err. ↓	L1 Acc. ↑	L2 Acc. ↑	L3 Acc. ↑
Python-only						
GPT-4o	28.72 ±2.69	23.48 ±1.74	56.26 ±6.23	10.03 ±1.51	14.00 ±1.15	0.71 ±1.43
GPT-5.2	8.93 ±1.20	8.63 ±0.76	16.61 ±1.35	17.18 ±1.51	21.04 ±0.22	15.71 ±3.69
GPT-5.6 Luna	8.36 ±0.49	7.64 ±0.35	14.45 ±2.60	17.86 ±0.34	21.91 ±0.58	12.14 ±1.43
GPT-5.6 Terra	7.83 ±1.01	7.66 ±0.31	11.72 ±1.91	17.52 ±0.65	22.49 ±1.49	16.43 ±2.74
GPT-5.6 Sol	7.24 ±0.07	7.14 ±0.10	11.69 ±2.61	18.20 ±0.65	23.75 ±0.67	16.43 ±2.74
GPT-6 Astra	7.03 ±0.07	6.99 ±0.10	11.34 ±1.15	18.20 ±0.34	23.65 ±0.37	12.86 ±1.65
Tool-use						
GPT-4o	41.22 ±3.48	37.65 ±1.50	63.37 ±6.52	5.78 ±0.39	9.46 ±1.85	2.14 ±2.74
GPT-5.2	9.00 ±0.77	8.65 ±0.60	19.71 ±1.71	17.35 ±0.39	20.46 ±1.14	13.57 ±2.74
GPT-5.6 Luna	7.63 ±0.25	7.83 ±0.23	13.21 ±1.68	17.52 ±0.34	20.75 ±1.46	14.29 ±2.33
GPT-5.6 Terra	7.16 ±0.51	7.79 ±0.64	12.37 ±2.87	18.20 ±0.34	23.17 ±1.64	12.86 ±1.65
GPT-5.6 Sol	7.45 ±0.26	7.18 ±0.28	10.57 ±2.59	19.05 ±0.00	24.23 ±0.86	15.71 ±3.69
GPT-6 Astra	6.93 ±0.13	6.97 ±0.22	12.67 ±2.20	18.20 ±0.65	23.36 ±0.22	16.43 ±4.88
Same-data reference	7.34	7.30	10.88	17.01	22.39	17.14

Table 12: Workflow performance of the six proprietary models by L1/L2/L3 observation conditions, reported as mean ± standard deviation over four runs. Each run has 147 L1, 259 L2, and 35 L3 cases.

E.3 Execution Behavior Under the Two Harnesses

Section 4.4 argues that Tool-use adds execution failures without reliably improving decision quality; this subsection defines the trajectory accounting behind that comparison. Table 13 is computed from the original trajectories without re-executing generated code. Each proprietary model contributes four runs of 441 cases per setting. An error is a failed execution result or a standalone protocol-format correction, counted once per event; executor infrastructure exits are excluded. A coding turn is a submitted action with an execution result, and errors per trace include error-free traces. All 21,168 records stay in the denominators, including 87 failed and 51 invalid final results, which receive 100% capped error and zero accuracy. The averages over the six proprietary models quoted in Section 4.4 pool the models within each run and then summarize the four run-level values. Figure 5a counts failed executions only. Total tokens in Figure 4 are averaged over traces with a complete provider usage record, 96.0% of GPT-4o Tool-use traces and at least 99.6% elsewhere; for the two open-weight models the counts come from the vLLM server with each model's own tokenizer and include visible reasoning tokens.

E.4 Full Table of Execution Behavior of Workflow Traces under Python-only and Tool-use

	Capped error (%) ↓		**Traces with errors (%) ↓**		**Errors per trace ↓**		**Coding turns per trace ↓**	
Model	Python-only	Tool-use	Python-only	Tool-use	Python-only	Tool-use	Python-only	Tool-use
GPT-4o	27.83 ±1.12	40.88 ±2.32	70.41 ±1.72	93.54 ±0.47	2.11 ±0.19	3.16 ±0.15	5.95 ±0.25	7.79 ±0.09
GPT-5.2	9.37 ±0.78	9.65 ±0.70	13.38 ±1.34	29.42 ±1.38	0.16 ±0.02	0.36 ±0.02	4.57 ±0.04	3.87 ±0.04
GPT-5.6 Luna	8.42 ±0.41	8.19 ±0.18	24.89 ±2.42	35.88 ±1.46	0.40 ±0.03	0.58 ±0.04	5.70 ±0.06	5.26 ±0.03
GPT-5.6 Terra	8.04 ±0.33	7.94 ±0.54	18.31 ±1.21	39.57 ±1.87	0.25 ±0.03	0.52 ±0.04	4.51 ±0.01	4.29 ±0.05
GPT-5.6 Sol	7.54 ±0.19	7.54 ±0.41	11.00 ±1.12	19.50 ±1.95	0.14 ±0.03	0.23 ±0.02	5.26 ±0.02	4.86 ±0.03
GPT-6 Astra	7.35 ±0.08	7.41 ±0.19	4.65 ±1.51	5.22 ±0.64	0.05 ±0.02	0.06 ±0.00	3.88 ±0.02	3.60 ±0.02
Muse-Glimmer-30B	13.20 ±0.85	12.97 ±0.68	98.19 ±0.26	97.96 ±0.49	6.25 ±0.25	6.72 ±0.50	14.38 ±0.35	15.22 ±0.43
Gemma-4-31B	12.98 ±1.47	12.28 ±1.36	85.09 ±1.18	98.53 ±0.54	2.69 ±0.73	2.53 ±0.23	3.89 ±0.10	4.65 ±0.35

Table 13: Execution behaviour of Workflow traces under Python-only and Tool-use, reported as mean ± standard deviation over four runs.

E.5 Proxy-Neuron Analysis of Workflow Traces

To complement the outcome and execution metrics in Section 4.4, this analysis asks whether tool access also changes the internal activation patterns elicited by the trajectories. We pass the assistant text from every four-run Workflow trace of the eight models through the Qwen3.5-2B proxy described in Appendix B.1. We analyze the first 5,000 assistant tokens in 32-token chunks and retain the 5,000 most contributive layer–neuron pairs from each chunk. Python-only and Tool-use traces are paired by run and case when both are completed and valid, and the four runs of a model are pooled, giving 13,782 pairs over the eight models.

To connect the aggregate comparison in the main paper to trajectory length, Figure 5b reports the distinct pairs accumulated over the first 1,024 tokens for pairs whose two traces both reach that length. This subset comprises 61% of GPT-4o pairs, 85% of Gemma-4-31B pairs, and at least 98% of the pairs of every other model. Figure 7 shows the full cumulative curves to 5,000 tokens with 10th–90th percentile bands across traces; shorter traces plateau at their final value. At the endpoint, Tool-use reaches 1.8k to 3.7k more pairs than Python-only for seven of the eight models. The case-clustered bootstrap intervals exclude zero, and the difference has the same sign in every run. For GPT-6 Astra, the gap closes to $+22$, consistent with Astra being the model least affected by tool access in Section 4.4.

F Paired Evaluation Example and Error Case Studies

Sections 4.2 and 4.4 establish the aggregate capability gap and the cost of Tool-use; this section makes those findings concrete with one paired task and two trace-level failure cases. It first isolates

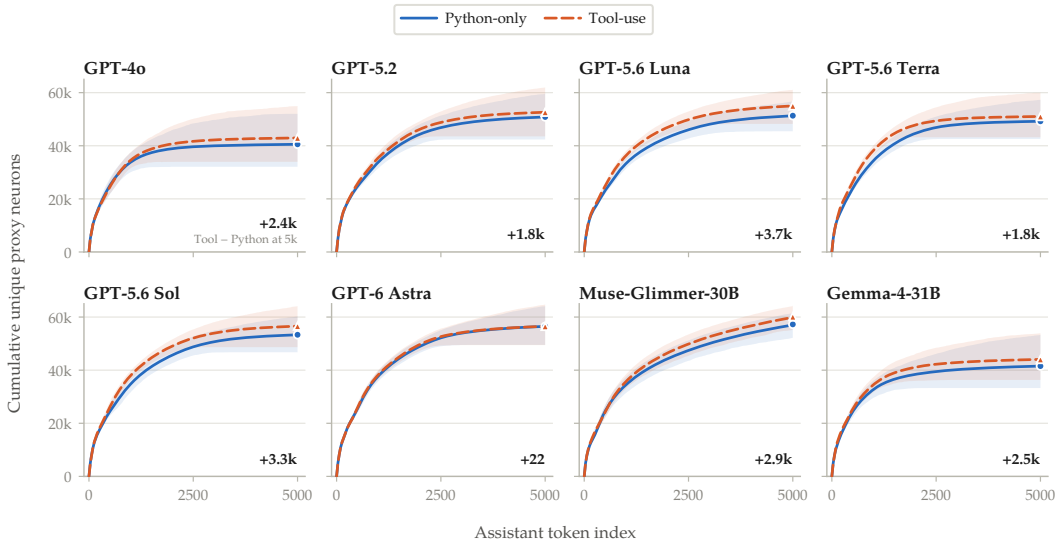


Figure 7: **Proxy-neuron coverage of four-run Workflow traces.** Mean cumulative number of distinct layer–neuron pairs over the first 5,000 assistant tokens, with 10th–90th percentile bands across pooled traces. Each panel states the Tool-use minus Python-only gap at the endpoint.

the effect of hidden parameters, then examines corrupted-record preparation and a failed specialized-solver call.

F.1 Example of Paired Evaluation

To illustrate the paired comparison underlying RQ1, we hold the optimization problem fixed and vary only the information available to the model. The parent is a single-period stocking problem from the stochastic inventory protocol. Both questions describe the same kiosk, costs, and decision, and they are scored against the same optimum. They differ only in the sentence about demand, shown in color: NL2Code states the seven demand scenarios and their probabilities, whereas Workflow withholds them and supplies `data.json` with 100 sampled lunch services.

NL2Code question

An airport food kiosk must choose one integer order quantity Q, between 35 and 120 inclusive, for chilled bento boxes before lunch demand is known. Lunch demand D, in boxes, has the following distribution: $P(D{=}42) = 0.06$, $P(D{=}55) = 0.11$, $P(D{=}67) = 0.19$, $P(D{=}78) = 0.26$, $P(D{=}91) = 0.20$, $P(D{=}104) = 0.13$, and $P(D{=}116) = 0.05$. Each box costs 4.60 euros to prepare, sells for 9.25 euros if demanded, and any unsold box can be salvaged after lunch for 1.50 euros. In addition, the kiosk earns a fixed zero-waste rebate of 18 euros on any lunch service when Q is at most D, and incurs a fixed customer-coupon cost of 45 euros on any lunch service when Q is less than D. Profit for a demand realization is sales revenue from $\min(Q, D)$ boxes plus salvage revenue from $\max(Q - D, 0)$ unsold boxes plus any triggered fixed adjustments, minus preparation cost for Q boxes. Choosing Q to maximize expected lunch profit, what is the maximum expected lunch profit in euros?

Workflow question

An airport food kiosk must choose one integer order quantity Q, between 35 and 120 inclusive, for chilled bento boxes before lunch demand is observed. Each box costs 4.60 euros to prepare, sells for 9.25 euros if demanded, and any unsold box can be salvaged after lunch for 1.50 euros. The kiosk earns a fixed zero-waste rebate of 18 euros when the ordered quantity is at most actual demand, and incurs a fixed customer-coupon cost of 45 euros when the ordered quantity is less than actual demand. The private lunch-demand forecast consists of discrete demand scenarios with probabilities that are not shown here. Profit for a demand realization is sales revenue plus salvage value plus any fixed adjustments, minus preparation cost. Using the hidden demand scenarios, choose Q to maximize expected lunch profit. What is the maximum expected lunch profit in euros?

The first records of the accompanying `data.json` are listed below; the remaining 97 have the same fields.

```
{"history": [{"demand": 78,  "record_id": 0, "scenario_id": "busy_connections"},
             {"demand": 78,  "record_id": 1, "scenario_id": "busy_connections"},
             {"demand": 104, "record_id": 2, "scenario_id": "delayed_departures"}, ...],
 "history_schema": {"record_type": "sampled_scenario_history", "record_budget": 100, ...},
 ...}
```

Demand D	42	55	67	78	91	104	116
Probability stated in NL2Code	0.06	0.11	0.19	0.26	0.20	0.13	0.05
Share of the 100 Workflow records	0.04	0.08	0.15	0.25	0.22	0.19	0.07

Table 14: Demand scenarios of the paired example: probabilities stated in NL2Code and shares observed in the 100 Workflow records.

This pair makes the finite-sample component of the RQ1 gap visible. The optimum is $y^\star = 306.88$ euros at $Q = 91$. The sample over-represents high-demand days, as Table 14 shows, so the same-data reference also orders 91 boxes but values them at $y_{\mathrm{ref}} = 329.49$ euros, a 7.37% error that arises entirely from the estimated probabilities. In Workflow, GPT-5.6 Luna, Terra, and Sol, GPT-6 Astra, and both open-weight models return exactly y_{ref} in all four runs of both settings. GPT-5.2 does so in all four Python-only runs and three of the four Tool-use runs, and GPT-4o in one of four runs in each setting. The agents thus solve the estimated problem correctly and inherit the sampling error of the plug-in estimate, which is the pattern behind the aggregate comparison with the same-data reference in Section 4.3. With the probabilities stated, the same task reduces to enumerating 86 order quantities under a known distribution, so the two conditions isolate the estimation step. The pattern is common: on Regular tasks, 67.7% of the Python-only answers and 59.5% of the Tool-use answers of the six proprietary models equal the reference value up to a relative difference of 10^{-6}, and 60.7% of all paired traces return the identical answer in both settings.

F.2 Case Study: Unhandled Observation Noise

The clean paired example above isolates sampling error; this L3 case shows why corrupted observations create an additional data-preparation challenge in RQ1. It is the L3 variant of a Hard perishable-inventory parent. A stadium kiosk orders three chilled items over six days under shared order and storage limits, and the day-specific joint demand distribution must be estimated from 100 logged records. The L3 history is written with repeated logging and measurement noise: it contains 80 distinct source records and 20 repeat copies, 41 rows have at least one missing cell, and `data.json` declares the logging mode together with the missing and corruption rates. Two logged copies of one source record illustrate the format.

```
{"record_id": 5,  "source_log_record_id": 5, "log_copy_index": 0, "epoch": 6, ...,
 "demand_by_product": {"sushi_pack": 1, "yogurt_parfait": null, "protein_box": 0}}
{"record_id": 56, "source_log_record_id": 5, "log_copy_index": 1, "epoch": 6, ...,
 "demand_by_product": {"sushi_pack": 1, "yogurt_parfait": 0,    "protein_box": 0}}
```

Because every run reads the same file, this setup lets us attribute differences between answers to how the records are prepared. The optimum is $y^\star = 1652.70$, and the same-data reference returns 1372.78. Table 15 groups the runs whose preparation we verified in the trace or by exact replay.

The trace-level contrast explains the answer groups in Table 15. In two of its four runs in each setting, GPT-5.2 prints the noise summary and the record keys during inspection but then estimates each day's joint demand from complete rows only, counting every logged row once:

```
# Empirical joint demand distributions by day (epoch), using complete cases only
for r in hist:
    ...
    for it in items:
        v = coerce_nonneg_int(dct.get(it, None) if isinstance(dct, dict) else None)
        if v is None:
            ok = False
            break
        vec.append(v)
```

Record preparation	Runs	Answer	Error (%)
All 100 rows counted as independent draws; rows with a missing cell dropped	GPT-5.2, 2 of 4 runs per setting	1242.78	24.80
Repeat copies merged; records with a missing cell dropped	Luna, 1 of 4 per setting; Terra, 1 Python-only	1236.50	25.18
Repeat copies merged; partially observed records kept in the likelihood	Astra, 4 of 4 runs per setting	1394.32–1420.19	14.07–15.63
Same-data reference		1372.78	16.94

Table 15: Answers to one L3 task grouped by how the logged records were prepared. Error is the capped relative error against $y^\star = 1652.70$.

```
    if ok:
        dist_by_day[day][tuple(vec)] += 1
```

That complete-case decision keeps 59 of the 100 rows, between 6 and 12 per day, and the dynamic program on that estimate returns 1242.78. Re-executing the archived code block reproduces this value, and feeding it a history whose repeat copies were merged beforehand gives 1236.50. Removing the duplicates therefore does not repair the estimate; the loss comes from discarding every record with a missing cell. All four GPT-6 Astra runs in each setting instead treat the copies of a source record as one observation and keep partially observed records by integrating the missing cells out of the likelihood, and most of them also model the declared corruption rate:

```
# Collapse repeat logs without treating them as independent demand observations.
for source, group in flat.groupby("source_log_record_id", sort=True):
    ...
# Each source record is one demand realization. Missing item readings are
# integrated out, rather than replaced by marginal means.
```

This preparation difference explains much of the case-level error gap and also appears across L3. Astra's errors are 1.3 to 2.9 points below that of the reference. Across all L3 traces, the code refers to the repeated-logging fields in 2.9% of GPT-4o traces and 55.4% of GPT-5.2 traces, against at least 99.6% for GPT-5.6 Luna, Terra, and Sol and GPT-6 Astra. Preparing the corrupted records carefully thus removes about ten points of error on this task, yet the remaining error stays close to that of the same-data reference, in line with the aggregate results in Section 4.3 and Appendix E.2.

F.3 Case Study: Reaching for an Offered Package

The Tool-use prompt names the available packages, and the agents reach for them from the first coding turn. Table 16 reports how often each model's first coding turn imports `pandas` to inspect the nested records, and how often that turn fails. Naming the package raises first-turn `pandas` use from 0.6% to 65% of traces for Terra, from 1% to 96% for Luna, and from 20% to 98% for Sol, and the models that rarely use it under Python-only fail in 4% to 18% of these first turns; Astra, which uses `pandas` in 97% of its Python-only first turns as well, fails in 0.2%. These first-turn failures are the largest single source of the additional coding errors: unhashable-type failures on the nested records rise from 112 to 732 across the five models other than GPT-4o, and 72% of all data-library failures under Tool-use occur in the first two coding turns, before any estimation or optimization code is written.

Terra's first run on an L1 queueing task shows the pattern. A hospital chooses a service rate above a confidential arrival rate to maximize the net clinical value of an M/M/1 station; the optimum is 220.32 and the same-data reference 224.04, a 1.69% error. Under Tool-use, the first turn builds a DataFrame from the whole file and summarizes it, which fails because the history column holds lists:

```
raw = json.load(open("data.json"))
df = pd.DataFrame(raw)
...
print(df.nunique(dropna=True).to_string())
--> error: unhashable type: 'list'
```

	First turn imports `pandas` (%)		Fails when it does (%)
Model	Python-only	Tool-use	Tool-use
GPT-4o	6.0	83.0	87.9
GPT-5.2	5.4	19.2	17.5
GPT-5.6 Luna	1.3	96.4	4.1
GPT-5.6 Terra	0.6	64.8	17.8
GPT-5.6 Sol	20.2	98.1	6.3
GPT-6 Astra	97.4	100.0	0.2

Table 16: First coding turn of the four-run Workflow traces: share of traces that import `pandas`, and the failure rate of that turn under Tool-use.

The second turn rebuilds the frame from the history list, the third estimates the arrival rate and maximizes the closed-form objective, and the run returns 224.04 after 3 coding turns and 10,113 tokens. Under Python-only, the same model inspects the file with the standard library in its first turn,

```
def inspect_obj(x, depth=0):
    if isinstance(x, dict):
        return {"type": "dict", "n_keys": len(x), "keys": list(x.keys())[:30], ...}
    if isinstance(x, list):
        return {"type": "list", "length": len(x), "first_element_structure": ...}
print(inspect_obj(raw))
```

and reaches the same answer after 3 coding turns and 8,921 tokens. Every proprietary model returns 224.04 in every run of both settings except one GPT-4o Tool-use run.

Solver packages show the same pattern at a lower frequency. In GPT-5.2's second run on an L1 energy-contract task, the third Tool-use turn enumerates the 16 contract subsets in plain Python and prints 1684.35, the fourth re-solves each recourse problem with `cvxpy` and requests a solver that the runtime does not provide,

```
prob.solve(solver=cp.ECOS, abstol=1e-9, reltol=1e-9, feastol=1e-9, max_iters=10000, ...)
--> error: The solver ECOS is not installed.
```

and the fifth replaces the call with `scipy.optimize.linprog` and returns the value obtained two turns earlier: one failed execution, one extra turn, and about 10,000 more tokens than the Python-only twin, for the same answer. Across the proprietary-model traces, calls to a module, solver, function, or argument absent from the runtime fail 661 times under Tool-use and 225 times under Python-only; for GPT-5.2, Luna, Terra, and Sol the count rises from 45 to 294, whereas Astra records none and two.

Within the Tool-use setting, where every trace receives the same prompt, importing a specialized package raises the coding-error rate of the same model on the same task by 3.2 points (bootstrap 95% interval 0.5 to 5.7 over the 1,017 model–task cells that contain traces of both kinds).