ContextEval: Evaluating LLM Agent Context Policies for ML Experiment Design

Hikaru Isayama
hisayama@ucsd.edu

Adrian Apsay
adapsay@ucsd.edu

Julia Jung
jmjung@ucsd.edu

Narasimhan Raghavan
naraghavan@ucsd.edu

Ryan Lingo
ryan_lingo@honda-ri.com

Abstract

We study how context visibility influences LLM-driven hyperparameter optimization through controlled factorial experiments varying four information axes across four machine learning benchmarks. Optimization behavior is strongly initialization-dependent. The agent rapidly corrects poorly configured models but yields diminishing returns near strong configurations. Context effects on objective performance are modest and task-dependent. Increased feedback depth encourages more conservative proposals and can hinder recovery from poor initializations. Semantic task descriptions increase trajectory instability on several benchmarks, consistent with the agent lacking a clear optimization direction when given unstructured task context. Explicit parameter bounds reliably eliminate constraint violations but do not improve final performance, highlighting a separation between feasibility and optimization quality. Overall, these results characterize LLM-based optimizers as corrective heuristics that do not consistently outperform random search, and they motivate treating context visibility as a controlled experimental variable in LLM optimization benchmarks.

Website: https://adrianapsay.github.io/context-eval-website/
Code: https://github.com/juliamsjung/context-eval

1 Introduction

Large language models (LLMs) are increasingly deployed as autonomous agents in machine learning workflows, iteratively proposing model configurations, observing evaluation outcomes, and refining decisions over multiple optimization steps Chan et al. (2025); Huang et al. (2024); Liu, Gao and Li (2025).

A central design element in these systems is the context available to the agent. Prompts may include task descriptions, evaluation metrics, parameter constraints, or prior optimization history — all of which influence how the agent interprets the environment and proposes subsequent configurations. However, in most agent-based optimization frameworks, the structure of this informational exposure is not treated as a controlled experimental variable. Context is typically introduced heuristically through prompt engineering, tool integration, or system-specific design choices. As a result, performance differences across agent systems may reflect differences in informational access rather than model capability, making it difficult to attribute observed results or reproduce findings across implementations.

In this work we treat context allocation as a first-class experimental variable. We introduce ContextEval, a controlled evaluation framework for studying how structured informational exposure influences the behavior of LLM optimization agents. A context policy specifies which categories of information are visible to the agent at each optimization step. By holding the model, environment, prompt template, and optimization horizon fixed while varying only the observation mapping, ContextEval isolates the effects of informational exposure on optimization behavior.

We vary four orthogonal axes of context visibility: (i) task description exposure, (ii) evaluation metric exposure, (iii) explicit parameter bounds defining the feasible region, and (iv) temporal depth of historical feedback. Across these axes we construct a full factorial grid of 16 context policies and evaluate agent behavior on four machine learning benchmarks: NOMAD materials regression, Forest Cover Type classification, California Housing regression, and Jigsaw toxicity classification. All experiments use the same frozen LLM policy (GPT-4o-mini) with deterministic decoding and identical prompt templates, ensuring that behavioral differences arise solely from differences in context visibility.

Our central research question is:

> **How does context visibility influence the optimization behavior of LLM agents in iterative machine learning experimentation, and how do these effects interact with initialization quality?**

Across benchmarks, we find that optimization behavior is strongly dependent on initialization quality. Starting from poor configurations, the agent rapidly improves performance within the first few steps, but improvements plateau thereafter. Runs initialized near high-performing configurations exhibit little to no improvement and occasionally degrade. The agent frequently proposes values outside feasible parameter bounds, consistent with coarse search behavior. Context policies produce modest and task-dependent effects on objective performance: explicit parameter bounds eliminate constraint violations but do not improve

optimization quality, while semantic task descriptions can increase trajectory instability. Together, these results suggest that LLM-based optimizers behave primarily as corrective heuristics — effective at repairing poor configurations but limited in their ability to refine strong ones.

These findings highlight the importance of treating context visibility as an explicit design dimension: when informational access is uncontrolled, performance differences across agent systems may reflect differences in what they are told rather than what they can reason.

2 Related Work

2.1 LLM Agents for Machine Learning Experimentation

Recent benchmarks evaluate LLM agents on end-to-end ML experimentation pipelines, including hyperparameter tuning, model selection, and iterative training refinement Chan et al. (2025); Huang et al. (2024). Other systems explore LLM-driven optimization strategies incorporating prior trial history or Bayesian-optimization-inspired prompting Liu, Gao and Li (2025); Agarwal et al. (2025). These frameworks primarily assess improvements in reasoning strategies, search heuristics, or tool integration. In most cases, the information available to the agent — task descriptions, evaluation metrics, prior feedback — is treated as fixed, and performance differences are attributed to model capability or algorithmic design rather than to the structure of the information provided. ContextEval instead treats information exposure as the experimental variable, holding model, prompt template, and environment fixed.

2.2 Hyperparameter Optimization and Initial Design

Classical hyperparameter optimization provides the methodological foundation for our experimental setup. Random search Bergstra and Bengio (2012) serves as a strong baseline for high-dimensional tuning, while Bayesian optimization Snoek, Larochelle and Adams (2012), SMAC Lindauer et al. (2022), and BOHB Falkner, Klein and Hutter (2018) formalize sequential model-based search with explicit initial design phases followed by iterative exploration. ContextEval adopts this two-phase structure — Sobol quasi-random sampling Sobol (1967); Yang and Zhang (2021) for landscape characterization, followed by sequential optimization — not to propose a new optimizer but to provide controlled experimental conditions. We use normalized regret for cross-benchmark comparability, following standard HPO benchmarking conventions Pineda-Arango et al. (2021); Eggensperger et al. (2021); Pfisterer et al. (2022), and stratify initializations using quantile-based partitioning similar to Tree-structured Parzen Estimator methods Bergstra et al. (2011).

2.3 Prompting and Context Sensitivity

LLM performance is highly sensitive to prompt structure and contextual cues. Studies on exemplar selection and ordering Wu et al. (2024), input-order sensitivity Guan et al. (2025), and prompt repetition Leviathan, Kalman and Matias (2025) show that seemingly minor prompt variations can substantially alter outcomes, and recent work demonstrates that even irrelevant context can influence model behavior Wu et al. (2025). However, most of this literature modifies prompt phrasing or reasoning scaffolds, entangling informational content with representational structure. Similarly, iterative agent frameworks such as ReAct Yao et al. (2023) and Reflexion Shinn et al. (2023) modify reasoning processes to improve outcomes, while AgentOccam Yang (2024) demonstrates that refining observation and action spaces alone can substantially improve performance. ContextEval isolates the informational component: prompt structure, decoding, and architecture remain fixed while only the visibility of specific informational categories varies.

3 Methodology

We model the optimization process as a sequential loop in which a frozen LLM policy proposes hyperparameter configurations based on an agent-visible context constructed from the execution history and environment metadata.

3.1 Problem Setting and Framework Overview

We study LLMs acting as iterative optimizers within fixed, offline machine learning environments. At each iteration t, the agent proposes a configuration $\theta_t \in \Omega$, which is evaluated by a deterministic training and scoring pipeline to produce a scalar score:

$$s_t = \mathscr{E}(\mathscr{T}(\theta_t)).$$

where $\mathscr{T}(\cdot)$ trains and evaluates a model and $\mathscr{E}(\cdot)$ extracts the primary objective.

Because the LLM may propose values outside the feasible region, raw proposals are projected onto the feasible set via clamping. Raw proposals are projected onto the feasible set via clamping:

$$\theta_t = \text{clamp}_\Omega(\hat{\theta}_t),$$

Let $\mathscr{H}_{<t} = \{(\theta_i, s_i)\}_{i=0}^{t-1}$ denote the execution trace prior to iteration t, and let $\mathscr{I}$ denote static environment metadata (task descriptions, metric definitions, and parameter bounds). The LLM agent is modeled as a frozen conditional policy

$$\theta_t \sim \pi_{\text{LLM}}(\cdot \mid \mathscr{C}_t),$$

where $\mathscr{C}_t$ is the agent-visible context at iteration t, constructed by a deterministic observational mapping:

$$\Pi_{\text{ctx}} : (\mathscr{H}_{<t}, \mathscr{I}) \to \mathscr{C}_t,$$

The context policy Π_{ctx} specifies which elements of the execution trace and environment metadata are exposed to the agent. Model weights, decoding configuration, prompt template, and environment remain fixed across all conditions; behavioral variation arises solely from differences in Π_{ctx}.

3.2 Initialization Design

To study how initialization quality interacts with context visibility, we construct controlled starting configurations via landscape characterization. Landscape characterization refers to evaluating a pool of reference configurations to estimate the difficulty distribution of the search space. We evaluate a pool of 256 configurations sampled using Sobol quasi-random sequences Sobol (1967); Yang and Zhang (2021), with hyperparameters spanning multiple orders of magnitude sampled in log-space Bergstra and Bengio (2012).

To enable cross-benchmark comparison, we compute normalized regret $r(\theta) \in [0, 1]$ by min-max scaling objective scores against this reference pool, following standard HPO benchmarking conventions Pineda-Arango et al. (2021); Eggensperger et al. (2021); Pfisterer et al. (2022). For higher-is-better metrics, $r(\theta) = \frac{f_{\max} - f(\theta)}{f_{\max} - f_{\min}}$; for lower-is-better metrics, the formula is inverted.

We partition configurations into three performance strata:

Initialization	Regret Range
High	$r \leq 0.20$
Neutral	$0.45 \leq r \leq 0.55$
Low	$r \geq 0.80$

Configurations with $0.20 < r < 0.45$ and $0.55 < r < 0.80$ form guard bands and are excluded. Within each stratum, we select the configuration with median regret as the representative initialization θ_0, avoiding extreme outliers while preserving the characteristic performance level of each regime.

3.3 Context Policies

We instantiate Π_{ctx} through four orthogonal axes corresponding to separable semantic components of the optimization process. Under a given context policy, the agent-visible context takes the form:

$$\mathscr{C}_t = (H_t, T_t, M_t, B),$$

where:

- H_t: truncated history of prior configuration–score pairs, controlled by `feedback_depth` $d \in \{1, 5\}$:
$$H_t = \{(\theta_i, s_i)\}_{i=\max(0,t-d)}^{t-1},$$
where $d \in \{1, 5\}$ controls the visible history window. When $d = 1$, the agent observes only the most recent configuration–score pair.
- T_t: task description (if `show_task` enabled), sourced verbatim from competition specifications.
- M_t: metric definition (if `show_metric` enabled), providing the mathematical evaluation rule but not explicit optimization directionality.
- B: parameter bounds (if `show_bounds` enabled), defining the feasible region Ω.

A full factorial design over these four binary axes yields $2^4 = 16$ context configurations per benchmark.

3.4 Experiment Design

Optimization loop. The sequential phase evaluates each context policy under a fixed horizon of $T = 10$ iterations, formalized in Algorithm 1. Each configuration is evaluated across three random seeds and three stratified initializations (high, neutral, low), yielding $16 \times 3 \times 3 = 144$ runs per benchmark.

Algorithm 1: ContextEval Sequential Optimization Loop

1. Fix initial configuration θ_0; evaluate to obtain s_0.
2. For $t = 1$ to T:
   (a) Construct $\mathscr{C}_t = \Pi_{\text{ctx}}(\mathscr{H}_{<t}, \mathscr{I})$.
   (b) Sample $\hat{\theta}_t \sim \pi_{\text{LLM}}(\cdot \mid \mathscr{C}_t)$.
   (c) Project to feasible region: $\theta_t = \text{clamp}_\Omega(\hat{\theta}_t)$.
   (d) Evaluate θ_t to obtain s_t.
   (e) Append (θ_t, s_t) to history.

Controls. To isolate context policy as the sole experimental variable, all other system components are fixed: model architecture (Table 1), preprocessing pipeline, evaluation metric, train/test split (80/20, `random_state=0`), optimization horizon, and prompt template. The LLM agent uses GPT-4o-mini with $temperature = 0$ (deterministic decoding). The agent is not informed of the optimization horizon or current step index. When the LLM produces an unparseable response, a deterministic fallback applies a ±15% alternating perturbation; across all experiments, this fallback was never invoked. Full implementation details are provided in Appendix A.

Baseline. As a non-adaptive baseline, random search samples configurations uniformly from Ω at each iteration under the same horizon and evaluation budget, without access to any natural-language context.

Table 1: Benchmark tasks used in experiments. Model architectures are fixed for each task.

Benchmark	Model	Dataset	Metric
NOMAD	HistGradientBoostingRegressor	Kaggle NOMAD 2018 bandgap prediction	RMSLE (↓)
Jigsaw	OneVsRest(LogisticRegression)	Kaggle Jigsaw Toxic Comments	Mean AUC (↑)
Forest	RandomForestClassifier	Kaggle Forest Cover Type	Accuracy (↑)
Housing	ExtraTreesRegressor	Kaggle California Housing	RMSE (↓)

All benchmarks use an 80/20 train/test split. Random seeds are configurable via `--seed` (default 0). Feature scaling is applied for regression tasks. For Jigsaw, TF-IDF uses `strip_accents=unicode` and `lowercase=True`.

Layered isolation guarantees. ContextEval enforces strict separation between an **execution layer** (training and scoring), a **context layer** (projection of agent-visible information via Π_{ctx}), and a **logging layer** (full trace recording). All agent-visible inputs are constructed via a centralized `ContextBuilder` with an explicit field allowlist, ensuring that context policy is the only source of informational variation across experiments. Details are provided in Appendix C.

Implementation. All experiments were implemented in Python using `scikit-learn` for model training and evaluation. Language model queries used the OpenAI API (`gpt-4o-mini`, `temperature` $= 0$). Each 10–step optimization run completed in approximately 2-5 minutes on a local CPU environment.

4 Results

4.1 Overall Optimization Behavior

Across all four benchmarks, normalized regret trajectories share a common pattern: rapid improvement in the first two to three steps, followed by stabilization.

Table 2 summarizes optimization performance. Housing and Forest achieve 100% improvement rates with low final regret, while NOMAD and Jigsaw improve in only 67–68% of runs. Jigsaw retains a final regret of 0.431, consistent with the limited trajectory-level gains in Figure 1.

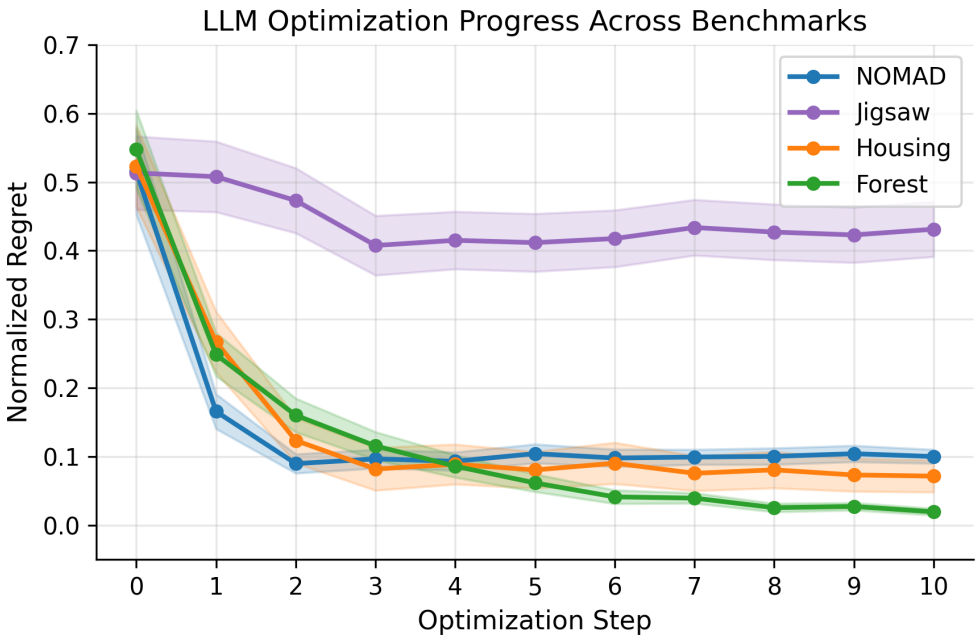

Figure 1: Normalized regret trajectories for the LLM optimizer across benchmarks. Most improvements occur during the first few iterations, followed by stabilization of performance.

4.2 LLM vs Random Search Baseline

Figure 2 compares final best normalized regret for the LLM optimizer and random search. On Forest, the LLM consistently achieves lower regret. On NOMAD and Housing, random search performs competitively or slightly better. The largest gap appears on Jigsaw, where random exploration substantially outperforms the LLM.

These results suggest the LLM's advantage depends on landscape structure. For Forest Cover, where model capacity is controlled by a small number of interpretable parameters, the LLM proposes effective updates. In more complex pipelines such as Jigsaw, random sampling explores the space more effectively within a limited budget. Overall, LLM optimization does not consistently dominate uninformed exploration. This motivates examining what factors — including context visibility and initialization quality — do predict optimization success, which we address in Sections 4.3–4.5.

Table 2: Summary of optimization performance across benchmarks. *% Improved*: fraction of runs where final best score exceeds the initialization score.

Benchmark	Metric	N Runs	Score Range	Best Achieved	% Improved	Final Regret
NOMAD	RMSLE	144	[0.079, 0.203]	0.0808	67%	0.100
Jigsaw	MEAN_AUC	144	[0.903, 0.979]	0.9732	68%	0.431
Housing	RMSE	144	[0.596, 1.116]	0.5895	100%	0.072
Forest	Accuracy	144	[0.630, 0.859]	0.8654	100%	0.019

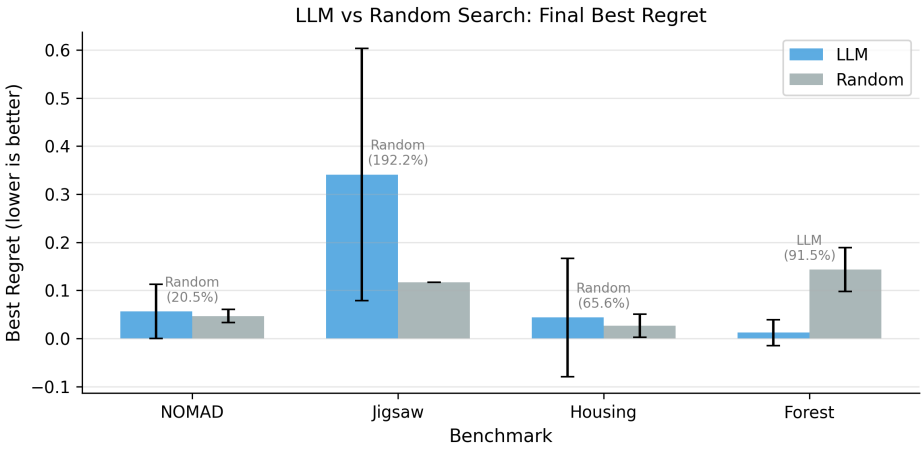

Figure 2: Comparison of LLM-based optimization and random search. Lower regret indicates better performance. Error bars represent standard deviation across three seeds.

4.3 Effect of Initialization Quality

Figure 3 shows regret reduction grouped by initialization quality. The pattern is consistent across benchmarks: poor initializations yield the largest reductions (often exceeding 0.8), neutral initializations show moderate improvements, and high-quality initializations produce only small gains — occasionally degrading, as observed on NOMAD. Jigsaw exhibits smaller improvements at all levels, consistent with its more complex landscape. These results indicate that the LLM optimizer functions primarily as a corrective mechanism, effective at escaping poor configurations but providing diminishing returns near strong ones. We return to the implications of this pattern in Section 5.

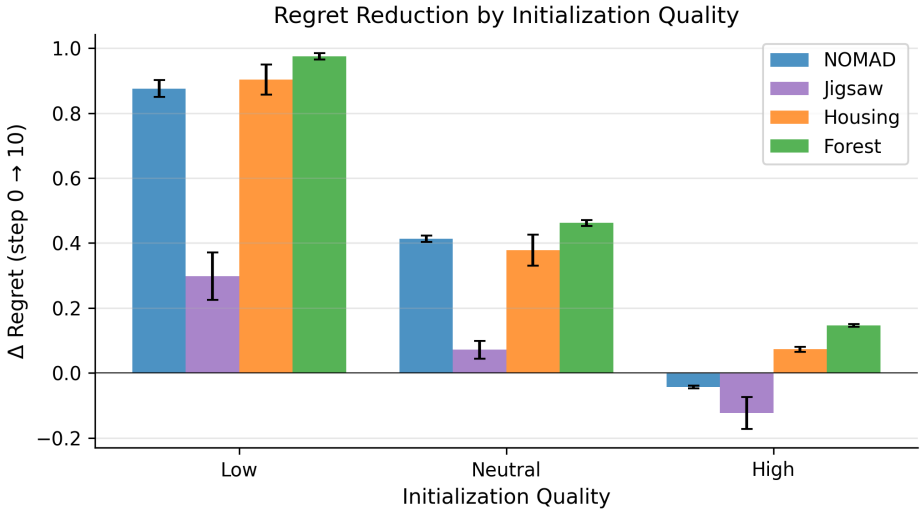

Figure 3: Reduction in normalized regret after $T = 10$ optimization steps across initialization regimes and benchmarks. Improvement is largest for poor initializations and decreases as starting performance improves.

Table 3: Main effects of context axes on final regret. Positive values indicate that enabling the factor increases regret (hurts performance). Feedback depth shows the largest effect across all benchmarks.

Benchmark	fd (1→5)	task (Off→On)	metric (Off→On)	bounds (Off→On)
NOMAD	+0.019	−0.018	+0.004	−0.029
Jigsaw	+0.263	−0.018	−0.015	−0.023
Housing	+0.103	−0.037	+0.032	+0.002
Forest	+0.019	−0.014	−0.002	+0.010

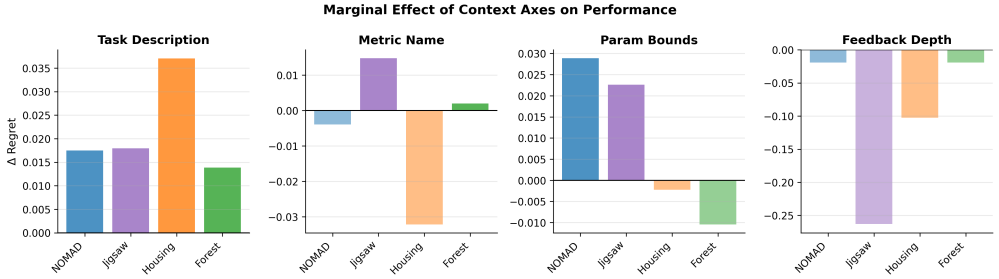

Figure 4: Marginal effect of context axes on final regret across benchmarks. Positive values indicate improved performance when the axis is enabled.

4.4 Effect of Context Policies

We compute the marginal effect of each context axis as the difference in average final regret between runs where the axis is enabled versus disabled. Because each configuration is evaluated across only three random seeds, these marginal effects should be interpreted as descriptive summaries of observed trends rather than statistically confirmed effects.

Table 3 reports the main effect of each axis. Feedback depth is the dominant context factor: increasing from $d = 1$ to $d = 5$ raises regret by 0.019 to 0.263 depending on the task, with the largest effect on Jigsaw. Task descriptions produce small improvements on all benchmarks (never exceeding 0.037 in magnitude). Metric definitions and parameter bounds have inconsistent or negligible effects on objective performance. Figure 4 visualizes these effects per benchmark.

Beyond objective performance, context policies have a striking and consistent effect on constraint adherence. Table 4 shows that exposing parameter bounds nearly eliminates out-of-bounds proposals on NOMAD (243 → 0), Jigsaw (133 → 5), and Housing (6 → 0). Notably, this improvement in feasibility does not translate to improved objective performance (Figure 4), highlighting a separation between constraint adherence and optimization quality.

To situate context effects relative to initialization quality as predictors of optimization outcomes, Table 5 reports effect sizes via η^2. Initialization quality is the dominant factor on Jigsaw ($\eta^2 = 0.497$) and a significant predictor across all benchmarks. Feedback depth explains substantial variance on Jigsaw ($\eta^2 = 0.252$) and Housing ($\eta^2 = 0.174$). Task descriptions and other context axes explain relatively little variance ($\eta^2 < 0.07$). The most

Table 4: Effect of bounds visibility on constraint violations.

Benchmark	Bounds Off	Bounds On	Reduction
NOMAD	243	0	100%
Jigsaw	133	5	96%
Housing	6	0	100%
Forest	60	52	13%

Table 5: Proportion of variance in final regret explained by experimental factors (η^2 from one-way ANOVA). Values above 0.06 indicate medium effects; above 0.14 indicate large effects.

Benchmark	Init η^2	fd η^2	task η^2	Init F	Init p
NOMAD	0.089	0.029	0.024	6.9	0.001
Jigsaw	0.497	0.252	0.001	69.6	<0.001
Housing	0.060	0.174	0.023	4.5	0.013
Forest	0.049	0.128	0.065	3.6	0.029

prominent context effect — feedback depth — interacts with initialization quality in ways a purely marginal analysis obscures, as we show next.

4.5 Interaction Between Initialization and Feedback Depth

The marginal analysis in Section 4.4 treats each context axis independently. However, the factorial design enables analysis of interaction effects — whether the impact of one factor depends on the level of another. We focus on the interaction between initialization quality and feedback depth, which produces the largest and most consistent effect across benchmarks.

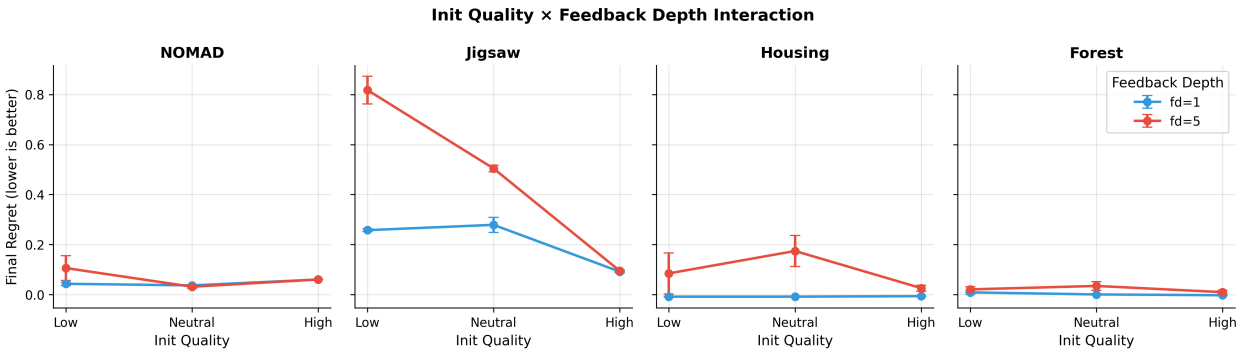

Figure 5: Interaction between initialization quality and feedback depth across benchmarks. Deeper feedback histories (`fd = 5`, red) produce substantially higher regret than shallow histories (`fd = 1`, blue) for poor and neutral initializations, but the gap narrows for high-quality starts. Error bars indicate standard deviation across context policies and seeds.

Table 6: Effect of feedback depth on final regret within each initialization stratum. $\Delta = \text{regret}(\texttt{fd} = 5) - \text{regret}(\texttt{fd} = 1)$; positive values indicate that deeper history hurts. Asterisks denote $p < 0.05$ (paired t-test across context policies; $n = 3$ seeds).

Benchmark	Init	fd=1	fd=5	Δ
NOMAD	Low	0.043	0.106	+0.063*
NOMAD	Neutral	0.037	0.031	−0.005
NOMAD	High	0.060	0.060	+0.000
Jigsaw	Low	0.258	0.818	+0.560*
Jigsaw	Neutral	0.279	0.505	+0.226*
Jigsaw	High	0.092	0.095	+0.003
Housing	Low	−0.008	0.085	+0.093*
Housing	Neutral	−0.008	0.174	+0.183*
Housing	High	−0.006	0.026	+0.032*
Forest	Low	0.009	0.021	+0.012
Forest	Neutral	0.001	0.035	+0.034*
Forest	High	−0.003	0.010	+0.012*

This interaction has a clear interpretation: when the agent starts far from a good configuration, longer feedback histories expose a sequence of poor scores that may anchor the agent's proposals rather than enabling recovery. When the agent starts near a strong configuration, there is little room for feedback depth to matter, as both conditions produce low regret. This asymmetry suggests that feedback depth interacts with the *quality* of the history rather than its *length* per se: a history of poor outcomes may be worse than no history at all.

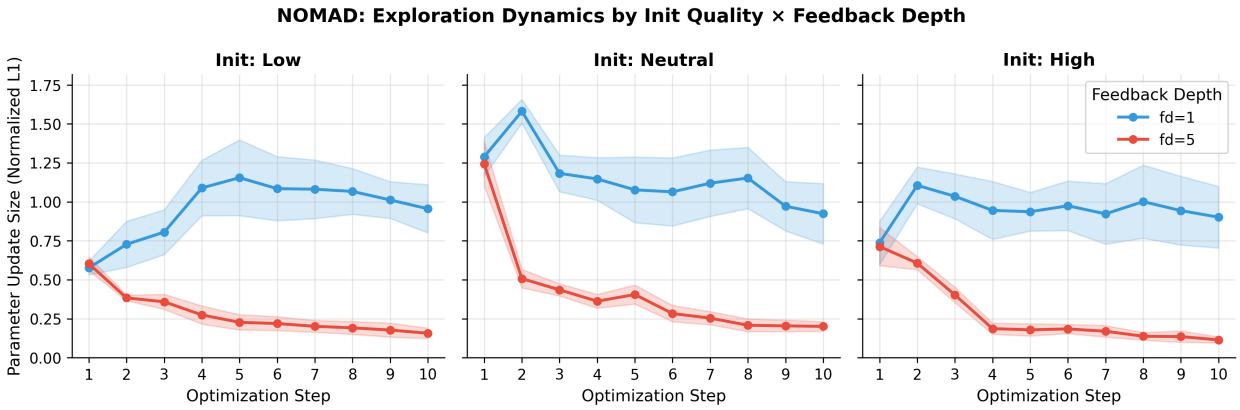

Figure 6: Parameter update magnitude across optimization steps for different initialization regimes and feedback depths on the NOMAD benchmark. Updates are largest early in the trajectory and decrease over time, indicating a transition from exploration to refinement.

4.6 Optimization Dynamics

Figure 6 shows parameter update magnitudes across optimization steps on the NOMAD benchmark. Updates are largest during early iterations and gradually decrease, consistent with a transition from coarse exploration to finer adjustment. Runs initialized poorly maintain larger movements throughout, while high-quality initializations produce smaller, more stable updates. Across all regimes, `feedback_depth`= 5 consistently produces smaller updates than `feedback_depth`= 1, reinforcing the finding that longer histories encourage conservative rather than exploratory behavior. Structurally, the LLM adjusts model-capacity parameters (`max_iter`, `max_leaf_nodes`) aggressively in early steps and tunes continuous parameters (`learning_rate`, `l2_regularization`) more conservatively throughout — consistent with prior-driven behavior discussed in Section 5.

Figure 7 contrasts best- and worst-case optimization trajectories on NOMAD. In the best runs, the agent reduces regret from approximately 0.49 to below 0.03 within two to three steps regardless of feedback depth, consistent with the corrective heuristic pattern. The worst runs reveal a distinct failure mode: all three use `feedback_depth` = 5 with `show_metric` enabled but `show_bounds` disabled, and all start from low-quality initializations. Rather than converging, these trajectories oscillate — regret fluctuates between 0.3 and 0.7 across steps. This reinforces the interaction analysis in Section 4.5: deeper feedback is most harmful precisely when other stabilizing signals such as parameter bounds are absent.

5 Discussion

The central finding across our experiments is that the LLM optimizer behaves as a *corrective heuristic* rather than a systematic search algorithm. It rapidly repairs poor configurations but provides diminishing returns near strong ones, and it does not consistently outperform random search. This characterization has practical implications: LLM-based optimization

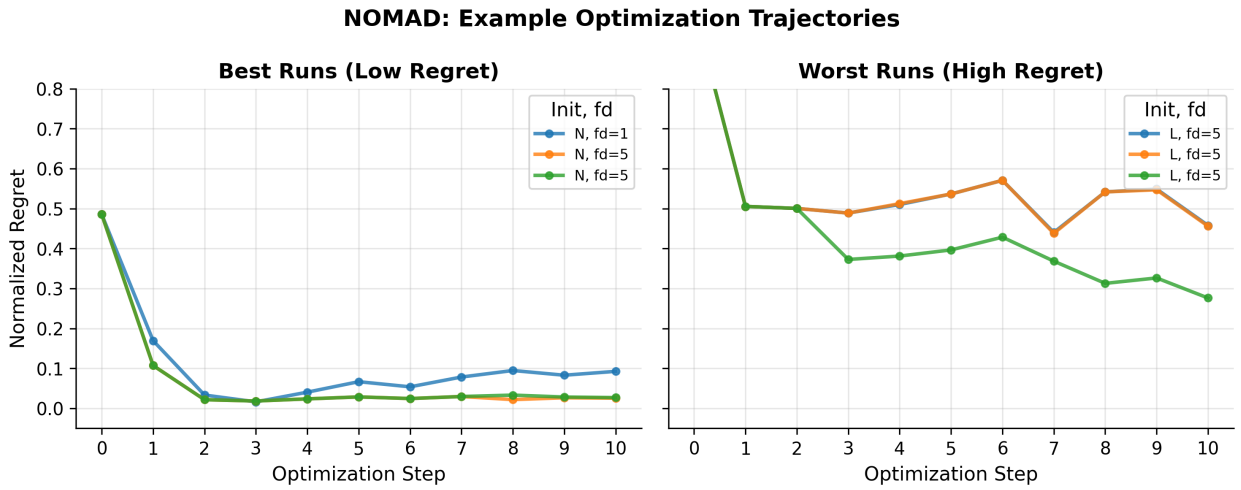

Figure 7: Example optimization trajectories on the NOMAD benchmark. **Left:** Best-performing runs from neutral initialization rapidly reduce regret within 2–3 steps and stabilize near zero. **Right:** Worst-performing runs from low initialization with `feedback_depth = 5` fail to converge, exhibiting sustained oscillation throughout the trajectory.

may be most valuable as a warm-start mechanism or configuration debugger rather than as a replacement for established hyperparameter optimization methods. We note, however, that initialization dependence and limited refinement under short horizons are general properties of any optimizer operating with a constrained evaluation budget — not necessarily unique to LLMs. Without comparison to a sequential model-based baseline such as TPE or Bayesian optimization under the same 10-step horizon, the corrective heuristic characterization is most informative in relative terms, describing *how* the LLM optimizes rather than claiming it is uniquely limited.

The context policy results complicate a natural assumption — that providing more relevant information should improve optimization. In practice, effects are modest, task-dependent, and sometimes counterproductive. Semantic task descriptions occasionally destabilize trajectories, and longer feedback histories encourage conservative proposals rather than better-informed ones. The clearest benefit is structural: explicit parameter bounds nearly eliminate constraint violations without improving objective performance. This separation between feasibility and optimization quality suggests the LLM can follow explicit constraints effectively but struggles to extract actionable optimization signal from richer semantic context.

One mechanism that could explain this pattern is that LLM optimization is driven primarily by general priors about reasonable hyperparameter values — learned during pretraining — rather than by genuine iterative refinement from observed feedback. This would explain why the agent corrects obviously poor configurations quickly (matching them to typical ranges) but cannot improve beyond these priors, and why additional context provides limited benefit when it does not align with what the model already expects. The parameter-level dynamics in Section 4.6 are consistent with this view: the agent adjusts

model-capacity parameters aggressively but tunes continuous parameters conservatively, mirroring a prior-driven rather than feedback-driven search.

The separation between feasibility and optimization quality, and the corrective heuristic characterization more broadly, are likely to generalize beyond GPT-4o-mini: both reflect structural properties of how explicit constraints interact with proposal generation and how pretraining priors interact with observed feedback, rather than quirks of a particular model. Whether larger or reasoning-specialized models exhibit genuine iterative learning — moving beyond prior-based correction — remains an open empirical question.

Finally, these findings have direct implications for LLM agent benchmark design. When context is not controlled as an experimental variable, performance differences across agent systems may reflect differences in informational access rather than in reasoning capability. Benchmark designers should report which information is exposed to the agent and evaluate sensitivity to context variation, rather than treating prompt content as a fixed design choice.

On the Jigsaw benchmark specifically, the limited optimization gains likely reflect a combination of coupled feature-extraction interactions (where `ngram_max` and `max_features` jointly alter the feature representation), a composite multi-label objective that averages over heterogeneous per-label difficulties, and a high-dimensional sparse TF-IDF space that weakens local gradient structure. Together these factors may explain why uninformed random exploration outperforms the LLM on this task: when local structure provides weak guidance, sequential proposals offer little advantage over random sampling.

6 Limitations

The `feedback_depth` axis is evaluated at only two values ($d \in \{1, 5\}$), leaving open whether context effects scale smoothly with history length or exhibit non-monotonic behavior. All experiments use a single language model (GPT-4o-mini) under deterministic decoding; optimization behavior may differ across model families, scales, or decoding strategies. The optimization horizon is fixed at $T = 10$, which suffices to observe early-stage dynamics but may not capture late-stage refinement. Each configuration is evaluated across three random seeds, which is sufficient for reporting uncertainty but limited for formal statistical testing; the effect-size estimates in Table 5 and the paired tests in Table 6 should be interpreted accordingly. Finally, the benchmark suite focuses on tabular ML tasks with fixed model classes; generalization to deep learning pipelines or online settings remains untested. Full methodological details — including Sobol sampling configuration, ContextBuilder validation logic, and the formal connection to partially observable decision processes — are provided in Appendices A and C.

7 Conclusion

We introduced ContextEval, a controlled evaluation framework that treats context visibility as an explicit experimental variable in LLM-based hyperparameter optimization. Our results suggest LLM-based optimization is best understood as a complement to, rather than a replacement for, classical methods: the agent functions as an effective corrective heuristic for poor configurations but provides limited value near strong ones, and does not consistently outperform random search. Context effects on objective performance are modest and task-dependent; the clearest structural benefit — that explicit parameter bounds eliminate constraint violations — does not translate into improved optimization quality. Richer context in the form of longer feedback histories can actively hurt performance, particularly when the agent is anchored to a sequence of poor scores. More broadly, these findings highlight that context visibility should be treated as a controlled variable in LLM agent benchmarks, since uncontrolled variation in informational access may confound evaluations of model capability. Future work should extend this analysis to stronger and reasoning-specialized models, adaptive context policies, and longer optimization horizons to determine whether the corrective-heuristic characterization reflects a fundamental limitation of current LLMs or an artifact of the experimental conditions studied here.

References

Agarwal, Dhruv, Manoj Ghuhan Arivazhagan, Rajarshi Das, Sandesh Swamy, Sopan Khosla, and Rashmi Gangadharaiah. 2025. "Searching for Optimal Solutions with LLMs via Bayesian Optimization." In *International Conference on Learning Representations*. [Link]

Bergstra, James, Remi Bardenet, Yoshua Bengio, and Balazs Kegl. 2011. "Algorithms for Hyper-Parameter Optimization." In *Advances in Neural Information Processing Systems*.

Bergstra, James, and Yoshua Bengio. 2012. "Random Search for Hyper-Parameter Optimization." *Journal of Machine Learning Research* 13: 281–305

Chan, Jun Shern, Neil Chowdhury, Oliver Jaffe, James Aung, Dane Sherburn, Evan Mays, Giulio Starace, Kevin Liu, Leon Maksin, Tejal Patwardhan, Aleksander Madry, and Lilian Weng. 2025. "MLE-Bench: Evaluating Machine Learning Agents on Machine Learning Engineering." In *International Conference on Learning Representations*. [Link]

Eggensperger, Katharina, Philipp Muller, Neeratyoy Mallik, Matthias Feurer, Rene Sass, Aaron Klein, Noor Awad, Marius Lindauer, and Frank Hutter. 2021. "HPOBench: A Collection of Reproducible Multi-Fidelity Benchmark Problems for Hyperparameter Optimization." In *NeurIPS Datasets and Benchmarks Track*.

Falkner, Stefan, Aaron Klein, and Frank Hutter. 2018. "BOHB: Robust and Efficient Hyperparameter Optimization at Scale." In *International Conference on Machine Learning*.

Guan, Bryan, Mehdi Rezagholizadeh, Tanya Roosta, and Peyman Passban. 2025. "The

Order Effect: Investigating Prompt Sensitivity to Input Order in LLMs." In *KDD Workshop on Prompt Optimization*.

Huang, Qian, Jian Vora, Percy Liang, and Jure Leskovec. 2024. "MLAgentBench: Evaluating Language Agents on Machine Learning Experimentation." In *International Conference on Machine Learning*. [Link]

Jones, Donald R, Matthias Schonlau, and William J Welch. 1998. "Efficient Global Optimization of Expensive Black-Box Functions." *Journal of Global Optimization* 13 (4): 455–492

Kaelbling, Leslie Pack, Michael L Littman, and Anthony R Cassandra. 1998. "Planning and Acting in Partially Observable Stochastic Domains." *Artificial Intelligence* 101 (1–2): 99–134

Leviathan, Yaniv, Matan Kalman, and Yossi Matias. 2025. "Prompt Repetition Improves Non-Reasoning LLMs."

Lindauer, Marius, Katharina Eggensperger, Matthias Feurer, Andre Biedenkapp, Difan Deng, Carolin Benjamins, Tim Ruhkopf, Rene Sass, and Frank Hutter. 2022. "SMAC3: A Versatile Bayesian Optimization Package for Hyperparameter Optimization." *Journal of Machine Learning Research* 23 (54): 1–9. [Link]

Liu, Nelson F., Kevin Lin, Michele Bevilacqua, John Hewitt, Fabio Petroni, and Percy Liang. 2023. "Lost in the Middle: How Language Models Use Long Contexts." *arXiv preprint arXiv:2307.03172*. [Link]

Liu, Siyi, Chen Gao, and Yong Li. 2025. "AgentHPO: Large Language Model Agent for Hyper-Parameter Optimization." In *Conference on Parsimony and Learning*. [Link]

Pfisterer, Florian, Lennart Schneider, Julia Moosbauer, Martin Binder, and Bernd Bischl. 2022. "YAHPO Gym: An Efficient Multi-Objective Multi-Fidelity Benchmark for Hyperparameter Optimization." In *AutoML Conference*.

Pineda-Arango, Sebastian, Hadi Jomaa, Martin Wistuba, and Josif Grabocka. 2021. "HPO-B: A Large-Scale Reproducible Benchmark for Black-Box Hyperparameter Optimization." In *NeurIPS Datasets and Benchmarks Track*.

Shinn, Noah, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. "Reflexion: Language Agents with Verbal Reinforcement Learning." In *Advances in Neural Information Processing Systems*.

Snoek, Jasper, Hugo Larochelle, and Ryan Adams. 2012. "Practical Bayesian Optimization of Machine Learning Algorithms." In *Advances in Neural Information Processing Systems*.

Sobol, Ilya M. 1967. "On the Distribution of Points in a Cube and the Approximate Evaluation of Integrals." *USSR Computational Mathematics and Mathematical Physics* 7 (4): 86–112

Wu, Yuchen, Edward Sun, Shuyue Stella Li, Longjie Guo, Yulia Tsvetkov, Jindong Wang, and Aylin Caliskan. 2025. "Irrelevant Context Helps: Understanding the Impact of Context in Large Language Models." [Link]

Wu, Zhaoxuan, Xiaoqiang Lin, Zhongxiang Dai, Wenyang Hu, Yao Shu, See-Kiong

Ng, Patrick Jaillet, and Bryan Low. 2024. "Prompt Optimization with EASE? Efficient Ordering-aware Automated Selection of Exemplars." In *ICML Workshop on In-Context Learning*.

Yang, Ke. 2024. "AgentOccam: Improving LLM-Based Web Agents via Observation and Action Space Adaptation."

Yang, Zebin, and Aijun Zhang. 2021. "Hyperparameter Optimization via Sequential Uniform Designs." *Journal of Machine Learning Research* 22: 1–47. [Link]

Yao, Shunyu, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. "ReAct: Synergizing Reasoning and Acting in Language Models." In *International Conference on Learning Representations*.

Appendices

A Experimental Setup and Reproducibility

A.1 Formal Connections and Methodological Scope

The optimization process defined in Section 3.1 can be understood as inducing a partially observable interface over the environment. The full environment state at iteration t consists of the complete execution trace $\mathscr{H}_{<t}$ and static metadata $\mathscr{I}$, while the context policy Π_{ctx} defines the observation function that projects this state into the agent-visible context $\mathscr{C}_t$. This corresponds to the observation function in the sense of partially observable decision processes Kaelbling, Littman and Cassandra (1998). The sequential evaluate-and-propose loop is structurally analogous to surrogate-based optimization methods such as Bayesian optimization Jones, Schonlau and Welch (1998), SMAC Lindauer et al. (2022), and BOHB Falkner, Klein and Hutter (2018), with the LLM replacing the explicit surrogate model as the proposal engine Liu, Gao and Li (2025); Agarwal et al. (2025). ContextEval is not itself an optimization algorithm; it is an evaluation framework that varies Π_{ctx} within this fixed loop structure.

Methodological scope. The main text presents a streamlined description of the experimental methodology. Several design choices are documented in detail across these appendices: hyperparameter search space definitions and scaling rules (Appendix A.3), prompt template structure and context formatting (Appendix A.4), stratified initialization boundaries and guard band rationale (Appendix B), ContextBuilder validation and trace-isolation architecture (Appendix C), and error handling and fallback logic (Appendix C.2). These details support reproducibility without burdening the main exposition.

A.2 Datasets and Target Metrics

To comprehensively evaluate the ability of the language model to optimize hyperparameter configurations, we utilized four distinct tabular datasets spanning multiple domains, task types, and feature scales. All datasets were sourced from Kaggle competitions and preprocessed in an offline environment to cleanly separate the feature engineering pipeline from the hyperparameter optimization phase evaluated by ContextEval. A summary of the benchmark suite is provided in Table A 1. **Benchmark Details:**

Table A 1: Summary of ContextEval Benchmarks

Benchmark	Domain	Features	Task Type	Target Metric
NOMAD 2018	Materials Science	11	Regression	RMSLE
Jigsaw	NLP / Text	TF-IDF (varies)	Multi-label Classification	Mean AUC
Forest Cover	Geospatial	54	Multi-class Classification	Accuracy
CA Housing	Economics	8	Regression	RMSE

- **NOMAD:** Targets bandgap energy prediction (eV). The optimizer is challenged to tune a HistGradientBoostingRegressor, navigating deep structural hyperparameters like max leaf nodes and L_2 regularization.
- **Jigsaw:** Requires tuning a TF-IDF vectorization pipeline coupled with Logistic Regression. The search space controls feature extraction bounds (e.g., maximum n-grams) and classifier regularization.
- **Forest Cover:** A dense, 54-feature dataset (including one-hot wilderness and soil indicators) where the agent navigates the parameter space of a RandomForestClassifier, balancing tree depth against minimum sample constraints to prevent overfitting.
- **CA Housing:** A classic housing price regression task tuning an ExtraTreesRegressor. This benchmark explicitly includes a binary categorical toggle (`bootstrap` $\in \{0, 1\}$) and expansive numeric bounds to test boundary adherence.

Evaluation Protocol. Each benchmark uses a fixed train/validation split (80/20, stratified for classification tasks) held constant across all experimental runs. The LLM optimizer observes only the validation score after each configuration proposal; no test set is used. This design ensures (1) consistent objective landscapes across context policies and seeds, and (2) no data leakage between the score signal and held-out evaluation. Split random states are seeded per-benchmark for reproducibility

Full details of the Sobol sampling procedure, including scrambling configuration and pool size selection, are provided in Appendix B.

A.3 Hyperparameter Search Spaces

The hyperparameter search space for each benchmark was explicitly defined to challenge the optimization capabilities of the language model agent across both continuous and discrete dimensions. Following the ”wide global” search philosophy established by Bergstra and Bengio (2012), we set parameter bounds to cover a broad range of the feasible region Ω. This design choice ensures that the optimization task requires a true global search rather than local refinement within a pre-tuned neighborhood.

By utilizing these expansive bounds, we test the agent's ability to interpret environment metadata and avoid "manual" guidance from overly restrictive ranges. Table A 2 outlines the exact bounds, types, and scaling rules of every tuned parameter across all four benchmarks. Continuous or extensive parameters spanning multiple orders of magnitude (e.g., learning rates and regularization penalties) were sampled uniformly in log-space during the initial landscape characterization phase Bergstra and Bengio (2012), and we explicitly define these scales below as they impart critical structural context regarding the objective landscape.

Table A 2: Hyperparameter Search Spaces per Benchmark

Benchmark	Hyperparameter	Min Bound	Max Bound	Type	Scale
NOMAD	`learning_rate`	0.005	0.5	Float	Logarithmic
	`l2_regularization`	10^{-4}	5.0	Float	Logarithmic
	`max_depth`	2	20	Integer	Linear
	`max_iter`	50	2000	Integer	Linear
	`max_leaf_nodes`	15	255	Integer	Linear
	`min_samples_leaf`	1	200	Integer	Linear
Jigsaw	`C`	0.001	100.0	Float	Logarithmic
	`max_features`	500	100,000	Integer	Logarithmic
	`ngram_max`	1	3	Integer	Linear
	`min_df`	1	30	Integer	Linear
	`max_iter`	50	2000	Integer	Linear
Forest Cover	`n_estimators`	50	2000	Integer	Linear
	`max_depth`	3	100	Integer	Linear
	`min_samples_split`	2	50	Integer	Linear
	`min_samples_leaf`	1	50	Integer	Linear
	`max_features`	0.1	1.0	Float	Linear
CA Housing	`n_estimators`	10	1500	Integer	Linear
	`max_depth`	2	80	Integer	Linear
	`min_samples_split`	2	80	Integer	Linear
	`min_samples_leaf`	1	60	Integer	Linear
	`max_features`	0.05	1.0	Float	Linear
	`bootstrap`	0	1	Integer	Linear

A.4 Prompt Templates and Context Formatting

The LLM agent receives prompts constructed from a fixed template with configurable context sections. Each prompt consists of a system message and a user message.

System Prompt. The system prompt is constant across all configurations:

System Prompt

You output ONLY valid JSON. No explanations, no markdown, no text outside the JSON object.

User Prompt Structure. The user prompt follows a modular structure with optional sections controlled by context policy flags:

User Prompt Template

Instruction
You are tuning a HistGradientBoostingRegressor.

Current Configuration
{ ”learning_rate”: 0.05,
”max_iter”: 200,
... }

Current Score
score: 0.1245

Previous Steps [if `feedback_depth > 1`]
step 0: score=0.2034, learning_rate=0.01, ...
step 1: score=0.1523, learning_rate=0.05, ...

Task Description [if `show_task=True`]
<competition description text>

Evaluation Metric [if `show_metric=True`]
<metric formula and description>

Parameter Bounds [if `show_bounds=True`]
learning_rate: [0.005, 0.5]
max_iter: [50, 1000]
...

Output Format
Return JSON with exactly these keys: [learning_rate, max_iter, ...].
Values must be numeric. Do not include units or explanations.

A.5 Full Experimental Results Tables

Tables A 3–A 5 present comprehensive results for all 48 experimental configurations (16 context policies × 3 initialization strata) per benchmark. Each cell reports the mean best score ± standard deviation across three random seeds. The “Clamps” column indicates the total number of out-of-bounds proposals that were clamped to valid ranges during optimization.

Table A 3: Full results for NOMAD benchmark (RMSLE, lower is better). Mean ± std over 3 seeds.

Init	Policy	Score (Mean ± Std)	Clamps
low	fd1_t0_m0_b0	0.0861 ± 0.0000	4
low	fd1_t0_m0_b1	0.0830 ± 0.0000	0
low	fd1_t0_m1_b0	0.0838 ± 0.0000	3
low	fd1_t0_m1_b1	0.0815 ± 0.0003	0
low	fd1_t1_m0_b0	0.0855 ± 0.0028	7
low	fd1_t1_m0_b1	0.0842 ± 0.0016	0
low	fd1_t1_m1_b0	0.0867 ± 0.0000	15
low	fd1_t1_m1_b1	0.0824 ± 0.0004	0
low	fd5_t0_m0_b0	0.1029 ± 0.0000	0
low	fd5_t0_m0_b1	0.0816 ± 0.0002	0
low	fd5_t0_m1_b0	0.1267 ± 0.0117	0
low	fd5_t0_m1_b1	0.0825 ± 0.0008	0
low	fd5_t1_m0_b0	0.0893 ± 0.0043	0
low	fd5_t1_m0_b1	0.0829 ± 0.0006	0
low	fd5_t1_m1_b0	0.0870 ± 0.0013	0
low	fd5_t1_m1_b1	0.0829 ± 0.0006	0
neutral	fd1_t0_m0_b0	0.0824 ± 0.0015	10
neutral	fd1_t0_m0_b1	0.0852 ± 0.0005	0
neutral	fd1_t0_m1_b0	0.0816 ± 0.0008	19
neutral	fd1_t0_m1_b1	0.0825 ± 0.0000	0
neutral	fd1_t1_m0_b0	0.0825 ± 0.0000	29
neutral	fd1_t1_m0_b1	0.0854 ± 0.0000	0
neutral	fd1_t1_m1_b0	0.0825 ± 0.0000	33
neutral	fd1_t1_m1_b1	0.0847 ± 0.0000	0
neutral	fd5_t0_m0_b0	0.0872 ± 0.0000	0
neutral	fd5_t0_m0_b1	0.0816 ± 0.0003	0
neutral	fd5_t0_m1_b0	0.0844 ± 0.0012	16
neutral	fd5_t0_m1_b1	0.0818 ± 0.0013	0
neutral	fd5_t1_m0_b0	0.0816 ± 0.0003	5
neutral	fd5_t1_m0_b1	0.0814 ± 0.0000	0
neutral	fd5_t1_m1_b0	0.0817 ± 0.0000	6
neutral	fd5_t1_m1_b1	0.0818 ± 0.0008	0
high	fd1_t0_m0_b0	0.0863 ± 0.0000	12
high	fd1_t0_m0_b1	0.0863 ± 0.0000	0
high	fd1_t0_m1_b0	0.0863 ± 0.0000	23
high	fd1_t0_m1_b1	0.0863 ± 0.0000	0
high	fd1_t1_m0_b0	0.0863 ± 0.0000	27
high	fd1_t1_m0_b1	0.0863 ± 0.0000	0
high	fd1_t1_m1_b0	0.0863 ± 0.0000	30
high	fd1_t1_m1_b1	0.0863 ± 0.0000	0
high	fd5_t0_m0_b0	0.0863 ± 0.0000	0
high	fd5_t0_m0_b1	0.0863 ± 0.0000	0
high	fd5_t0_m1_b0	0.0863 ± 0.0000	4
high	fd5_t0_m1_b1	0.0863 ± 0.0000	0
high	fd5_t1_m0_b0	0.0863 ± 0.0000	0
high	fd5_t1_m0_b1	0.0863 ± 0.0000	0
high	fd5_t1_m1_b0	0.0863 ± 0.0000	0
high	fd5_t1_m1_b1	0.0863 ± 0.0000	0

Table A 4: Full results for HOUSING benchmark (RMSE, lower is better). Mean ± std over 3 seeds.

Init	Policy	Score (Mean ± Std)	Clamps
low	fd1_t0_m0_b0	0.5908 ± 0.0000	0
low	fd1_t0_m0_b1	0.5924 ± 0.0000	0
low	fd1_t0_m1_b0	0.5911 ± 0.0007	0
low	fd1_t0_m1_b1	0.5923 ± 0.0000	0
low	fd1_t1_m0_b0	0.5908 ± 0.0000	0
low	fd1_t1_m0_b1	0.5904 ± 0.0013	0
low	fd1_t1_m1_b0	0.5916 ± 0.0000	0
low	fd1_t1_m1_b1	0.5924 ± 0.0011	0
low	fd5_t0_m0_b0	0.6011 ± 0.0000	0
low	fd5_t0_m0_b1	0.6162 ± 0.0103	0
low	fd5_t0_m1_b0	0.5967 ± 0.0027	3
low	fd5_t0_m1_b1	0.8795 ± 0.1774	0
low	fd5_t1_m0_b0	0.5974 ± 0.0064	0
low	fd5_t1_m0_b1	0.6209 ± 0.0073	0
low	fd5_t1_m1_b0	0.5999 ± 0.0000	0
low	fd5_t1_m1_b1	0.6079 ± 0.0061	0
neutral	fd1_t0_m0_b0	0.5912 ± 0.0008	0
neutral	fd1_t0_m0_b1	0.5924 ± 0.0000	0
neutral	fd1_t0_m1_b0	0.5903 ± 0.0001	0
neutral	fd1_t0_m1_b1	0.5913 ± 0.0015	0
neutral	fd1_t1_m0_b0	0.5917 ± 0.0005	0
neutral	fd1_t1_m0_b1	0.5919 ± 0.0007	0
neutral	fd1_t1_m1_b0	0.5911 ± 0.0000	0
neutral	fd1_t1_m1_b1	0.5923 ± 0.0000	0
neutral	fd5_t0_m0_b0	0.7362 ± 0.0000	0
neutral	fd5_t0_m0_b1	0.6056 ± 0.0026	0
neutral	fd5_t0_m1_b0	0.7358 ± 0.0008	0
neutral	fd5_t0_m1_b1	0.7399 ± 0.1299	0
neutral	fd5_t1_m0_b0	0.7198 ± 0.0735	0
neutral	fd5_t1_m0_b1	0.5993 ± 0.0020	0
neutral	fd5_t1_m1_b0	0.7528 ± 0.0157	0
neutral	fd5_t1_m1_b1	0.6026 ± 0.0122	0
high	fd1_t0_m0_b0	0.5954 ± 0.0000	0
high	fd1_t0_m0_b1	0.5918 ± 0.0008	0
high	fd1_t0_m1_b0	0.5928 ± 0.0000	0
high	fd1_t0_m1_b1	0.5919 ± 0.0007	0
high	fd1_t1_m0_b0	0.5924 ± 0.0007	0
high	fd1_t1_m0_b1	0.5918 ± 0.0000	0
high	fd1_t1_m1_b0	0.5921 ± 0.0006	0
high	fd1_t1_m1_b1	0.5916 ± 0.0003	0
high	fd5_t0_m0_b0	0.5958 ± 0.0011	0
high	fd5_t0_m0_b1	0.6386 ± 0.0003	0
high	fd5_t0_m1_b0	0.5989 ± 0.0064	1
high	fd5_t0_m1_b1	0.6279 ± 0.0058	0
high	fd5_t1_m0_b0	0.6002 ± 0.0073	0
high	fd5_t1_m0_b1	0.6099 ± 0.0019	0
high	fd5_t1_m1_b0	0.5949 ± 0.0004	2
high	fd5_t1_m1_b1	0.6080 ± 0.0014	0

Table A 5: Full results for FOREST benchmark (Accuracy, higher is better). Mean ± std over 3 seeds.

Init	Policy	Score (Mean ± Std)	Clamps
low	fd1_t0_m0_b0	0.8492 ± 0.0023	0
low	fd1_t0_m0_b1	0.8584 ± 0.0004	0
low	fd1_t0_m1_b0	0.8503 ± 0.0025	0
low	fd1_t0_m1_b1	0.8575 ± 0.0000	0
low	fd1_t1_m0_b0	0.8614 ± 0.0022	0
low	fd1_t1_m0_b1	0.8586 ± 0.0012	0
low	fd1_t1_m1_b0	0.8606 ± 0.0031	0
low	fd1_t1_m1_b1	0.8581 ± 0.0006	0
low	fd5_t0_m0_b0	0.8449 ± 0.0000	16
low	fd5_t0_m0_b1	0.8538 ± 0.0026	11
low	fd5_t0_m1_b0	0.8469 ± 0.0017	18
low	fd5_t0_m1_b1	0.8538 ± 0.0000	9
low	fd5_t1_m0_b0	0.8544 ± 0.0085	10
low	fd5_t1_m0_b1	0.8585 ± 0.0011	10
low	fd5_t1_m1_b0	0.8610 ± 0.0027	14
low	fd5_t1_m1_b1	0.8588 ± 0.0017	9
neutral	fd1_t0_m0_b0	0.8574 ± 0.0010	0
neutral	fd1_t0_m0_b1	0.8591 ± 0.0015	0
neutral	fd1_t0_m1_b0	0.8610 ± 0.0025	0
neutral	fd1_t0_m1_b1	0.8575 ± 0.0003	0
neutral	fd1_t1_m0_b0	0.8598 ± 0.0006	0
neutral	fd1_t1_m0_b1	0.8575 ± 0.0023	0
neutral	fd1_t1_m1_b0	0.8601 ± 0.0000	0
neutral	fd1_t1_m1_b1	0.8569 ± 0.0020	0
neutral	fd5_t0_m0_b0	0.8576 ± 0.0031	0
neutral	fd5_t0_m0_b1	0.8481 ± 0.0032	9
neutral	fd5_t0_m1_b0	0.8526 ± 0.0048	0
neutral	fd5_t0_m1_b1	0.8392 ± 0.0061	1
neutral	fd5_t1_m0_b0	0.8625 ± 0.0025	0
neutral	fd5_t1_m0_b1	0.8333 ± 0.0023	0
neutral	fd5_t1_m1_b0	0.8606 ± 0.0010	0
neutral	fd5_t1_m1_b1	0.8530 ± 0.0012	0
high	fd1_t0_m0_b0	0.8604 ± 0.0006	0
high	fd1_t0_m0_b1	0.8601 ± 0.0006	0
high	fd1_t0_m1_b0	0.8597 ± 0.0008	0
high	fd1_t0_m1_b1	0.8581 ± 0.0006	0
high	fd1_t1_m0_b0	0.8588 ± 0.0000	0
high	fd1_t1_m0_b1	0.8575 ± 0.0025	0
high	fd1_t1_m1_b0	0.8601 ± 0.0023	0
high	fd1_t1_m1_b1	0.8600 ± 0.0013	0
high	fd5_t0_m0_b0	0.8587 ± 0.0044	2
high	fd5_t0_m0_b1	0.8545 ± 0.0035	0
high	fd5_t0_m1_b0	0.8566 ± 0.0005	0
high	fd5_t0_m1_b1	0.8512 ± 0.0069	0
high	fd5_t1_m0_b0	0.8608 ± 0.0017	0
high	fd5_t1_m0_b1	0.8546 ± 0.0008	0
high	fd5_t1_m1_b0	0.8585 ± 0.0038	0
high	fd5_t1_m1_b1	0.8578 ± 0.0045	3

Table A 6: LLM optimizer vs. random search baseline across benchmarks.

Benchmark	Metric	Random	LLM (Best / Worst)	Δ
NOMAD	RMSLE↓	0.0847 ± 0.002	**0.0814** / 0.1267	−3.9%
Housing	RMSE↓	0.6095 ± 0.016	**0.5903** / 0.8795	−3.1%
Forest	Accuracy↑	0.8245 ± 0.012	**0.8625** / 0.8333	+4.6%

A.6 Random Search Baseline Comparison

Table A 6 provides a compact comparison between LLM-driven optimization and a random search baseline. Random search samples configurations uniformly from the parameter space without access to task context or optimization history.

Results are aggregated across initialization strata. Random search reports mean ± standard deviation over three seeds, while the LLM columns show the best and worst performing context policies (mean over three seeds).

The results show that LLM-based optimization is competitive with random search across tasks and can outperform it when appropriate context policies are used. However, poorly configured context policies may underperform random exploration, highlighting the importance of carefully designing the information presented to the agent.

B Landscape Characterization

B.1 Stratified Initialization Boundaries

The optimization trajectories of language model agents are highly sensitive to their starting positions. To systematically investigate this sensitivity, we drew upon methods in hyperparameter optimization benchmarking Pineda-Arango et al. (2021); Eggensperger et al. (2021); Pfisterer et al. (2022) to classify the evaluated Sobol pool into distinct performance strata. Because raw metric values, distributions, and optimization directions (i.e., higher-is-better versus lower-is-better) differ significantly across our four benchmarks, we mapped all objective scores to a unified **Normalized Regret** scale $r(\theta) \in [0,1]$, following established conventions for comparing optimizer performance across heterogeneous tasks Pineda-Arango et al. (2021); Pfisterer et al. (2022).

For metrics where higher scores indicate better performance (e.g., Accuracy, Mean AUC), normalized regret is computed as:

$$r(\theta) = \frac{f_{\max} - f(\theta)}{f_{\max} - f_{\min}}$$

For metrics where lower scores indicate better performance (e.g., RMSE, RMSLE), the formula is inverted:

$$r(\theta) = \frac{f(\theta) - f_{\min}}{f_{\max} - f_{\min}}$$

Under this unified transformation, a normalized regret of $r = 0.0$ always corresponds to the absolute best-performing configuration observed in the initial $n = 256$ Sobol pool, while $r = 1.0$ corresponds to the absolute worst performance.

Strata Definition and Guard Bands. Using these continuous regret values, we partitioned the landscape pool into three discrete initialization conditions. This quantile-based partitioning of observed performance is a standard technique in model-based hyperparameter optimization, notably utilized in **Tree-structured Parzen Estimator (TPE)** methods to separate promising configurations from the remainder of the search space Bergstra et al. (2011). To ensure that these strata represented distinctly different levels of objective performance and were robust against local noise or minor evaluation variance, we employed strict inclusion thresholds separated by exclusion *guard bands*. The exact definitions are as follows:

- **High** initialization: top 20% of the initial pool, $r \leq 0.20$.
- **Neutral** initialization: median band, $0.45 \leq r \leq 0.55$.
- **Low** initialization: bottom 20% of the initial pool, $r \geq 0.80$.

Configurations falling into the intermediate regions ($0.20 < r < 0.45$ and $0.55 < r < 0.80$) functioned as guard bands Eggensperger et al. (2021). These borderline configurations were entirely discarded from the selection process to prevent any overlap in relative performance quality between the independent experimental conditions Pineda-Arango et al. (2021).

Within each valid stratum, we selected the configuration exhibiting the median regret value to act as the final representative starting point θ_0 for Phase 2 optimization Bergstra et al. (2011). This median selection prevented our study from initializing at anomalous topological outliers while accurately representing the general objective behavior of that performance class Pfisterer et al. (2022).

B.2 Robustness to Sampling Stochasticity

In our primary evaluation protocol, the language model agent operates under deterministic decoding (`temperature = 0`). This experimental control serves to isolate the behavioral effects generated directly by variations in the context policy Π_{ctx}. By eliminating the confounding variance introduced by stochastic token sampling, any behavioral divergence observed across identical random seeds can be securely attributed to the differences in

agent-visible information Wu et al. (2024).

However, because deployed LLM agents frequently employ non-zero generation temperatures to encourage search space exploration, we evaluated whether our macroscopic findings were brittle artifacts of greedy decoding. This is particularly relevant given recent findings that prompt repetition and sampling diversity can significantly alter model performance in non-reasoning tasks Leviathan, Kalman and Matias (2025). To evaluate the robustness of our results against sampling stochasticity, we repeated the configuration grid experiments using models operating at `temperature` $\in \{0.2, 0.5\}$.

Across these non-deterministic evaluations, the core context axis trends observed in the primary experiments remained qualitatively similar. Most notably:

1. **Initialization Dependence:** The strong dependence on initialization quality persisted. The LLM agent consistently reached large objective improvements when instantiated within a *low* (poor) performance stratum but struggled to refine configurations initialized in the *high* stratum, occasionally degrading them. This reinforces the view that LLM-based optimizers behave primarily as corrective heuristics rather than fine-grained search engines Liu, Gao and Li (2025).
2. **Context Sensitivity:** The destabilizing effects of unstructured semantic exposure—where providing task and metric descriptions without explicit boundaries increased constraint violations—continued to manifest under non-zero temperatures Guan et al. (2025).

These supplementary findings confirm that the observed optimization dynamics are a fundamental characteristic of how the LLM interacts with exposed information, rather than a brittle phenomenon unique to deterministic decoding.

C System Infrastructure

C.1 Context-Trace Separation

A critical requirement of ContextEval is ensuring that behavioral changes in the language model agent are driven exclusively by controlled variations in the parameterized context policy Π_{ctx}, rather than arbitrary changes in prompt formatting, input parsing, or metadata leakage. To guarantee this, the system architecture enforces a strict, programmatic boundary between two distinct informational layers:

1. **The Trace Layer**: Overseen by a `RunLogger`, this layer has full observational access to the environment. It captures complete configuration dictionaries, comprehensive metric outcomes, token utilization, and latency signals.
2. **The Context Layer**: Overseen by a centralized `ContextBuilder`, this layer gen-

erates the localized, partial observation states exposed to the language model per iteration t.

To maintain rigorous isolation, the agent's observation state is strictly materialized as an immutable `ContextBundle` object. The `ContextBuilder` utilizes the context policy axes (`show_task`, `show_metric`, `show_bounds`, `feedback_depth`) as deterministic gates to selectively pull information from the Trace Layer into the Context Layer. This design reflects the principle that observation space adaptation is a primary driver of agentic success, often independent of the underlying model's reasoning capacity Yang (2024).

For instance, although the Trace Layer records the full evaluation metrics dictionary, the `ContextBuilder` applies a fixed `score_extractor` operator that reduces this to a single, context-invariant scalar score. This ensures that exposing full training metrics does not implicitly inject unintended hints about objective directionality. Such strict filtering is essential given the documented sensitivity of LLMs to minor prompt variations and artifacts Wu et al. (2024); Guan et al. (2025).

Furthermore, we employ aggressive structural validation on every instantiated `ContextBundle`. Before an API call is made, the bundle undergoes recursive key-level validation against a strict `TRACE_ONLY_FIELDS` blocklist. If any observability keys—such as `token_usage` or `latency_sec`—are detected within the agent-visible dictionary, the system raises a `ContextLeakageError`. This preventative measure aligns with the rigorous isolation standards required for reliable agent benchmarking to prevent shortcut learning or data contamination Chan et al. (2025); Huang et al. (2024).

C.2 Error Handling and Fallback Logic

Because ContextEval relies on large language models to generate executable code directly from natural language prompts, the orchestration architecture must be robust to parsing failures and output hallucinations. The `BaseBenchmark` enforcing the evaluation loop requires strict JSON formatting from the language model agent. This design follows the "evaluate-and-propose" pattern established in recent LLM-driven optimization frameworks, which necessitates high-fidelity structural outputs to interface with black-box environments Liu, Gao and Li (2025); Agarwal et al. (2025).

In the rare event that the agent produces an unparseable response, a malformed JSON object, or fails to include required keys, the system does not crash or explicitly prompt the model for a correction. We avoid dynamic error correction—common in conversational agents—because it would constitute an uncontrolled variation in context exposure, violating our experimental controls. Instead, the system logs an error event to the Trace Layer and applies a deterministic, heuristic fallback.

The `fallback_config()` mechanism generates a safe, proximate configuration by apply-

ing alternating perturbations to the most recently evaluated configuration. For continuous scales (e.g., learning rates and regularization), the fallback logic applies a ±15% multiplicative factor based on the parity of the current optimization step:

$$\theta_{t,\text{fallback}} = \begin{cases} 1.15 \cdot \theta_{t-1} & \text{if } t \bmod 2 = 0 \\ 0.85 \cdot \theta_{t-1} & \text{otherwise} \end{cases}$$

For integer parameters, small, scale-appropriate additive steps are applied, combined with the identical step-parity logic. Crucially, all perturbed fallback values are immediately passed through the clamping function to guarantee structural validity. Across our primary experiments, this fallback mechanism was nearly never invoked, confirming the format-adherence capabilities of the baseline models under our prompt structure, consistent with observations in other benchmark studies Chan et al. (2025).

C.3 Hallucinations and Structural Degradation

While parsing failures were rare in our finalized factorial grid, preliminary testing on Kaggle tasks revealed significant "context rot"—a form of structural degradation occurring when the agent is exposed to dense, unfiltered metadata. This phenomenon aligns with the "Lost in the Middle" effect, where large language models struggle to access and utilize information located in the center of long input contexts, even in models explicitly designed for high-token windows Liu et al. (2023). In our case, exhaustive competition schemas effectively "buried" the structural JSON requirements, causing the model to prioritize task-specific reasoning at the expense of schema adherence.

This degradation necessitated the strict `ContextBuilder` isolation described in Appendix C.1. By filtering the observation space to only essential axes, we ensured that critical instructions remained at the "primacy" or "recency" positions of the prompt, where performance is empirically highest Liu et al. (2023). However, even with stable formatting, we observed "objective-scale hallucinations" where the agent proposed values far outside the feasible bounds Ω.

To ensure environment safety, we pass every raw proposal $\hat{\theta}_t$ through a `sanitize_with_clamp_tracking` module:

$$\theta_t = \max(\theta_{\min}, \min(\theta_{\max}, \hat{\theta}_t))$$

If the proposal requires truncation, the system records a *clamp event* to the Trace Layer. We observed that these events were effectively eliminated only after we transitioned from restricted local ranges to the fixed, wide global bounds described in Section A 2, and enabled the `show_bounds` axis within the isolated `ContextBuilder` protocol. This confirms

that constraint violations are often a byproduct of the model's inability to robustly retrieve information from the middle of long contexts (architectural rot) rather than a fundamental lack of reasoning capacity Liu et al. (2023); Yang (2024).