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

Large language models (LLMs) are increasingly used as agents for iterative machine learning (ML) experimentation, yet little is understood about how different types of contextual information influence their optimization behavior. We introduce ContextEval, a controlled framework for isolating the causal effects of context visibility on LLM-driven hyperparameter optimization. Our system separates execution traces from agent-visible context, enabling manipulation of which signals—task description, evaluation metric, parameter bounds, and optimization history—are exposed to the agent. We evaluate these context axes on real ML benchmarks and measure their impact on optimization trajectory, convergence behavior, and constraint compliance. [WIP: Insert number of benchmarks, seeds, and total runs.] Preliminary results indicate that explicit constraint visibility (parameter bounds) eliminates invalid proposals but can alter exploration dynamics. [WIP: Insert quantitative clamp reduction statistics.] We also observe that longer feedback histories reduce constraint violations even without explicit bound information. [WIP: Insert quantitative comparison between feedback depth settings.] In contrast, verbose metric descriptions provide limited additional benefit when scalar score feedback is already available. [WIP: Insert effect size for metric axis.] These findings suggest that the structure and visibility of context—not merely model capability—significantly shape LLM optimization dynamics. [WIP: Insert strongest quantitative headline result here.] ContextEval provides a reproducible testbed for studying these effects and offers practical guidance for designing more reliable LLM-based ML agents.

Website: Placeholder
Code: https://github.com/juliamsjung/context-eval

1 Introduction

1.1 Motivation

Large language models (LLMs) are increasingly deployed as autonomous agents in machine learning workflows, where they iteratively propose model configurations, interpret evaluation results, and refine decisions over multiple optimization steps. In these settings, agent behavior is shaped not only by the underlying model, but also by the information exposed to the agent through its *context*—such as task descriptions, metric definitions, iteration history, and resource usage. Despite their importance, these context design choices are typically made heuristically and vary widely across systems. t This lack of systematic treatment matters for three reasons. First, more context is not always better: irrelevant or noisy information can distract or mislead LLM agents. Second, differences in context visibility are often confounded with model capability, making it difficult to attribute performance gains to reasoning ability rather than information access. Finally, without explicit context specifications, results from different agents or studies are not directly comparable or reproducible. Together, these issues motivate treating context visibility as a first-class experimental variable and studying its effects through controlled, causal experimentation. Understanding these effects is not only a matter of scientific rigor, but also has practical implications: identifying which context axes genuinely improve agent performance can inform the design of more efficient and effective LLM-based optimization systems.

While prompt engineering focuses on optimizing the wording of instructions, our work instead treats context allocation as a system-level design variable. A context policy specifies what categories of information are visible to the agent at each iteration and under what constraints. In our setting, these axes include the depth of prior feedback exposed to the agent, the granularity of task and metric descriptions, explicit resource budgets (e.g., token limits), and the availability of diagnostic signals such as failure modes and step-termination reasons. By fixing the base model and offline environment and varying only these context dimensions, we isolate the causal effect of information exposure on outcome (best metric), efficiency (steps/tokens), and stability (variance and oscillation across runs). Rather than tuning phrasing, we study how structured differences in epistemic access shape agent behavior over full experimentation trajectories.

1.2 Research questions

Our overarching research question is:

How does context visibility affect the performance, behavior, and stability of LLM agents in iterative machine learning optimization?

In our framework, context visibility is operationalized as a set of independently controllable axes, including:

(i) semantic context (task and metric description granularity), (ii) temporal context (feedback depth and iteration history), (iii) resource constraints (e.g., token budgets), and (iv)

diagnostic signals (e.g., failure modes and step-termination reasons).

We investigate the following sub-questions:

- **RQ1: Does context visibility causally affect outcome, efficiency, and stability?** We evaluate whether exposing different types of context produces statistically significant differences in (a) final optimization outcome (best metric), (b) convergence efficiency (steps and tokens required to improve over baseline), and (c) stability (variance and oscillation across runs), under fixed tasks, models, and optimization horizons.
- **RQ2: Which context axes contribute most to agent behavior?** We quantify the relative impact of individual context axes—task description, metric description, feedback depth, resource constraints, and diagnostic visibility—to determine which forms of information most strongly influence optimization trajectories.
- **RQ3: Are there interaction effects between context axes?** We analyze whether combinations of context axes produce non-additive effects, such as whether temporal feedback becomes more informative when paired with precise metric descriptions, or whether diagnostic signals alter the usefulness of resource constraints.
- **RQ4: Is the relationship between context exposure and optimization behavior monotonic?** We investigate whether increasing context exposure consistently improves outcome and efficiency, or whether diminishing returns, instability, or degradation emerge as additional context is introduced.

2 Related Works

2.1 LLM-Based Agents for Machine Learning Experimentation

Recent work has explored the use of large language models as autonomous agents for machine learning experimentation, including tasks such as hyperparameter tuning, model selection, and iterative refinement of training configurations. These systems often evaluate agent performance in terms of final model quality, convergence speed, or tool-use effectiveness, with an emphasis on improving agent capability or reasoning strategies. In most cases, however, the information exposed to the agent (task descriptions, evaluation metrics, or intermediate results) is treated as fixed or implicitly defined. As a result, differences in agent behavior are typically attributed to model architecture or prompting techniques rather than to variations in information access. We address this gap by explicitly treating context visibility as a controllable experimental variable within otherwise fixed optimization workflows.

2.2 Prompting, Memory, and Context in LLM Agents

A growing body of research studies how prompting strategies, memory mechanisms, and self-reflection influence LLM agent behavior. This includes work on prompt engineering, multi-step reasoning, and the use of historical interaction logs or summaries to provide

agents with longer-term memory. While these approaches demonstrate that additional information can improve agent performance in some settings, changes in context are often bundled with modifications to prompt structure, reasoning format, or agent architecture. Consequently, the causal role of context visibility itself is difficult to isolate. Our framework contributes to this space by separating agent-visible context from prompt and model changes, enabling controlled analysis of how different forms of context independently affect optimization behavior in machine learning.

2.3 Agent Benchmarks and Evaluation Frameworks

Several benchmarks and evaluation frameworks have been proposed to standardize the assessment of LLM-based agents across tasks and domains. These frameworks typically fix datasets, evaluation metrics, and interaction protocols to enable reproducible comparisons between agents. However, the information made available to agents (task descriptions, metric definitions, or access to prior iterations) is often underspecified or varies across implementations. As a result, context policies may differ even when agents are evaluated on the same benchmark, complicating comparisons across studies. We complement this line of work by explicitly controlling and reporting context visibility as part of the experimental condition, enabling more precise analysis of agent behavior under different information constraints.

3 Methodology

3.1 Problem Setting

We study large language model (LLM) agents operating in iterative, offline optimization environments. At each iteration t, the agent proposes a configuration θ_t (e.g., hyperparameters), which is evaluated by a fixed training and scoring pipeline. The agent does not observe the dataset, training procedure, or evaluation process directly. Instead, it receives a serialized context constructed from prior execution traces.

Our goal is not to optimize task performance per se, but to evaluate how different *context policies*—rules governing what information is exposed to the agent—affect agent behavior, stability, and execution characteristics under otherwise identical conditions.

Each benchmark defines a deterministic offline environment consisting of a fixed dataset D, a training procedure $\mathscr{T}(\cdot)$, and an evaluation function $\mathscr{E}(\cdot)$ producing a scalar score. Given a configuration θ_t at iteration t, the environment returns

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

The environment is stateless across iterations; all stateful behavior arises from information recorded in execution traces and selectively exposed to the agent.

The agent is a pretrained LLM with fixed parameters, operating as a conditional policy

$$\theta_{t+1} \sim \pi(\cdot \mid \mathscr{C}_t),$$

where $\mathscr{C}_t$ denotes the agent-visible context constructed externally. The agent parameters are never updated: no gradient-based learning, reinforcement learning, or policy adaptation occurs. All adaptation arises solely from changes in the input context.

3.2 ContextEval Framework

ContextEval employs a modular controller architecture that explicitly separates *observability* from *agent-visible context*. The system consists of an offline environment, a controller that manages execution and logging, and an LLM agent that proposes configurations based only on policy-gated context. All benchmarks share identical control flow and execution logic; differences in agent behavior arise exclusively from controlled changes in context visibility.

Algorithm 1 formalizes the ContextEval optimization loop, illustrating how policy-gated context projection, agent proposals, and trace logging interact within the fixed controller architecture.

Algorithm 1: ContextEval Optimization Loop with Policy-Gated Context

1. **Initialization:** Given initial configuration θ_0, environment $\mathscr{E}$, LLM agent π, context policy Π_{ctx} parameterized by axes α, and number of iterations T, initialize an empty trace history $\mathscr{H}$ and trace logger $\mathscr{L}$.
2. **Baseline evaluation:** Evaluate θ_0 to obtain s_0, log (θ_0, s_0), and append $(0, \theta_0, s_0)$ to $\mathscr{H}$.
3. **Iterative optimization:** For $t = 1$ to T:
   (a) Construct agent-visible context via policy-gated projection:
   $$\mathscr{C}_t \leftarrow \Pi_{\text{ctx}}(\mathscr{H}; \alpha).$$
   (b) Sample a proposed configuration $\hat{\theta}_t \sim \pi(\mathscr{C}_t)$.
   (c) Sanitize the proposal to obtain a valid configuration θ_t.
   (d) Evaluate θ_t to obtain score s_t.
   (e) Log (θ_t, s_t) and append (t, θ_t, s_t) to $\mathscr{H}$.
4. **Termination:** End the run and finalize the trace log.

All execution artifacts are recorded in a *trace layer*, which serves as the system's ground-truth record for analysis and reproducibility. The trace layer logs full configuration parameters, complete metric dictionaries, token usage, latency, proposal source, random seeds, and run metadata at every iteration. The trace layer is always enabled and is never exposed to the agent.

Agent-visible context is constructed independently via a centralized `ContextBuilder`, which applies an explicit context policy to project a subset of the trace history into agent-visible

form:

$$\mathscr{C}_t = \Pi_{\text{ctx}}(\mathscr{H}_{\leq t}).$$

The default (baseline) context is

$$\mathscr{C}_t^{(0)} = \{\theta_t,\, s_t\},$$

which is stateless, semantics-blind, incentive-neutral, and invariant across benchmarks.

Context Integrity Guarantees. ContextEval enforces strict guarantees to prevent unintended information leakage: (1) all agent-visible inputs are constructed exclusively via a centralized `ContextBuilder`; (2) context is represented as an immutable, typed `ContextBundle` with an explicit allowlist of fields; (3) structural validation rejects any trace-only fields during context construction; and (4) debug-time assertions verify that forbidden fields do not appear in serialized prompts. As a result, any observed behavioral differences across experiments can be causally attributed to explicit context policy settings rather than accidental exposure.

3.3 Context Policies

We define a *context policy* as a deterministic function

$$\Pi_{\text{ctx}} : \mathscr{H}_{\leq t} \to \mathscr{C}_t,$$

which maps the execution trace history up to iteration t to the agent-visible context provided at decision time. A context policy controls *only* information exposure: it does not modify the environment, agent parameters, training procedure, or evaluation function. All components of the system other than agent-visible context are held fixed across experiments.

Context policies are parameterized by a set of orthogonal *context axes*, each governing a distinct category of information exposure:

$$\Pi_{\text{ctx}} = \big(\alpha_{\text{temp}}, \alpha_{\text{task}}, \alpha_{\text{metric}}\big).$$

Temporal axis. The temporal axis $\alpha_{\text{temp}} = N$ specifies a history window of size N, exposing the last N configuration–score pairs $\{(\theta_{t-k}, s_{t-k})\}_{k=1}^{N}$. When $N = 0$, the agent is stateless.

Task semantic axis. The task semantic axis $\alpha_{\text{task}} \in \{0, 1\}$ controls whether a natural-language description of the optimization task is exposed.

Metric semantic axis. The metric semantic axis $\alpha_{\text{metric}} \in \{0, 1\}$ controls whether a textual description of the evaluation metric is exposed.

Unless otherwise stated, experiments use the *default context policy* $\Pi_{\text{ctx}}^{\text{default}} = (0, 0, 0)$, yielding an agent-visible context $\mathscr{C}_t = \{\theta_{t-1}, s_{t-1}\}$.

3.4 Experimental Setup

We evaluate ContextEval across three benchmark tasks: (i) a toy synthetic logistic regression benchmark, (ii) the NOMAD materials science regression benchmark, and (iii) the Jigsaw toxic comment classification benchmark.

We perform controlled ablations by varying context policy axes while holding the environment, agent, model, temperature, optimization horizon, and response budget fixed. Each experimental condition corresponds to a unique context policy and is logged with explicit metadata tags to enable trace-driven analysis.

Beyond final task performance, we analyze execution and behavior-level properties, including trajectory stability, sensitivity to semantic cues, temporal feedback utilization, and execution pressure as measured by token usage and termination behavior.

4 Results

4.1 Main findings

4.2 Context axis ablations

4.3 Interaction effects between axes

4.4 Cost-performance tradeoffs

5 Analysis & Discussion

5.1 When does more context help/hurt?

5.2 Task-dependent patterns

5.3 Implications for agent design

5.4 Limitations

6 Conclusion

6.1 Summary

6.2 Future work

7 Contributions

We have made major progress across the core components outlined in our proposal.

Firstly, we established a structured system architecture, defining the trace and context layers, where the trace layer is dedicated to observability and experiment logging, while the context layer controls the projection of information shown to the LLM agent. This structure allows us to manipulate contextual signals without contaminating experimental logging, causing context leaks. On top of this, we have added another benchmark (Jigsaw), and several new experiment flags and grids for deeper experimentation, which were two main goals outlines in our plan.

Secondly, we successfully started running experiments using the grid, and started performing analysis on the agent's behavior (located in the `experiments/` directory, which is not yet pushed to the repo). Through these experiments we have been discovering bugs in our

codebase, but this iteration of running experiments and debugging has helped refine our project. As a result, the system is now very close to be ready for large-scale experimentation and formal analysis.

Here are some brief description of what each member did:

- Hikaru: worked on established the system architecture, adding new flags and experiments grids. All contribution currently pushed to main.
- Adrian: worked on adding new benchmark, and refining codebase through running scripts and experiments, and checking behavior of results. All contribution currently pushed to main.
- Julia: ran full experiment grid for nomad and began experimentation (data loading, transformations, and plotting). Contributions currently not pushed to main, but plan to do so once results are more polished.
- Raghavan: developed evaluation metrics for analyzing behavior (variance, oscillation, instability). Contributions currently in a pull request.

References

Appendices

A.1 Prompt templates

A.2 Full experimental results tables

A.3 Hyperparameter bounds per benchmark

A.4 Previous Proposal

B Proposal

B.1 Problem

Large language model (LLM) agents are increasingly used to automate end-to-end machine learning (ML) experimentation: reading/writing files, proposing model pipelines, and iterating on experiments. Recent work (e.g., MLE-Bench, MLAgentBench) shows that these agents can reach competitive performance on realistic tasks, but also exhibit high variance across runs and extreme sensitivity to seemingly minor prompt changes.

In our Q1 work, we built an initial "ContextEval" framework for ML experimentation tasks: a fixed offline environment E (e.g., the NOMAD benchmark), a fixed LLM agent M (e.g., `gpt-4o-mini`), and a controller that iteratively proposes configurations, trains models, and logs detailed traces. What remains under-explored is the *context policy*: given a fixed E and M, how does changing what the agent sees—history length, dataset descriptions, clarifying tools—affect outcome (best metric), efficiency (steps/tokens), and stability (variance across seeds and prompts)?

B.2 Evidence of problem

Existing agent benchmarks mostly report aggregate task success rates or leaderboard scores. They implicitly bundle together many design decisions:

- prompt scaffolds (how the task is described),
- history handling (how many past configurations/metrics are shown),
- and tool/memory behavior (clarifications, retrieval, etc.).

As a result, it is difficult to answer questions such as: *Is a policy with long history actually better than a short-history one? Does allowing clarifications improve robustness, or just increase cost? Are some policies more stable across seeds than others?*

Our Q1 traces already show that:

- the same LLM agent can either steadily improve over the baseline or oscillate, depending on how much history it sees;
- small changes in prompt context (e.g., including or omitting certain past steps) can change both the chosen model family and the resulting metric trajectory;
- logs contain enough detail (per-step configs, metrics, context summaries) to quantify these behaviors, but we have not yet done so systematically.

Taken together, this motivates a focused Q2 effort: *treat context policies as first-class objects, and evaluate them under controlled conditions* on ML experimentation tasks.

B.3 Data

Our data consists of both:

1. **Offline ML environments.**
   - **NomadEnv:** a tabular regression benchmark derived from the NOMAD 2018 Kaggle competition, with `features.npy`, `targets.npy`, `dataset_context.json`, and a training script that trains gradient-boosted models and reports MAE and related metrics.
   - **Few more environments planned for Q2...**

   These environments are and will be strictly offline: given a configuration, the training and scoring are deterministic up to fixed seeds.
2. **Interaction traces (JSONL).** For each run of the agent on a task, we log a JSONL trace under `traces/` with events such as:
   - `run.start` / `run.end`: run metadata (task, policy type, reasoning mode, seed).
   - `op.config_proposal`: the model family and hyperparameters proposed by the LLM.
   - `op.train`: calls to `train.py` with configuration hashes, durations, and metrics.
   - `agent.iteration`: inner-loop behavior in `agentic` mode (prompts, tool calls,

clarifier questions/answers).

- `step.summary`: per-iteration summary with current and best metrics, and basic context statistics.

We treat these traces as our primary dataset: each run is an episode, and each event is a state–action–feedback record we can analyze.

B.4 Approach (Q2 Plan)

In Q1, we implemented the basic ContextEval loop and logging. In Q2, we will shift from infrastructure-building to *running full end-to-end ML experiments under controlled variations of context policies* and systematically analyzing their effects. Our plan has three main components:

1. Curated experiment grid. We will define a small but meaningful grid of settings for full ML experimentation episodes:

- **Context policies:** `short_context` vs. `long_context` (different retrieval budgets for history, dataset descriptions, and intermediate summaries), as well as a third, newly designed policy (e.g., one that provides minimal guidance and relies more heavily on agent clarifications).
- **Reasoning mode:** primarily `controller` mode (one LLM call per iteration), with selective use of `agentic` mode to study how clarifying questions and tool usage emerge during full ML experiment runs.
- **Seeds:** multiple seeds per configuration (e.g., 3–5) to estimate variance and sensitivity to initialization.

Rather than over-investing in automation, we will manually maintain an experiment sheet (CSV) mapping each `run_id` to its task, context policy, reasoning mode, seed, and final best metric. This enables a clear and interpretable experimental setup while maintaining full reproducibility.

2. Metric design and analysis. Using the JSONL traces, we will compute:

- **Outcome:** best-achieved validation metric per run, and distributions per policy.
- **Efficiency:** iterations (and, when available, tokens/time) needed to reach a target improvement over baseline.
- **Stability:** variance of best metrics across seeds for each policy, and qualitative sensitivity to history length and prompt changes.

We will build and refine analysis notebooks that load traces, reconstruct per-run trajectories, and produce:

- trajectory bands (mean ± variance over steps),
- policy-level summaries (bar/violin plots of best metrics),
- and 1–2 "case study" visualizations of agent behavior under different context policies.

3. Qualitative study of agentic mode. For a small number of runs, we will enable `agentic` mode and inspect `agent.iteration` events:

- How often does the agent ask clarifying questions under different context policies?
- What kinds of information does it request (metric definition, dataset size, feature descriptions)?
- Does clarification correlate with better or more stable performance in those runs?

These case studies will not be a full benchmark, but they will give us narrative and visual examples to complement the quantitative results.

B.5 Goals

By the end of Q2, our goals are:

- **A clean, reproducible set of full end-to-end ML experiment runs** conducted under multiple context policies (short, long, and one new variant) using a fixed LLM agent. These runs will span complete experimentation loops—from configuration proposal to training, evaluation, and iteration—rather than focusing on a single benchmark environment.
- **Well-defined and implemented metrics** that quantify outcome (best model performance), efficiency (iterations/tokens required to improve over baseline), and stability (variance across seeds and prompt initializations), all computed directly from JSONL traces of full experiment episodes.
- **Publication-ready figures and tables** that:
  - show how different context policies shape full-trajectory behavior and final experiment outcomes,
  - illustrate trade-offs between context length, performance, and computational cost,
  - and include at least one qualitative visualization of agentic behavior, highlighting clarifying-question dynamics within full ML runs.
- **A strong draft of a short paper or extended report** (with updated Methods and Results sections) that could be developed into a workshop submission, centered on "ContextEval: Evaluating Context Policies for LLM Agents in End-to-End ML Experimentation."

We intentionally de-prioritize heavy automation (e.g., full sweep orchestration) in Q2 in

favor of a smaller, well-understood set of runs and a clear scientific story. The Q1 infrastructure gives us a solid base; Q2 is about turning that infrastructure into interpretable, publishable insight about context policies.