AGENTIC BAYESIAN OPTIMIZATION THROUGH SURROGATE-AUGMENTED AUTORESEARCH

Anonymous authors
Paper under double-blind review

ABSTRACT

Bayesian optimization (BO) is the standard tool for sample-efficient gradient-free optimization through uncertainty-aware search driven by generic statistical priors. Richer domain priors can improve BO, but encoding them through *e.g.*, tailored kernels is difficult. LLMs can help bring these priors into the optimization, drawing them directly from natural language, code, and documentation. However, existing LLM-based BO methods either insert the LLM into a fixed role—surrogate, acquisition proxy, or configuration interface—or hand it broad control, sacrificing the systematic exploration that makes BO reliable. We introduce **agentic Bayesian optimization**: a paradigm in which an LLM agent controls the BO loop while a Bayesian backend holds the probabilistic surrogate. The agent selects evaluations and can revise the strategy during the run by tightening bounds, proposing targeted evaluations, or reframing the problem as evidence or instructions change. We instantiate this idea in Sara, a surrogate-augmented autoresearch agent, and lenz, a modular BO backend that the agent can inspect and modify through a structured interface. Across synthetic and real-world benchmarks, Sara preserves the reliability of standard BO without prior knowledge, outperforms LLM-based baselines, and uses natural-language priors to improve beyond standard BO. Furthermore, in dynamic settings, Sara can reconfigure the full optimization problem on the fly as requirements change—a capability not available in standard BO.

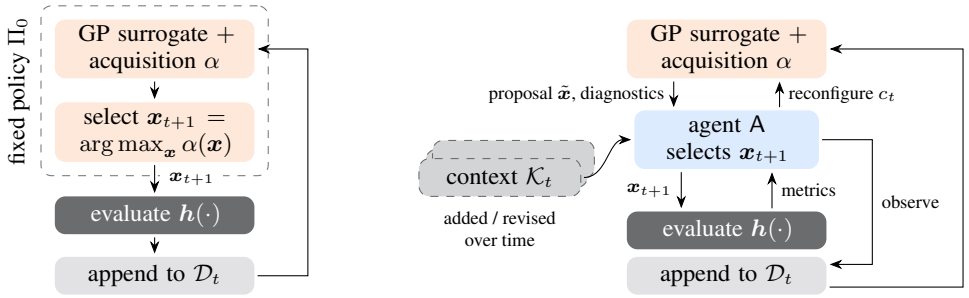
1 INTRODUCTION

Bayesian optimization (BO) is the standard tool for sample-efficient, zero-order optimization of expensive black-box functions, with applications ranging from hyperparameter tuning and robotics to experimental design (Snoek et al., 2012; Shahriari et al., 2016; Garnett, 2023) (Figure 1a). Its sample efficiency comes from uncertainty-aware decision making: a probabilistic surrogate, typically a Gaussian process (GP) (Rasmussen & Williams, 2006), models the objective, and an acquisition function uses its posterior to balance exploration and exploitation (Jones et al., 1998; Srinivas et al., 2010). However, classical BO workflows largely commit up front to the ingredients that determine the search—the surrogate, acquisition function, and search bounds. Adaptive and interactive BO methods relax this rigidity by adapting regions or incorporating user priors (Eriksson et al., 2019; Hvarfner et al., 2022; Seng et al., 2025; Fehring et al., 2025), but the adaptation remains structured and often specified in advance. A domain expert, by contrast, can revise the search strategy itself as evidence or requirements evolve, drawing on knowledge expressed in natural language, code, documentation, and past experimental logs. Classical BO cannot readily absorb such heterogeneous context. While problem-specific kernels can encode useful knowledge and accelerate search (Marco et al., 2017), deriving them in practice for a specific problem can be challenging. LLMs can consume these sources directly and reason over them as optimization unfolds, making them natural candidates for adaptive, domain-informed black-box optimization.[1]

Existing LLM-based optimization methods realize only part of this promise. LLM-as-optimizer methods and autoresearch agents offer broad control and access to semantic priors, but select evaluations without the explicit probabilistic surrogate that underpins BO's systematic exploration (Yang et al.,

[1]Throughout the paper, we use *black-box optimization* and *zero-order optimization* interchangeably to denote optimization without access to analytical gradients, not to imply a complete absence of problem knowledge.

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



(a) Standard BO (b) Agentic BO

Figure 1: *Standard vs. agentic Bayesian optimization.* In Figure 1a, a fixed policy Π_0 selects the next query point. In Figure 1b, the agent sits at the center of the loop and decides on the next point—adopting, refining, or overriding the surrogate's proposal $\tilde{\boldsymbol{x}}$—while also querying surrogate diagnostics and incorporating additional natural-language context or instructions.

2024; Liu et al., 2024b;a; Lu et al., 2024). Hybrid methods retain a surrogate but assign the LLM a fixed role, such as designing the kernel, selecting an acquisition function, proposing candidates, or translating feedback (Suwandi et al., 2026; Ngo et al., 2026; Cissé et al., 2025; Kobalczyk et al., 2026). What is missing is an agent that can change *how* it optimizes—tightening bounds, switching acquisition functions, or reframing the problem mid-run—without surrendering posterior uncertainty.

We introduce **agentic Bayesian optimization**, a paradigm in which an LLM agent sits at the center of the optimization loop while delegating probabilistic modeling and acquisition optimization to a Bayesian backend (Figure 1b). Before committing each expensive evaluation, the agent may inspect the surrogate, request, refine, or override proposals, and reconfigure the optimization problem in response to new evidence or changing instructions. This division lets the agent contribute semantic priors and judgments that are difficult to encode in standard BO, while the backend provides posterior uncertainty for systematic, sample-efficient search. This design also bridges two communities. From the BO perspective, it is *agentic Bayesian optimization*: an agent controls and revises the optimization process while retaining a probabilistic backend. From the autoresearch perspective, it is *surrogate-augmented autoresearch*: a research agent delegates uncertainty-aware numerical search to an explicit surrogate and optimization engine. Following the broader tool-use principle that LLMs should delegate precise computation to specialized systems (Schick et al., 2023; Gao et al., 2023; Chen et al., 2023b), we instantiate these perspectives in Sara, a Surrogate-augmented autoresearch agent, and lenz (pronounced [lɛns], as in "lens"), a modular BoTorch-based backend designed for run-time adaptation and agentic use. In summary, our core contributions are

- *(i)* **Agentic Bayesian optimization**, a paradigm in which an agent revises the optimization policy from trial data, surrogate diagnostics, and external context.
- *(ii)* **A surrogate-augmented autoresearch system** comprising Sara, which drives the optimization, and lenz, which exposes modular BO primitives through a structured CLI.
- *(iii)* **An empirical study** showing that Sara matches standard BO without prior knowledge, outperforms LLM-based methods, and can incorporate natural-language priors and instructions.

2 Background and Problem Setting

We first introduce the class of problems we consider and then frame standard Bayesian optimization as a fixed policy defined before the campaign begins.

Problem. We aim to optimize an expensive-to-evaluate black-box function over a d-dimensional domain $\mathcal{X}$. Each evaluation yields a vector of noisy outcomes $\boldsymbol{y}_i = \boldsymbol{h}(\boldsymbol{x}_i) + \boldsymbol{\varepsilon}_i$, where $\boldsymbol{h} : \mathcal{X} \to \mathbb{R}^p$, and after t evaluations the collected data is $\mathcal{D}_t = \{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=1}^t$. A problem specification assigns components of $\boldsymbol{h}$ to objective functions f_o, indexed by $o \in \mathcal{O}$, and constraint functions $g \in \mathcal{C}$, with each constraint expressed as $g(\boldsymbol{x}) \leq 0$. This gives the constrained multi-objective problem

$$\max_{\boldsymbol{x}\in\mathcal{X}} \left(f_o(\boldsymbol{x})\right)_{o\in\mathcal{O}} \qquad \text{s.t.} \qquad g(\boldsymbol{x}) \leq 0 \quad \forall\, g \in \mathcal{C}, \tag{2.1}$$

with the maximum understood in the Pareto sense. The familiar cases are recovered by choice of $\mathcal{O}$ and $\mathcal{C}$, *i.e.* single-objective BO as $|\mathcal{O}| = 1$, $\mathcal{C} = \emptyset$, constrained BO as $\mathcal{C} \neq \emptyset$, and multi-objective BO

as $|\mathcal{O}| > 1$. Which outcomes serve as objectives and which as constraints is a modeling choice, not a property of the experiment itself.

The standard BO loop. BO fits a probabilistic surrogate to $\mathcal{D}_t$ and uses it to select the next query. Placing a GP on each outcome gives posterior predictives that can be used in an acquisition function $\alpha(\boldsymbol{x}; \mathcal{D}_t)$ to score candidates by their expected utility. It is useful to separate two steps that the standard loop conflates: the acquisition function *proposes* by ranking $\mathcal{X}$, and a selection rule *commits* one candidate as the next evaluation. Standard BO uses the $\arg\max$ rule, $\boldsymbol{x}_{t+1} = \arg\max_{\boldsymbol{x} \in \mathcal{B}} \alpha(\boldsymbol{x}; \mathcal{D}_t)$, over an active search region $\mathcal{B} \subseteq \mathcal{X}$. We keep these two steps distinct in what follows, since an agent can retain the surrogate's proposal while overriding the selection.

Policy and configuration. At step t, the model class $\mathcal{M}_t$, acquisition function α_t, active region $\mathcal{B}_t$, and problem partition $(\mathcal{O}_t, \mathcal{C}_t)$ define the **configuration** $c_t = (\mathcal{M}_t, \alpha_t, \mathcal{B}_t, \mathcal{O}_t, \mathcal{C}_t)$. A standard BO loop similar to the one described above then amounts to a fixed policy Π_0 that maps the accumulated data to the active configuration and next query,

$$(c_t, \boldsymbol{x}_{t+1}) = \Pi_0(\mathcal{D}_t), \qquad \text{with } \Pi_0 \text{ fixed before the campaign.} \tag{2.2}$$

Here, *fixed* refers to the mapping Π_0, but not its outputs, *i.e.*, the configuration may change, but only according to prescribed rules and heuristics. Modern BO frameworks (Olson et al., 2025; Akiba et al., 2019) are concrete instances of such a policy. For example, Ax switches from Sobol initialization to model-based search, but the switching rule and subsequent strategy are set before the first evaluation; most specialized BO algorithms are likewise specified by heuristics at startup.

3 Related Work

Our work sits at the intersection of BO, AutoML, LLM-driven optimization, and autonomous research agents. We organize prior work by the role that LLMs play in the optimization loop, and compare the resulting classes along three axes: access to a probabilistic surrogate, run-time policy control, and use of natural-language priors and tools (Table 1). We defer a more detailed discussion to Section A.

Bayesian optimization and AutoML. Classical BO uses a surrogate with an acquisition function (Rasmussen & Williams, 2006; Jones et al., 1998; Shahriari et al., 2016), and frameworks such as Ax and Optuna package these components into reusable policies (Olson et al., 2025; Akiba et al., 2019). Methods including TuRBO and πBO adapt regions or user priors through rules fixed before the campaign (Eriksson et al., 2019; Hvarfner et al., 2022). Interactive BO admits new priors during a run, but restricts interaction to structured priors interpreted by a fixed selection rule (Seng et al., 2025; Fehring et al., 2025). AutoML methods likewise automate search through a strategy selected in advance (Bergstra et al., 2011; Li et al., 2018; Chen et al., 2022; Tornede et al., 2024). Agentic BO retains the probabilistic model but allows for additional unstructured context to influence the search.

Hybrid LLM–BO methods. A growing line of work retains Bayesian machinery but gives the LLM one role: kernel design (Suwandi et al., 2026), learning surrogate features or low-fidelity predictions (Ranković et al., 2025; Chen et al., 2026), acquisition selection (Ngo et al., 2026), candidate proposal or filtering (Cissé et al., 2025; Yang et al., 2025; Chang et al., 2025), or preference translation (Kobalczyk et al., 2026). Other approaches optimize a latent space and use the LLM as a decoder (Agarwal et al., 2025). These methods demonstrate that LLMs and probabilistic models are complementary, but the point of intervention and surrounding policy remain fixed. Even adaptive invocations rely on prescribed triggers or switching schedules (Cissé et al., 2025; Mahammadli & Ertekin, 2024; Chang et al., 2025). Agentic BO is complementary and these components could be exposed as backend tools that an agent selects when appropriate.

LLMs as optimizers and research agents. OPRO and related methods ask an LLM to emit the next configuration from a trajectory summary (Yang et al., 2024; Zhang et al., 2023a). LLAMBO applies this pattern to BO (Liu et al., 2024b), and program-search methods use LLM-proposed edits (Romera-Paredes et al., 2024). These methods do not maintain an explicit probabilistic surrogate, and LLM uncertainty is often poorly calibrated (Kadavath et al., 2022; Kristiadi et al., 2024). Centaur instead shares CMA-ES state with an LLM through its prompt, but retains a fixed division of decisions between them (Ferreira et al., 2026). Agent-based systems broaden control to hyperparameter optimization and ML workflows (Liu et al., 2024a; Madiraju & Madiraju, 2025; Zhang et al., 2023b; Huang et al., 2023), while open-ended agents orchestrate tools and scientific tasks beyond optimization (Yao et al., 2023; Lu et al., 2024; Boiko et al., 2023; M. Bran et al., 2023). They offer

Table 1: Classical BO and AutoML ground every decision in a probabilistic surrogate but run a *fixed* policy. LLM-as-optimizer methods and open-ended autoresearch agents exercise run-time control but discard posterior uncertainty. Sara retains access to a BO backend to help reason about the search.

	Calibrated surrogate	Full run-time policy control	Natural-language priors & tool use
Classical BO & AutoML (Ax, TPE, OptFormer, ...)	✓	✗	✗
Interactive BO (IBO-HPC, DynaBO, ...)	✓	✗	✗
Hybrid LLM-BO (LILO, BORA, Reasoning BO, ...)	✓	✗	✓
LLM-as-optimizer (LLAMBO, OPRO, AgentHPO, ...)	✗	✓	✓
Open-ended autoresearch agents (HyperAgents, ...)	✗	✓	✓
Agentic BO / Surrogate-augmented autoresearch (Sara + lenz)	✓	✓	✓

flexible reasoning but no calibrated posterior for fine-grained search. Sara instead gives the agent run-time control while lenz assists in uncertainty-aware search, following the principle of delegating precise computation to tools (Gao et al., 2023; Chen et al., 2023b; Schick et al., 2023).

4 Agentic Bayesian Optimization

To our knowledge, no existing approach combines three capabilities: *(i)* a probabilistic surrogate, *(ii)* strategy adaptation from accumulating evidence, and *(iii)* incorporation of unstructured domain knowledge or changing task instructions. We now introduce agentic BO as a paradigm that places an agent in control of a BO backend and specifies the information, actions, and state updates available to it. Formally, it extends the fixed policy in (2.2) by replacing the prescribed selection step with an agent-controlled *deliberation process* over the backend. Before committing an expensive evaluation, the agent may inspect the surrogate, request and refine proposals, or persistently reconfigure the optimization strategy. We model this interaction as a *metalevel decision process*, inspired by metalevel MDPs (Hay et al., 2012; Callaway et al., 2022). Throughout, *deliberation* refers to the observable sequence of computational actions and backend responses. In Section I, we complete the formulation into a full meta-MDP suitable for training a specialized policy; here, we show that a general-purpose LLM can already implement an effective policy without specialized post-training.

State. Let t index expensive evaluations and j index computational actions taken between evaluations. During deliberation, the agent observes the state $s_t^j = \left(\mathcal{D}_t,\ c_t^j,\ \mathcal{K}_t,\ \mathcal{H}_t^j\right)$, where $\mathcal{D}_t$ is the observed trial data, c_t^j is the current optimization configuration (*cf*. Section 2), $\mathcal{K}_t$ is the exogenous context, and $\mathcal{H}_t^j$ is the deliberation history available to the agent. The context contains the problem description, domain knowledge, and requirements supplied by a human or another agent; new or revised requirements may be appended during the campaign.

Action space. At each deliberation step, the agent chooses among three classes of actions, $\mathcal{A} = \mathcal{A}_{\text{comp}} \cup \mathcal{A}_{\text{eval}} \cup \{\text{stop}\}$, which differ in their effect on the state and the black-box evaluation budget:

(i) **Evaluate** ($\mathcal{A}_{\text{eval}}$). The agent commits a point $\boldsymbol{x}_{t+1} \in \mathcal{X}$ to the expensive oracle. This is the only action that advances the campaign counter t and consumes black-box evaluation budget.

(ii) **Computational** ($\mathcal{A}_{\text{comp}}$). Backend interactions in this class inform the next decision without advancing t or consuming black-box evaluation budget, although they incur computational overhead.[2] We further partition them into three main classes:

- **probe**: read-only queries to the backend—predictions, acquisition scores, diagnostics, the current Pareto front, or trial history—whose responses are added to $\mathcal{H}_t^j$;
- **reconfigure**: mutations of c_t^j, *e.g.*, tightening the search region $\mathcal{B}$, swapping the acquisition function α, or revising objectives $\mathcal{O}$ and constraints $\mathcal{C}$;
- **propose**: requests for one or more candidate points $\tilde{\boldsymbol{x}}$ from the surrogate backend via acquisition-function optimization over the current active region.

(iii) **Stop** (stop). Terminate the campaign and return the incumbent feasible solution in the single-objective case or the current feasible Pareto set in the multi-objective case.

[2]We report token usage in Section G.4; Section I shows how a full meta-MDP may account for comp. costs.

A computational action and its response are appended to the history; a reconfiguration additionally updates $c_t^j \mapsto c_t^{j+1}$. Thus, probe and propose actions can affect the eventual evaluation without mutating the configuration, whereas reconfigure persistently changes subsequent backend calls.

Metalevel policy and deliberation. The agent implements a metalevel policy $\mathsf{A} : \mathcal{S} \to \mathcal{A}$, which is applied repeatedly to the evolving deliberation state s_t^j until an evaluation or stop action is selected. A campaign step consists (*cf.* Figure 2) of $k_t \geq 0$ computational actions followed by an evaluation or termination:

$$a_t^0, \ldots, a_t^{k_t-1} \in \mathcal{A}_{\text{comp}},\ a_t^{k_t} \in \mathcal{A}_{\text{eval}} \cup \{\text{stop}\}. \quad (4.1)$$

If $a_t^{k_t} = \boldsymbol{x}_{t+1} \in \mathcal{A}_{\text{eval}}$, the oracle returns $\boldsymbol{y}_{t+1} = \boldsymbol{h}(\boldsymbol{x}_{t+1}) + \boldsymbol{\varepsilon}_{t+1}$ and the campaign advances as $\mathcal{D}_{t+1} = \mathcal{D}_t \cup \{(\boldsymbol{x}_{t+1}, \boldsymbol{y}_{t+1})\}$, and $c_{t+1}^0 = c_t^{k_t}$. New exogenous information may then be appended for $\mathcal{K}_{t+1}$, and the deliberation history carries forward to the next step. If instead $a_t^{k_t} = \text{stop}$, the campaign terminates.

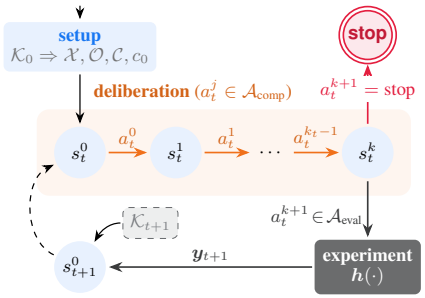


Figure 2: *One campaign step in agentic Bayesian optimization.*

We additionally distinguish an initial **setup** phase ($t = 0$) in which the agent formalizes the problem from $\mathcal{K}_0$, *i.e.*, it defines $\mathcal{X}$, partitions outcomes into $\mathcal{O}$ and $\mathcal{C}$, and initializes c_0^0. Analogous to the fixed BO policy in (2.2), we summarize the campaign step as $(c_{t+1}^0, \boldsymbol{x}_{t+1}) = \text{Rollout}(\mathsf{A}, s_t^0)$, where Rollout subsumes the response-conditioned deliberation sequence in (4.1) and may instead return stop. Ultimately, it is the *agent* that commits $\boldsymbol{x}_{t+1}$. It may obtain candidate proposals $\tilde{\boldsymbol{x}}$ from the surrogate and then adopt, refine, or override them.

This formulation includes several familiar regimes as special cases:

- *(i)* without backend interactions, the agent proposes evaluations directly, recovering LLM-as-optimizer methods (Liu et al., 2024b;a);
- *(ii)* with a fixed configuration and one surrogate proposal that is always accepted per campaign step, it recovers the standard BO loop (2.2);
- *(iii)* with probes, reconfigurations, multiple proposals, or agent overrides, it yields full agentic BO.

The interaction therefore provides two levels of control beyond the standard loop (Figure 1): at the *metalevel*, the agent chooses a response-conditioned sequence of computational actions; at the *evaluation level*, it commits $\boldsymbol{x}_{t+1}$ rather than following a fixed rule. Agentic adaptation may thus appear in probe and proposal patterns even when the persistent backend configuration remains fixed.

5 Sara and lenz as the surrogate-augmented research team

Agentic BO requires two pieces: a backend that exposes inspectable and mutable BO state, and an agent that decides which backend operations to use. We instantiate these roles with Sara and lenz (Figure 3). Sara reads the context, chooses computational actions, and commits evaluations. lenz stores observations, fits GP surrogates, and exposed BO primitives through a structured interface. At each campaign step, Sara may probe the posterior, request and assess proposals, or reconfigure the optimization problem before evaluating. Each lenz response is added to the deliberation history, so later actions can depend on what the surrogate reports; a step may contain zero, one, or many backend calls. After Sara commits a point, the experiment returns its metrics and Sara records the result in lenz. New domain knowledge or changed requirements enter through the LLM context, allowing the campaign to be redirected without discarding previous trials.

lenz is a modular backend based on GPyTorch (Gardner et al., 2018) and BoTorch (Balandat et al., 2020). It makes the raw trial log the single source of truth and allows the optimization configuration to be mutable. All surrogates are derived from retained observations when needed. A structured JSON command-line interface exposes setup, propose, probe, and reconfigure actions, together with operations that record evaluations. The architecture, surrogate and acquisition details, and representative commands and outputs are in Section L.

Sara is the reasoning agent that interprets natural-language context, frames the problem, derives priors, and deliberates with lenz before choosing which point to evaluate. Her behavior is determined

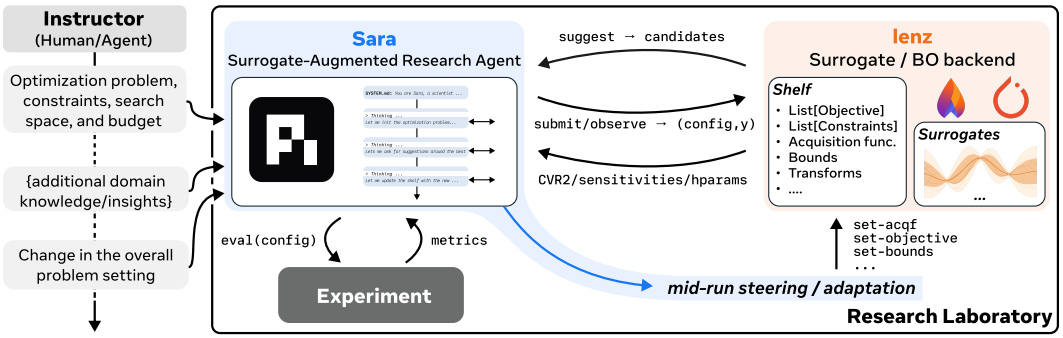


Figure 3: *Our proposed instantiation of agentic BO.* Sara reads trial data, natural-language context, queries and reconfigures the BO backend lenz, and commits the next evaluation.

by the underlying LLM, a system prompt, and the lenz documentation. The prompt instructs Sara to own the optimization decisions, use lenz for calibrated search, reflect between trials, and state her prior beliefs and intended strategy. Table 10 gives the full directive summary and rationale. The full harness setup, system prompt, and lenz reference are in Sections J and K.

6 Experiments

We evaluate Sara and lenz across settings that test standard BO reliability, the use of natural-language priors, and agentic decision making under dynamic resource constraints.

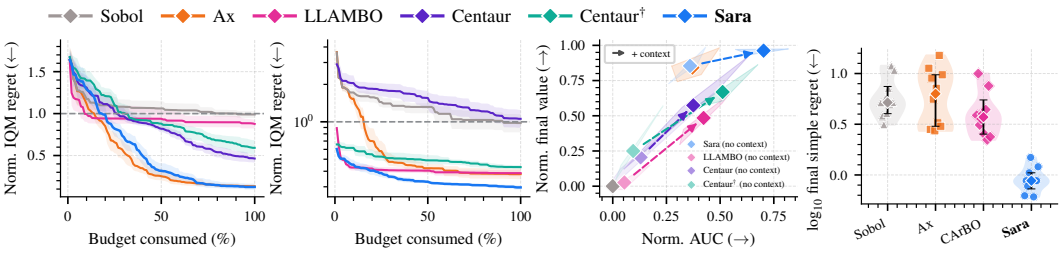
On the experimental setup. Standard test functions (Branin, Ackley, ...) and their optima are well-represented in LLM pretraining corpora. To make direct exploitation of memorized solutions more difficult, every synthetic benchmark is presented through an opaque interface with the domain normalized to the unit hypercube and the optimum shifted from its textbook location. We observed that without these measures, LLM-based methods were able to one-shot problems such as Ackley whose optimum lies at the center of the domain. Because Sara executes experiments autonomously and can inspect her working directory, we place each run in a sandbox that contains the task description and a symbolic link to the evaluation oracle. The sandbox is named by a random token such that neither the path nor the directory contents reveal which benchmark she is solving.

Baselines. We compare against four baselines (full details in Section D): **Sobol** quasi-random sampling, as a space-filling lower bound; **Ax** (Olson et al., 2025) with its default strategy, representing a well-tuned classical BO policy with best practices implemented for all problem classes; **LLAMBO** (Liu et al., 2024b), which renders the trial history as a table and queries an LLM for the next candidate directly; **Centaur** (Ferreira et al., 2026), a recent hybrid approach similar to our surrogate-augmented research agent that incorporates CMA-ES (Hansen, 2016) internal state into the context of an LLM autoresearcher. We furthermore include a variant of this algorithm **Centaur**† which always on the first evaluation invokes the LLM to fully leverage warm-starting capabilities (details in Section D). When a natural-language problem description is available, it is provided to all LLM-based methods, ensuring the comparison isolates algorithmic differences. Unless otherwise specified, all LLM-based baselines use Claude Opus 4.8 with *high* reasoning mode.

Metrics. In all experiments, we report *simple regret* (best-so-far gap to the known optimum, or raw *best value* for yield objectives) aggregated over 10 independent seeds. For cross-benchmark summary curves, we normalize within each problem and report the interquartile mean (IQM) across problem–seed pairs; Section C formalizes the normalization, aggregation, and confidence intervals we use. The main body shows aggregated performance and the full optimization curves behind these aggregates are in Section E. Section B provides the full setup details for all experiments.

6.1 Benchmark evaluation

We first evaluate three regimes spanning no-prior and prior-informed optimization, before turning to cost-aware asynchronous optimization under a wall-clock deadline. Quantitative results for the standard benchmark suite are listed in Table 2.



(a) Synthetics (*5 benchmarks*) (b) LCBench (*3 benchmarks*) (c) Yield (*4 benchmarks*) (d) JAHS-Bench-201 (*Full in Section F*)

Figure 4: *Aggregated results.* (a) Synthetics and (b) LCBench show normalized IQM regret versus consumed budget, aggregated over all benchmarks in each suite; the dashed line marks the Sobol reference. (c) summarizes each method by its normalized AUC and normalized final value (polygons show the spread of individual benchmarks). (d) shows final regret on the cost-aware and asynchronous JAHS-Bench experiment. Sara achieves strong aggregate performance across all four settings.

Synthetic problems. We first consider synthetic problems where the language model receives no domain description. The suite spans low- (Branin, 2D), moderate- (Hartmann, 6D), and high-dimensional (Ackley, 10D and 20D) unconstrained problems, together with constrained Hartmann (6D). These results are aggregated in Figure 4a, and the full optimization curves, including additional multi-objective results, are shown in Figure 10. Sara tracks Ax closely across the single-objective suite, converging slightly more slowly but reaching comparable final performance, while the LLM-in-the-loop baselines substantially underperform. Thus, when no useful domain knowledge is available, Sara can fall back on lenz's surrogate-backed search, confirming that the framework gracefully reduces to standard BO when the agent has no useful domain knowledge to contribute.

Prior-informed optimization. We next consider settings where domain knowledge is available through a natural-language problem description. On LCBench (Zimmer et al., 2021), Sara and the other LLM-based baselines receive a description of the training setup (*cf*. Section B). Figure 4b shows a clear warm-starting benefit with both Sara and LLAMBO achieve strong early performance, whereas Ax spends several initial evaluations discovering the landscape from scratch. Plain Centaur underperforms throughout, while its warm-started variant Centaur† improves markedly early on but plateaus above Sara, LLAMBO, and Ax. We observe a similar pattern on the chemistry-informed reaction benchmarks in Figure 4c. Ax converges to near-optimal yield but slowly, whereas LLAMBO and Centaur show strong warm starts but fail to refine the identified regime. Sara instead combines a prior-guided warm start with model-based refinement, reaching the strongest final performance. Full optimization curves are shown in Figures 11 and 13, and reaction benchmarks details in Section B.2.

Cost-aware AutoML. We further evaluate Sara on JAHS-Bench-201 (Bansal et al., 2022) in a mixed search space with continuous, ordered, and categorical variables, where the goal is to find the best model configuration within a fixed wall-clock deadline using asynchronous parallel evaluations. In this setting, we show how surrogate-augmented deliberation extends beyond reasoning about the search space alone. For this deliberation, we equip Sara with a surrogate over runtime through lenz and additionally provide access to the training code as structural context. This allows Sara to reason over promising model configurations, predicted runtime, worker availability, and the remaining wall-clock budget when making allocation decisions; ablating either the runtime surrogate or code access degrades performance. Sara outperforms standard and cost-aware BO under the tight budget (*cf*. Figure 4d). The complete setup, scheduling analysis, results, and ablations are in Section F.

6.2 Run-time adaptation with Sara and lenz

We next investigate Sara's tool use during a standard optimization and then for mid-run adaptation.

Adaptive tool use. Figure 5 shows the relative frequency of lenz calls over normalized trial progress on Buchwald–Hartwig, with and without the natural-language problem description. A list of all available commands is given in Section L. The frequencies are averaged across 10 seeds, which smooths the individual call sequences but still reveals clear trends in tool use. With additional context, Sara performs more agent-driven evaluations early on and quickly restricts the search space through `set-bounds` to move away from global `suggest`. Without context, global surrogate proposals

remain prominent much longer, while local refinement emerges gradually as observations accumulate. In both settings, `diagnostics`, `incumbent`, `status` are used to both monitor the model and the optimization process. Thus, natural-language context affects not only the initial configurations and resulting warm start but also the agent's pattern of interaction with the surrogate throughout the run. Some full per-seed tool-use timelines, which expose the variation hidden by this average, are in Section G.3.

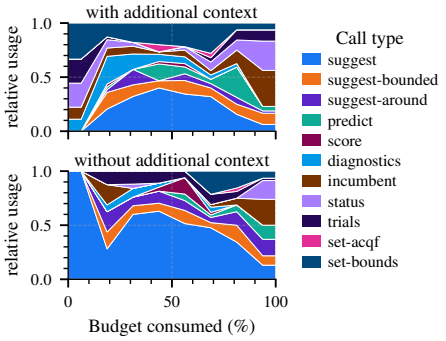


Figure 5: *Tool use over an optimization campaign.* Relative frequency of lenz calls on Buchwald–Hartwig with and without natural-language context.

Mid-run problem reformulation. Beyond adapting its tool use within a fixed problem, Sara can reconfigure the optimization specification when the requirements themselves change. We demonstrate this capability on a synthetic scaling-law study in which Sara initially minimizes training loss subject to a hard compute constraint (*cf*. Section B.3 and Figure 6). Partway through the campaign, the instructor changes the task from finding the best model to mapping the full loss–compute trade-off within new constraints. In response, Sara promotes compute from a constraint to a second objective, revises the remaining constraints, and switches to a hypervolume-based acquisition function. Because lenz maintains the raw trial log as its single source of truth, all evaluations from the constrained phase remain valid observations under the new specification. The reformulation is carried out directly from the natural-language instruction. This example illustrates how agentic BO can adapt not only its search strategy but also the problem being optimized as requirements evolve.

6.3 Ablations

We ablate the key components of agentic BO and our instantiation in Sara and lenz, summarizing the findings here and deferring full experiments and analysis to the appendix in Section G.

A1 *Value of a surrogate backend.* We compare the full system against the same agent with `bash` only, with lenz's computational actions hidden, and against an evaluate-only variant (Figure 18). On synthetics, the full system is more reliable, while the `bash`-only agent sometimes recognizes the benchmark from its observed structure and exploits this knowledge. We therefore also test random GP sample paths absent from pretraining data, where the surrogate benefit is clearer; the multi-objective results in Figure 10f show the same trend. On JAHS-Bench-201, ablating the runtime surrogate likewise degrades performance, showing its value for allocation under a wall-clock budget. The `bash`-only agent nevertheless adapts capably, using space-filling search, coordinate descent, local quadratic surrogates, and even custom BO loops; see Section G.1.

A2 *Model capability and reasoning level.* Fixing the lenz backend and instructions and varying only the LLM (Opus 4.8, Sonnet 4.6, Haiku 4.5), we find that sensitivity is task-dependent (Figure 20): Haiku underperforms on Ackley (10-D), where performance depends strongly on effective use of lenz, but the gap narrows on prior-informed tasks. We further ablate the reasoning level and find little effect on standard single-objective optimization, whereas in the more complex JAHS-Bench-201 setting higher reasoning effort improves performance.

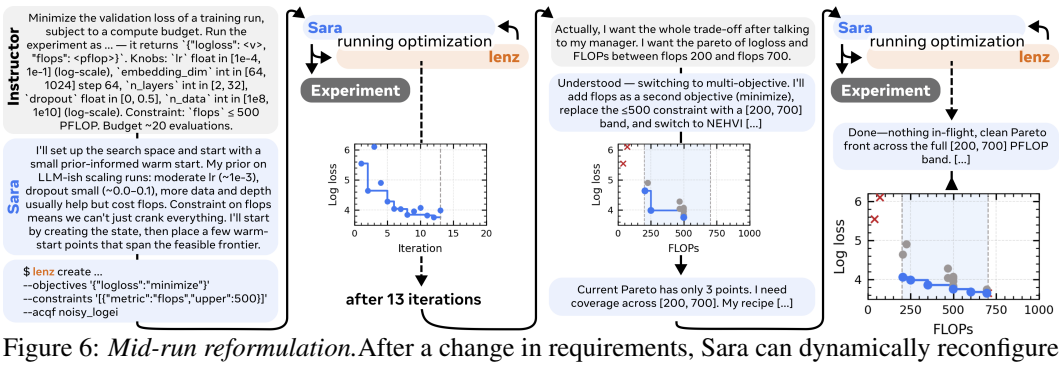


Figure 6: *Mid-run reformulation.* After a change in requirements, Sara can dynamically reconfigure the problem without restarting the campaign. *(Actual data and excerpt from a trajectory)*

Table 2: Final regret / best value ($\text{median}^{q25}_{q75}$ over seeds). Lower is better. Best median in each row and methods not significantly different under a two-sided Mann–Whitney test ($p \geq 0.05$) are in **bold**.

	Problem	Sobol	Ax	LLAMBO	Centaur	Centaur†	Sara (ours)
Shifted synthetics	Branin	$1.2241^{1.9899}_{0.9120}$	$\mathbf{0.0006}^{0.0010}_{0.0005}$	$0.6140^{6.5453}_{0.2023}$	$0.2328^{0.3492}_{0.0471}$	$0.1308^{0.2513}_{0.0101}$	$\mathbf{0.0010}^{0.0059}_{0.0002}$
	Hartmann-6	$1.2235^{1.5718}_{1.0220}$	$\mathbf{0.0116}^{0.0435}_{0.0031}$	$1.0756^{1.7324}_{0.9110}$	$0.2848^{0.4196}_{0.1779}$	$0.3646^{0.5690}_{0.3089}$	$\mathbf{0.0040}^{0.0995}_{0.0009}$
	C-Hartmann	$2.2844^{2.5587}_{1.6398}$	$0.0118^{0.0759}_{0.0022}$	$3.0216^{3.3222}_{1.9934}$	–	–	$\mathbf{0.0010}^{0.0124}_{0.0006}$
	Ackley-10	$19.3208^{19.6592}_{19.0833}$	$4.3349^{4.5010}_{4.1948}$	$17.3726^{18.8270}_{16.5854}$	$11.7241^{13.1286}_{10.7962}$	$11.3270^{12.2706}_{8.5242}$	$\mathbf{3.5700}^{4.0177}_{3.4063}$
	Ackley-20	$20.2772^{20.5228}_{20.0543}$	$\mathbf{6.5389}^{11.9659}_{5.3032}$	$17.7163^{18.7241}_{17.2203}$	$15.0561^{15.5771}_{13.9672}$	$13.8722^{15.7696}_{12.4000}$	$\mathbf{6.8023}^{8.2227}_{5.0166}$
GP-sample paths	GP-sample 4-D	$\mathbf{-2.1971}^{-1.9307}_{-2.4791}$	$\mathbf{-2.8276}^{-2.2426}_{-2.9622}$	$-1.4010^{-0.8925}_{-1.9966}$	$-2.0853^{-2.0351}_{-2.4079}$	$-1.9348^{-1.7686}_{-2.1981}$	$\mathbf{-2.5357}^{-2.1413}_{-2.8568}$
	GP-sample 8-D	$-2.4400^{-2.2687}_{-2.5272}$	$\mathbf{-4.6380}^{-3.9291}_{-4.7556}$	$-1.4777^{-1.3651}_{-2.0001}$	$-3.5408^{-2.7540}_{-3.6391}$	$-2.7053^{-2.4693}_{-3.0986}$	$\mathbf{-4.0782}^{-3.6559}_{-4.8572}$
	GP-sample 12-D	$-2.6777^{-2.5629}_{-3.2180}$	$\mathbf{-4.9569}^{-4.7722}_{-5.6020}$	$-1.9106^{-1.6844}_{-2.5519}$	$-3.5094^{-3.2633}_{-3.9814}$	$-3.2664^{-2.5350}_{-3.8738}$	$\mathbf{-5.4472}^{-4.6927}_{-5.6760}$
	GP-sample 16-D	$-3.0236^{-2.7044}_{-3.2078}$	$\mathbf{-5.5091}^{-5.2064}_{-5.9126}$	$-1.7866^{-1.4595}_{-1.9566}$	$-3.4865^{-3.2441}_{-3.6891}$	$-3.7974^{-3.3079}_{-4.1424}$	$\mathbf{-5.6648}^{-5.2669}_{-6.0926}$
Prior-informed benchmarks	Suzuki	$65.6743^{74.0407}_{56.4974}$	$\mathbf{1.0256}^{1.5070}_{0.8644}$	$41.4614^{48.5536}_{24.2302}$	$19.2596^{27.4467}_{11.7587}$	$19.6900^{24.6092}_{13.8482}$	$\mathbf{0.9133}^{3.4598}_{0.3663}$
	Heck	$76.9690^{79.4275}_{67.4721}$	$11.3688^{12.3328}_{7.8195}$	$49.1530^{52.3030}_{38.3898}$	$63.9634^{73.6918}_{40.8638}$	$37.8807^{47.5237}_{34.5870}$	$\mathbf{0.8768}^{6.4854}_{0.5095}$
	Buchwald	$88.6124^{90.4188}_{85.5472}$	$16.3853^{29.8537}_{15.5621}$	$30.4706^{50.7704}_{12.1470}$	$38.3680^{56.4602}_{17.3170}$	$38.6513^{48.3442}_{25.3535}$	$\mathbf{1.8539}^{2.0214}_{0.2483}$
	Grignard	$79.9999^{80.0000}_{79.8885}$	$13.1945^{13.6690}_{8.6096}$	$27.5503^{41.0600}_{15.1578}$	$5.1943^{8.3346}_{4.4677}$	$3.9533^{4.4757}_{3.0191}$	$\mathbf{0.4565}^{1.9252}_{0.2947}$
	LCBench Airlines	$2.2888^{3.7445}_{2.0277}$	$1.3032^{1.3312}_{1.2732}$	$1.2149^{1.2505}_{1.1073}$	$2.7764^{4.8164}_{2.0743}$	$1.2938^{1.3974}_{1.1919}$	$\mathbf{1.0731}^{1.0932}_{0.9704}$
	LCBench Covertype	$10.4014^{12.5400}_{8.6155}$	$3.6258^{3.8541}_{3.4690}$	$3.3835^{3.5729}_{3.2524}$	$9.3576^{11.1695}_{8.8375}$	$5.6514^{6.4436}_{4.7534}$	$\mathbf{3.0837}^{3.2061}_{3.0306}$
	LCBench Dionis	$12.5219^{16.0751}_{9.0688}$	$3.0078^{3.3239}_{2.8663}$	$4.9541^{5.3412}_{3.5565}$	$15.5670^{18.6681}_{8.1948}$	$4.7372^{4.8799}_{3.8158}$	$\mathbf{2.4598}^{2.6145}_{2.3977}$

A3 *Model family.* Comparing Opus 4.8 with GPT 5.5 under an identical system prompt (Figure 21), we find that Sara also performs strongly with GPT 5.5. GPT 5.5 tends to do better on prior-informed benchmarks, whereas Opus 4.8 converges faster on synthetic functions. Their different behavior suggests that model-specific prompt tuning may further improve performance. In Section G.2 we further report results with open-weight LLMs backing Sara.

7 Limitations and Future Work

Structured search spaces. Sara and lenz assume a well-defined parameter space over which a GP surrogate is straightforward to specify. Open-ended settings where optimization is conducted over code diffs or similar artifacts fall outside this scope. Although the agent-in-the-loop architecture is backend-agnostic, extending it to such spaces requires representations and surrogates that preserve similarity between structured candidates; naïve categorical encodings scale poorly and discard this structure and active contribution of the LLM agent to formalize these correlations can be promising.

Sensitivity to prompt and interface design. Sara's decisions are shaped by the system prompt and the tool descriptions of lenz. We found that directive wording noticeably influences her behavior; Table 10 lists directives that improved interaction with lenz and consistency across seeds. Robust interfaces between agents and surrogate tools therefore remain important for agentic BO; also for robustness across LLM models and families.

Non-determinism. Because an LLM drives the decision loop, repeated runs on an identical problem will generally follow different trajectories (Figure 24). This introduces failure modes absent from classical BO. We also observe *tool-use inertia*: calling patterns adopted early tend to persist and reduce strategy diversity. Prompting or architectural interventions may help maintain diverse tool use.

Based on the ablations and limitations, we provide recommendations for practitioners in Section H.

8 Conclusion

We introduced **agentic Bayesian optimization**, in which an LLM agent drives the optimization loop while a Bayesian backend provides posterior uncertainty and acquisition functions. We instantiated this paradigm in Sara and lenz: Sara frames the problem, derives priors, selects evaluations, and steers the search, while lenz supports run-time reconfiguration and computational actions for deliberation. Without semantic context, Sara matches standard BO on synthetic benchmarks and outperforms LLM-based optimizers, preserving systematic search when no useful prior is available. On hyperparameter and reaction-yield benchmarks, natural-language descriptions guide the search while the backend supports continued refinement. Research on both improved BO methodology and more robust agentic tool use will directly benefit agentic BO. The paradigm is particularly suited to expensive campaigns with domain knowledge or evolving requirements, combining the systematic exploration of BO with the prior knowledge and sequential decision-making capabilities of LLMs.

AI USE STATEMENT

This research studies LLM agents, and the use of generative AI is therefore intrinsic to its methodology and proposed paradigm. Sara and other LLM-based baselines generated optimization decisions, tool interactions, and interaction trajectories that were analyzed as experimental outputs. All the objective values reported in our experiments were returned by the benchmark oracles. Sara and other coding agents were also used during the development of lenz, assisting with implementation, debugging, and refinement of the command-line interface. Generative AI tools were additionally used in preparing the final manuscript to polish formulations and improve clarity. The authors reviewed all AI-assisted changes to the code and manuscript, tested the resulting system, and take full responsibility for the final content, claims, and artifacts.

ETHICS STATEMENT

This work proposes a general optimization methodology and evaluates it on synthetic and benchmark optimization tasks. It does not involve human subjects, release new datasets containing sensitive information, or require access to private data. The main ethical consideration is that the proposed framework gives an LLM agent control over the optimization loop. As a result, model biases may influence the search strategy and the choice of evaluations. Overall, we do not expect the paradigm of agentic BO itself to introduce substantial ethical risks beyond those associated with the chosen application domain, the underlying LLM, and the general dual-use considerations that accompany improved optimization techniques.

REPRODUCIBILITY STATEMENT

To support reproducibility, we provide the full experimental details in Sections B and F, including the prompts used for the LLM-based methods, hyperparameters, and the equations underlying the new reaction benchmarks. Section D also lists the implementation details for all baselines included in our comparisons. In our experiments, we additionally use the LCBench dataset[3] and the JAHS-Bench-201 dataset[4], both of which are publicly available. To enable re-instantiation of Sara, we include the full system prompt and agent setup using Pi[5] in Section J. To support independent re-implementation of lenz, we document the backend interface, including the available functions, their inputs and outputs, and the prompt used to expose them to the agent in Section K and Section L. The packages that lenz builds on, *i.e.* BoTorch[6] and GPyTorch[7], are also publicly available under permissive licenses.

[3] LCBench is available under the Apache 2.0 license under https://github.com/automl/LCBench.

[4] JAHS-Bench-201 is available under the MIT license under https://github.com/automl/jahs_bench_201.

[5] Pi is available under the MIT license under https://github.com/earendil-works/pi.

[6] BoTorch is available under the MIT license under https://github.com/meta-pytorch/botorch.

[7] GPyTorch is available under the MIT license under https://github.com/cornellius-gp/gpytorch.

REFERENCES

Dhruv Agarwal, Manoj Ghuhan Arivazhagan, Rajarshi Das, Sandesh Swamy, Sopan Khosla, and Rashmi Gangadharaiah. Searching for optimal solutions with llms via Bayesian optimization. In *International Conference on Learning Representations (ICLR)*, 2025.

Rishabh Agarwal, Max Schwarzer, Pablo Samuel Castro, Aaron C Courville, and Marc Bellemare. Deep reinforcement learning at the edge of the statistical precipice. *Advances in Neural Information Processing Systems (NeurIPS)*, 34:29304–29320, 2021.

Lakshya A Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J Ryan, Meng Jiang, et al. Gepa: Reflective prompt evolution can outperform reinforcement learning. In *International Conference on Learning Representations (ICLR)*, 2026.

Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, 2019.

Sebastian Ament, Samuel Daulton, David Eriksson, Maximilian Balandat, and Eytan Bakshy. Unexpected improvements to expected improvement for Bayesian optimization. *Advances in Neural Information Processing Systems (NeurIPS)*, 36:20577–20612, 2023.

Maximilian Balandat, Brian Karrer, Daniel R. Jiang, Samuel Daulton, Benjamin Letham, Andrew Gordon Wilson, and Eytan Bakshy. BoTorch: A framework for efficient monte-carlo Bayesian optimization. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

Archit Bansal, Danny Stoll, Maciej Janowski, Arber Zela, and Frank Hutter. JAHS-Bench-201: A foundation for research on joint architecture and hyperparameter search. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research (JMLR)*, 2012.

James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2011.

Daniil A. Boiko, Robert MacKnight, Ben Kline, and Gabe Gomes. Autonomous chemical research with large language models. *Nature*, 2023.

Paul Brunzema and Sebastian Trimpe. BayeSQP: Bayesian optimization through sequential quadratic programming. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.

Frederick Callaway, Bas van Opheusden, Sayan Gul, Priyam Das, Paul M. Krueger, Thomas L. Griffiths, and Falk Lieder. Rational use of cognitive resources in human planning. *Nature Human Behaviour*, 6:1112–1125, 2022.

Chih-Yu Chang, Milad Azvar, Chinedum Okwudire, and Raed Al Kontar. LLINBO: Trustworthy llm-in-the-loop Bayesian optimization. *arXiv preprint arXiv:2505.14756*, 2025.

Angelica Chen, David Dohan, and David So. EvoPrompting: Language models for code-level neural architecture search. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023a.

Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W. Cohen. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *Transactions on Machine Learning Research (TMLR)*, 2023b.

Yutian Chen, Xingyou Song, Chansoo Lee, Zi Wang, Richard Zhang, David Dohan, Kazuya Kawakami, Greg Kochanski, Arnaud Doucet, Marc'Aurelio Ranzato, Sagi Perel, and Nando de Freitas. Towards learning universal hyperparameter optimizers with transformers. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

Zhuo Chen, Xinzhe Yuan, Jianshu Zhang, Jinzong Dong, Ruichen Zhou, Yingchun Niu, Tianhang Zhou, Yu Yang Fredrik Liu, Yuqiang Li, Nanyang Ye, et al. Labo: Llm-accelerated Bayesian optimization through broad exploration and selective experimentation. In *International Conference on Machine Learning*, 2026.

Abdoulatif Cissé, Xenophon Evangelopoulos, Vladimir V Gusev, and Andrew I Cooper. Language-based Bayesian optimization research assistant (bora). In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2025.

Samuel Daulton, Maximilian Balandat, and Eytan Bakshy. Differentiable expected hypervolume improvement for parallel multi-objective Bayesian optimization. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

Samuel Daulton, Maximilian Balandat, and Eytan Bakshy. Parallel Bayesian optimization of multiple noisy objectives with expected hypervolume improvement. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

David Eriksson and Matthias Poloczek. Scalable constrained Bayesian optimization. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2021.

David Eriksson, Michael Pearce, Jacob R. Gardner, Ryan D. Turner, and Matthias Poloczek. Scalable global optimization via local Bayesian optimization. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

Lukas Fehring, Marcel Wever, Maximilian Spliethöver, Leona Hennig, Henning Wachsmuth, and Marius Lindauer. Dynamic priors in Bayesian optimization for hyperparameter optimization. *arXiv preprint arXiv:2511.02570*, 2025.

Fabio Ferreira, Lucca Wobbe, Arjun Krishnakumar, Frank Hutter, and Arber Zela. Can LLMs beat classical hyperparameter optimization algorithms? a study on autoresearch. *arXiv preprint arXiv:2603.24647*, 2026.

Peter I. Frazier. A tutorial on Bayesian optimization. *arXiv preprint arXiv:1807.02811*, 2018.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. PAL: Program-aided language models. In *International Conference on Machine Learning (ICML)*, 2023.

Jacob Gardner, Geoff Pleiss, Kilian Q Weinberger, David Bindel, and Andrew G Wilson. Gpytorch: Blackbox matrix-matrix Gaussian process inference with gpu acceleration. *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

Jacob R Gardner, Matt J Kusner, Zhixiang Eddie Xu, Kilian Q Weinberger, and John P Cunningham. Bayesian optimization with inequality constraints. In *International Conference on Machine Learning (ICML)*, volume 2014, pp. 937–945, 2014.

Roman Garnett. *Bayesian Optimization*. Cambridge University Press, 2023.

Michael A Gelbart, Jasper Snoek, and Ryan P Adams. Bayesian optimization with unknown constraints. *arXiv preprint arXiv:1403.5607*, 2014.

Rushil Gupta, Jason Hartford, and Bang Liu. Llms for Bayesian optimization in scientific domains: Are we there yet? *arXiv preprint arXiv:2509.21403*, 2025.

Nikolaus Hansen. The CMA evolution strategy: A tutorial. *arXiv preprint arXiv:1604.00772*, 2016.

Nicholas Hay, Stuart Russell, David Tolpin, and Solomon Eyal Shimony. Selecting computations: Theory and applications. In *Uncertainty in Artificial Intelligence (UAI)*, 2012.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, DDL Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 2022.

Noah Hollmann, Samuel Müller, and Frank Hutter. Large language models for automated data science: Introducing CAAFE for context-aware automated feature engineering. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Ceyao Zhang, Jinlin Wang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. MetaGPT: Meta programming for a multi-agent collaborative framework. In *International Conference on Learning Representations (ICLR)*, 2024.

Qian Huang, Jian Vora, Percy Liang, and Jure Leskovec. Benchmarking large language models as ai research agents. *arXiv preprint arXiv:2310.03302*, 2023.

Carl Hvarfner, Danny Stoll, Artur Souza, Marius Lindauer, Frank Hutter, and Luigi Nardi. πBO: Augmenting acquisition functions with user beliefs for Bayesian optimization. In *International Conference on Learning Representations (ICLR)*, 2022.

Donald R. Jones, Matthias Schonlau, and William J. Welch. Efficient global optimization of expensive black-box functions. *Journal of Global Optimization*, 13(4):455–492, 1998.

Saurav Kadavath, Tom Conerly, Amanda Askell, Tom Henighan, Dawn Drain, Ethan Perez, Nicholas Schiefer, Zac Hatfield-Dodds, Nova DasSarma, Eli Tran-Johnson, et al. Language models (mostly) know what they know. *arXiv preprint arXiv:2207.05221*, 2022.

Katarzyna Kobalczyk, Jerry Lin, Benjamin Letham, Zhuokai Zhao, Maximilian Balandat, and Eytan Bakshy. LILO: Bayesian optimization with natural language feedback. In *International Conference on Machine Learning (ICML)*, 2026.

Agustinus Kristiadi, Felix Strieth-Kalthoff, Marta Skreta, Pascal Poupart, Alan Aspuru-Guzik, and Geoff Pleiss. A sober look at LLMs for material discovery: Are they actually good for Bayesian optimization over molecules? In *International Conference on Machine Learning (ICML)*, 2024.

Eric Hans Lee, Valerio Perrone, Cedric Archambeau, and Matthias Seeger. Cost-aware bayesian optimization. *arXiv preprint arXiv:2003.10870*, 2020.

Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *Journal of Machine Learning Research (JMLR)*, 2018.

Siyi Liu, Chen Gao, and Yong Li. Large language model agent for hyper-parameter optimization. *arXiv preprint arXiv:2402.01881*, 2024a.

Tennison Liu, Raul Astudillo, Vasan Bhavana, and Mihaela van der Schaar. Large language models to enhance Bayesian optimization. In *International Conference on Learning Representations (ICLR)*, 2024b.

Chris Lu, Cong Lu, Robert Tjarko Lange, Jakob Foerster, Jeff Clune, and David Ha. The AI scientist: Towards fully automated open-ended scientific discovery. *arXiv preprint arXiv:2408.06292*, 2024.

Andres M. Bran, Sam Cox, Oliver Schilter, Carlo Baldassari, Andrew D. White, and Philippe Schwaller. ChemCrow: Augmenting large-language models with chemistry tools. *arXiv preprint arXiv:2304.05376*, 2023.

Meher Bhaskar Madiraju and Meher Sai Preetam Madiraju. OptiMindTune: A multi-agent framework for intelligent hyperparameter optimization. *arXiv preprint arXiv:2505.19205*, 2025.

Kanan Mahammadli and Seyda Ertekin. Sequential large language model-based hyper-parameter optimization. *arXiv preprint arXiv:2410.20302*, 2024.

Alonso Marco, Philipp Hennig, Stefan Schaal, and Sebastian Trimpe. On the design of LQR kernels for efficient controller learning. In *IEEE Conference on Decision and Control (CDC)*, 2017.

Giang Ngo, Dat Phan Trong, Dang Nguyen, Sunil Gupta, and Svetha Venkatesh. Adaptive acquisition selection for Bayesian optimization with large language models. In *International Conference on Learning Representations (ICLR)*, 2026.

Miles Olson, Elizabeth Santorella, Louis C Tiao, Sait Cakmak, Mia Garrard, Samuel Daulton, Zhiyuan Jerry Lin, Sebastian Ament, Bernard Beckerman, Eric Onofrey, et al. Ax: A platform for adaptive experimentation. In *AutoML 2025 ABCD Track*, 2025.

Ali Rahimi and Benjamin Recht. Random features for large-scale kernel machines. *Advances in Neural Information Processing Systems (NeurIPS)*, 2007.

Bojana Ranković, Ryan-Rhys Griffiths, and Philippe Schwaller. Large language models as uncertainty-calibrated optimizers for experimental discovery. *arXiv preprint arXiv:2504.06265*, 2025.

Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning*. MIT Press, 2006.

James Requeima, John Bronskill, Dami Choi, Richard E. Turner, and David Duvenaud. LLM processes: Numerical predictive distributions conditioned on natural language. In *International Conference on Machine Learning (ICML)*, 2024.

Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M. Pawan Kumar, Emilien Dupont, Francisco J. R. Ruiz, Jordan S. Ellenberg, Pengming Wang, Omar Fawzi, Pushmeet Kohli, and Alhussein Fawzi. Mathematical discoveries from program search with large language models. *Nature*, 2024.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Andrej Schwanke, Lyubomir Ivanov, David Salinas, Fabio Ferreira, Aaron Klein, Frank Hutter, and Arber Zela. Improving llm-based global optimization with search space partitioning. *arXiv preprint arXiv:2505.21372*, 2025.

Jonas Seng, Fabrizio Ventola, Zhongjie Yu, and Kristian Kersting. Hyperparameter optimization via interacting with probabilistic circuits. In *International Conference on Automated Machine Learning (AutoML)*. PMLR, 2025.

Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P. Adams, and Nando de Freitas. Taking the human out of the loop: A review of Bayesian optimization. *Proceedings of the IEEE*, 2016.

Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. Practical Bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2012.

Xingyou Song, Yingtao Tian, Robert Tjarko Lange, Chansoo Lee, Yujin Tang, and Yutian Chen. Position: leverage foundational models for black-box optimization. In *International Conference on Machine Learning (ICML)*, 2024.

Niranjan Srinivas, Andreas Krause, Sham M. Kakade, and Matthias Seeger. Gaussian process optimization in the bandit setting: No regret and experimental design. In *International Conference on Machine Learning (ICML)*, 2010.

David Stenger and Dirk Abel. Benchmark of Bayesian optimization and metaheuristics for control engineering tuning problems with crash constraints. *arXiv preprint arXiv:2211.02571*, 2022.

Richard Suwandi, Feng Yin, Juntao Wang, Renjie Li, Tsung-Hui Chang, and Sergios Theodoridis. Adaptive kernel design for Bayesian optimization is a piece of cake with llms. *Advances in Neural Information Processing Systems (NeurIPS)*, 38, 2026.

Katherine Tian, Eric Mitchell, Huaxiu Yao, Christopher D. Manning, and Chelsea Finn. Just ask for calibration: Strategies for eliciting calibrated confidence scores from language models fine-tuned with human feedback. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2023.

Alexander Tornede, Difan Deng, Theresa Eimer, Joseph Giovanelli, Aditya Mohan, Tim Ruhkopf, Sarah Segel, Daphne Theodorakopoulos, Tanja Tornede, Henning Wachsmuth, and Marius Lindauer. AutoML in the age of large language models: Current challenges, future opportunities and risks. *Transactions on Machine Learning Research (TMLR)*, 2024.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *Transactions on Machine Learning Research (TMLR)*, 2023.

Miao Xiong, Zhiyuan Hu, Xinyang Lu, Yifei Li, Jie Fu, Junxian He, and Bryan Hooi. Can LLMs express their uncertainty? an empirical evaluation of confidence elicitation in LLMs. *arXiv preprint arXiv:2306.13063*, 2024.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V. Le, Denny Zhou, and Xinyun Chen. Large language models as optimizers. In *International Conference on Learning Representations (ICLR)*, 2024.

Zhuo Yang, Daolang Wang, Lingli Ge, Beilun Wang, Tianfan Fu, and Yuqiang Li. Reasoning BO: Enhancing Bayesian optimization with the long-context reasoning power of LLMs. *arXiv preprint arXiv:2505.12833*, 2025.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023.

Michael R. Zhang, Nishkrit Desai, Juhan Bae, Jonathan Lorraine, and Jimmy Ba. Using large language models for hyperparameter optimization. *arXiv preprint arXiv:2312.04528*, 2023a.

Shujian Zhang, Chengyue Gong, Lemeng Wu, Xingchao Liu, and Mingyuan Zhou. AutoML-GPT: Automatic machine learning with GPT. *arXiv preprint arXiv:2305.02499*, 2023b.

Lucas Zimmer, Marius Lindauer, and Frank Hutter. Auto-PyTorch: Multi-fidelity metalearning for efficient and robust AutoDL. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.

APPENDIX CONTENTS

A Extended Related Work **17**
- A.1 Bayesian optimization and AutoML . . . 17
- A.2 Hybrid LLM–BO methods . . . 17
- A.3 LLMs as the optimizer . . . 18
- A.4 LLM agents for optimization and ML research . . . 18

B Experimental design **18**
- B.1 GP sample-path benchmarks . . . 20
- B.2 Reaction-yield family . . . 20
- B.3 Neural scaling-law study . . . 21

C Metrics **22**

D Details on baselines **23**

E Detailed benchmark results **24**
- E.1 Synthetic benchmarks . . . 24
- E.2 Hyperparameter optimization . . . 24
- E.3 Chemistry-informed reaction optimization . . . 25

F Cost-aware and asynchronous optimization on JAHS-Bench-201 **27**
- F.1 Experimental setting . . . 27
- F.2 Cost-aware optimization under a wall-clock deadline . . . 28
- F.3 Influence of reasoning effort . . . 29
- F.4 Runtime surrogate and source-code context . . . 30
- F.5 Task materials . . . 31

G Ablations and behavioral analyses **33**
- G.1 Surrogate backend and action space . . . 33
- G.2 Dependence on the backing LLM . . . 35
- G.3 Tool-use behavior . . . 37
- G.4 Computational cost . . . 37

H Recommendations for practitioners **40**

I A meta-MDP completion of agentic BO **40**

J Sara implementation and system prompt **42**
- J.1 Instantiation and core directives . . . 42
- J.2 Complete system prompt . . . 42

K lenz reference sheet **44**

L The lenz backend and command-line interface **45**
- L.1 Architecture and Bayesian backend . . . 45
- L.2 Command-line interface . . . 45

A Extended Related Work

This appendix expands the related-work discussion in Section 3, providing fuller coverage of BO, AutoML, hybrid LLM–BO methods, LLM optimizers, and agentic research systems.

A.1 Bayesian optimization and AutoML

Classical BO couples a probabilistic surrogate, typically a GP (Rasmussen & Williams, 2006), with an acquisition function that balances exploration and exploitation (Jones et al., 1998; Srinivas et al., 2010; Shahriari et al., 2016; Frazier, 2018; Garnett, 2023). Mature frameworks such as Ax (Olson et al., 2025) or Optuna (Akiba et al., 2019) package these components into reusable routines, but by design the optimization policy is declared before the campaign and held fixed throughout. Notable extensions equip BO with adaptive mechanisms: TuRBO (Eriksson et al., 2019; Eriksson & Poloczek, 2021) restricts the search to a local region that expands or contracts with progress, while user priors can bias early search through the acquisition function (Hvarfner et al., 2022). Both are *hand-designed* adaptation rules for a specific situation. Interactive BO extends this idea by allowing users to steer the search during a campaign. IBO-HPC conditions candidate generation on user-specified beliefs through probabilistic circuits (Seng et al., 2025), while DynaBO incorporates successive priors and safeguards against misleading ones (Fehring et al., 2025). Both retain a probabilistic surrogate but restrict interaction to structured priors interpreted by a fixed selection rule. Agentic BO broadens this interaction by allowing an agent to reason over unstructured context and the evolving campaign, and to adapt the optimization process accordingly. Beyond BO, the broader AutoML literature—from random search (Bergstra & Bengio, 2012) and tree-structured estimators (Bergstra et al., 2011) to multi-fidelity methods (Li et al., 2018) and meta-learned optimizers (Chen et al., 2022)—similarly commits to a fixed search strategy in advance (Tornede et al., 2024). Sara keeps the explicit probabilistic model but inverts the control relationship: *which* strategy to apply, and *when*, becomes a decision made by reasoning over the run, informed by the priors extracted from the instructions and the observations gathered during it.

A.2 Hybrid LLM–BO methods

A growing line of work keeps the Bayesian machinery but hands a *single, fixed* component of the loop to an LLM; methods differ in which component that is. Some target the *kernel*, using an LLM to design or adapt the GP covariance from a contextual description of the problem (Suwandi et al., 2026). Others reshape the *surrogate* itself, training LLM embeddings through the GP marginal likelihood so that uncertainty stays calibrated (Ranković et al., 2025), or fusing computationally cheap LLM predictions with real evaluations in a multi-fidelity surrogate (Chen et al., 2026). A third line acts at the *acquisition* step, letting the LLM select the acquisition function from a portfolio at each iteration (Ngo et al., 2026). A fourth group acts on candidate selection, letting the LLM propose or filter candidates (Cissé et al., 2025; Yang et al., 2025) or arbitrate between LLM and GP proposals (Chang et al., 2025). Finally, Kobalczyk et al. (2026) intervene on outcome interpretation, translating natural-language feedback into preferences for a preference GP. Agarwal et al. (2025) go further, running BO in a latent embedding space where acquisition proposals bias in-context examples for LLM generation, effectively turning the LLM into a conditioned decoder from surrogate proposals to candidate text. Together these results show that an LLM and a probabilistic model are complementary, and several report strong gains in the early, low-data regime. In every case, however, the point of intervention—and the policy around it—is fixed before the first evaluation and the LLM essentially fills one slot in a standard pipeline. Even the methods that invoke the LLM adaptively do so through a hand-designed trigger, *e.g.* a heuristic on GP uncertainty and performance plateaus (Cissé et al., 2025), or a fixed probabilistic switching schedule (Mahammadli & Ertekin, 2024; Chang et al., 2025), rather than leaving the model itself to decide when to intervene. Agentic BO instead places the agent *at the center* of the loop, where it can reconfigure any component—bounds, objectives, constraints, acquisition behavior, even the objective/constraint partition—at run time, while delegating calibrated numerical search to lenz. Importantly, the methods above are complementary to, not superseded by, agentic BO, since an LLM-designed kernel (Suwandi et al., 2026), a latent-space generation strategy (Agarwal et al., 2025), or a preference surrogate (Kobalczyk et al., 2026) can each be wrapped as a backend capability that the agent invokes when appropriate, and the system prompt can incorporate the acquisition selection strategy of Ngo et al. (2026). The

contribution of the agentic layer is orthogonal; it decides *when* and *how* to combine such capabilities in response to accumulating evidence, rather than fixing the choice before the first evaluation.

A.3 LLMs AS THE OPTIMIZER

At the other extreme, the optimization itself moves inside the LLM, an approach advocated as a general direction for black-box problems (Song et al., 2024). OPRO (Yang et al., 2024) and related methods prompt the model with a trajectory summary and ask it to emit the next configuration directly (Yang et al., 2024; Zhang et al., 2023a); program-search methods evolve solutions from LLM-proposed edits (Romera-Paredes et al., 2024); and LLAMBO (Liu et al., 2024b) frames this as BO by prompting for a promising point. None maintains an explicit probabilistic surrogate, so their search is not guided by posterior uncertainty over the objective. While LLM processes (Requeima et al., 2024) show that LLMs can form predictive distributions, a large body of work finds that their uncertainties can be poorly calibrated (Kadavath et al., 2022; Tian et al., 2023; Xiong et al., 2024), including for molecular BO specifically (Kristiadi et al., 2024). Recent evidence reinforces the value of a dedicated optimization substrate. Ferreira et al. (2026) find that classical HPO methods (CMA-ES, TPE) outperform pure-LLM optimizers on a language-model tuning task, and similar findings question whether LLMs are ready for scientific BO on their own (Gupta et al., 2025). On this basis, Ferreira et al. (2026) build Centaur, a hybrid that shares CMA-ES internal state with an LLM through its prompt. Centaur is the closest baseline in spirit to agentic BO; however, augmenting a prompt underutilizes the sequential decision-making of LLM agents. We instead give Sara full control over a BO backend and show substantially better performance than Centaur in our experiments.

A.4 LLM AGENTS FOR OPTIMIZATION AND ML RESEARCH

A rapidly growing line of work wraps LLMs in agent loops to drive ML workflows. AgentHPO (Liu et al., 2024a) reads a task description, proposes configurations, and refines them from past trials; OptiMindTune (Madiraju & Madiraju, 2025) splits the job across cooperating agents; others orchestrate whole pipelines (Zhang et al., 2023b), generate features (Hollmann et al., 2023), or search architectures by evolving code (Chen et al., 2023a), and benchmarks now probe how well LLMs act as ML research agents (Huang et al., 2023). Like the in-context methods above, these drive search by prompting over text summaries of past trials with no calibrated posterior. Agentic BO differs in that a calibrated surrogate can inform every decision the agent makes. We believe that agentic BO could be readily incorporated in the above approaches. More broadly, general reasoning-and-acting frameworks (Yao et al., 2023), tool-use paradigms (Schick et al., 2023), multi-agent systems (Hong et al., 2024), open-ended embodied agents (Wang et al., 2023), and end-to-end scientific discovery pipelines (Lu et al., 2024; Boiko et al., 2023; M. Bran et al., 2023) emphasize broad autonomy. A recurring lesson is that LLMs are unreliable at direct computation but effective when they *delegate* to precise tools (Gao et al., 2023; Chen et al., 2023b; Schick et al., 2023). Sara applies this principle to zero-order optimization, informing her search with a BO backend while remaining deliberately narrower than open-ended agents.

B EXPERIMENTAL DESIGN

This appendix details the benchmark suite, the agent and backend configuration, the GP sample paths, the reaction-yield family, and the neural scaling-law study used for the mid-run reformulation of Section 6.2. We follow the evaluation protocol of Section 6 with every method running under the same evaluation budget, and we report simple regret—or the raw best value for the yield objectives—against the number of evaluations.

Benchmark suite and budgets. Table 3 lists the problems, their dimensionality, and the per-problem evaluation budget. The suite spans low- to high-dimensional synthetic functions, constrained and multi-objective variants, the LCBench hyperparameter-optimization surrogates (Zimmer et al., 2021), and the reaction-yield family described below. Ax uses a short Sobol initialization of five trials. Sara and LLAMBO begin proposing from the first trial, with Sara choosing her own initialization strategy through lenz. Centaur runs CMA-ES alone for the first ten trials before the LLM begins to intervene as described in Ferreira et al. (2026).

Table 3: Benchmark suite and evaluation budgets. Synthetic problems are optimized in the normalized unit cube; the reaction-yield family shares a single 6-D operating space across four reactions.

Benchmark	Dimensionality	Budget (evaluations)
Branin	2-D	50
Hartmann	6-D	100
Ackley	10-D	150
Ackley	20-D	200
Constrained Hartmann	6-D	100
Bi-objective GP samples	2-, 4-, 6-, 8-D	40, 60, 80, 120
LCBench (HPO surrogate)	7-D	80
Reaction yield (all four variants)	6-D	40

```
# Prior knowledge for this tuning task

You are tuning a **funnel-shaped feed-forward neural network** (the layer widths taper from a
maximum width down toward the output) trained with **SGD + cosine annealing of the learning rate (
no warm restarts)**.
The objective is **validation accuracy on a held-out set, as a percentage (0-100) -- maximize it.**


## Search space

Seven hyperparameters.
Several knobs span orders of magnitude and are therefore best
searched on a **log scale**.

| parameter | range | scale | type | what it does |
|---|---|---|---|---|
| `learning_rate` | 1e-4 .. 1e-1 | log | float | SGD step size |
| `batch_size` | 16 .. 512 | log | int | SGD minibatch size |
| `max_units` | 64 .. 1024 | log | int | width of the widest layer (the funnel`s mouth) |
| `num_layers` | 1 .. 4 | linear | int | depth of the funnel |
| `momentum` | 0.1 .. 0.99 | log | float | SGD momentum; lives close to 1 |
| `weight_decay` | 1e-5 .. 1e-1 | linear | float | L2 regularization strength |
| `max_dropout` | 0.0 .. 1.0 | linear | float | dropout rate cap; regularizes |
```

Figure 7: Problem description for all LCBench benchmarks.

Agent and backend. Unless noted, Sara is backed by Claude Opus 4.8; the model-capability ablation (Section 6.3) additionally reports Sonnet 4.6 and Haiku 4.5, and the reasoning ablation sweeps the effort levels *off*, *low*, *medium*, and *high*. The lenz backend fits a GP surrogate with a Matérn kernel, normalizes inputs to the unit cube, and standardizes outcomes. By default, it uses variants of log expected improvement for single-objective and constrained problems and the corresponding hypervolume-based acquisition for multi-objective problems. Outcome constraints are handled through the posterior probability of feasibility as $\mathbb{P}(g(\boldsymbol{x}) \leq 0)$ (*cf*. Section L). Sara receives the same evaluation budget as the baselines.

Preventing benchmark recognition. Because textbook benchmark functions and their optima are well represented in pretraining corpora, each synthetic optimum is relocated per seed by up to a quarter of every dimension's range, the domain is canonicalized to the unit cube, and the parameters are renamed. Each Sara run additionally executes inside a sandbox that contains only the task description and a symbolic link to the evaluation oracle and is named by a random token, so neither the path nor the directory contents reveal which benchmark is being solved (*cf*. Sections 6 and G).

Natural-language priors. Where a description is available—for the LCBench and reaction-yield tasks—the *same* prior text is provided to Sara, LLAMBO, and Centaur, so that any difference reflects the algorithm rather than the information supplied. Each prior conveys genuine domain knowledge (the model family and parameter scales for LCBench, the reaction class) without revealing parameter values or, for LCBench, the identity of the dataset. The synthetic benchmarks receive no prior and thus probe the no-knowledge regime.

```
This is a Suzuki-Miyaura cross-coupling.

### Shared search space

| Parameter        | Range   | Meaning                                                  |
|------------------|---------|----------------------------------------------------------|
| temperature_C    | 0-150   | Reaction temperature                                     |
| pressure_bar     | 1-10    | Reaction pressure                                        |
| catalyst_mol_pct | 0.1-10  | Catalyst loading (mol %)                                 |
| residence_min    | 1-180   | Residence time                                           |
| base_strength    | 0-1     | Dimensionless: 0 = weak/no base; 1 = very strong base    |
| water_frac       | 0-1     | Volume fraction of water                                 |

The metric is the reaction yield in percent (higher is better).
```

Figure 8: *Representative problem description for the reaction-yield benchmarks.* The first line names the reaction; all four problems share the remaining search-space description.

B.1 GP SAMPLE-PATH BENCHMARKS

We draw approximate squared-exponential GP samples using $M = 1028$ random Fourier features (Rahimi & Recht, 2007),

$$f_i(\boldsymbol{x}) = \sum_{m=1}^{M} w_m \phi_m(\boldsymbol{x}), \qquad \phi_m(\boldsymbol{x}) = \sqrt{\frac{2}{M}} \cos(\boldsymbol{\theta}_m^\top \boldsymbol{x} + \tau_m), \tag{B.1}$$

where $w_m \overset{\text{iid}}{\sim} \mathcal{N}(0,1)$, $\boldsymbol{\theta}_m$ follows the kernel's spectral density, $\tau_m \sim \mathcal{U}(0, 2\pi)$, and all lengthscales are 0.2. For the bi-objective sweep, we independently draw two such paths, minimize both, and measure hypervolume relative to $(0,0)$ in dimensions 2, 4, 6, and 8, with evaluation budgets of 40, 60, 80, and 120, respectively.

B.2 REACTION-YIELD FAMILY

We model continuous-flow process optimization for four named organic reactions. The reactions share a six-dimensional operating space—temperature $T \in [0, 150]\,^\circ\mathrm{C}$, pressure $P \in [1, 10]\,\mathrm{bar}$, catalyst loading $c \in [0.1, 10]\,\mathrm{mol}\%$, residence time $t \in [1, 180]\,\mathrm{min}$, dimensionless base strength $b \in [0, 1]$, and solvent water fraction $w \in [0, 1]$—but each favors a different operating regime. Base strength ranges from weak or absent base ($b = 0$) to very strong base ($b = 1$). Pressure is an intentionally inactive nuisance dimension for every reaction, and base strength is additionally inactive for the Cu-catalyzed conjugate addition of a Grignard reagent.

The yield is a product of phenomenological factors, each valued in $(0, 1]$ and equal to one at its regime-specific optimum:

$$y(T, P, c, t, b, w) \;=\; y_{\max} \cdot f_T \cdot f_P \cdot f_c \cdot f_t \cdot f_b \cdot f_w \cdot f_{\text{int}}, \qquad f_P \equiv 1. \tag{B.2}$$

Temperature, catalyst loading, and residence time use asymmetric Gaussians:

$$f_z = \exp\!\Big(-\tfrac{1}{2}\big((z - \mu_z)/\sigma_z(z)\big)^2\Big), \qquad \sigma_z(z) = \begin{cases} \sigma_z^-, & z < \mu_z, \\ \sigma_z^+, & z \geq \mu_z, \end{cases} \quad z \in \{T, c, t\}. \tag{B.3}$$

These factors represent incomplete conversion below the preferred temperature, loading, or residence time and decomposition or deactivation above it. Base strength and water fraction use symmetric Gaussians, except that the base factor is flat for Grignard:

$$f_b = \begin{cases} \exp\!\Big(-\tfrac{1}{2}\big((b - \mu_b)/\sigma_b\big)^2\Big), & \text{if base strength is active,} \\ 1, & \text{for Grignard,} \end{cases} \qquad f_w = \exp\!\Big(-\tfrac{1}{2}\big((w - \mu_w)/\sigma_w\big)^2\Big). \tag{B.4}$$

Finally, a temperature–water interaction term lets the ideal water content drift linearly with temperature:

$$f_{\text{int}} = \exp\!\Big(-\tfrac{1}{2}\Big(\frac{w - \big[\mu_w + \beta\,(T - \mu_T)\big]}{\sigma_{\text{int}}}\Big)^2\Big). \tag{B.5}$$

We instantiate four reactions whose regimes reflect qualitative process requirements and list them in Table 4: Suzuki–Miyaura coupling (warm, moderately basic, and aqueous-tolerant; $y_{\max} = 95\%$), the Mizoroki–Heck reaction (hot, basic, and favoring low water; 88%), Buchwald–Hartwig amination (warm-to-hot, strongly basic, and dry; 92%), and Cu-catalyzed conjugate addition of a Grignard reagent (cold and rigorously anhydrous; 80%). The problem description supplied to the agent names only the reaction and the shared parameters and ranges and is listed in Figure 8. Thus, any informative prior must come from the model's chemical knowledge rather than being encoded in the prompt.

Table 4: Representative maximizers for the reaction-yield family. Pressure is flat for every reaction and is omitted in the table.

Reaction	$y_{\max}$ (%)	μ_T (°C)	μ_c (mol%)	μ_t (min)	μ_b	μ_w
Suzuki–Miyaura	95	62	2.4	74	0.47	0.32
Mizoroki–Heck	88	132	1.1	138	0.71	0.06
Buchwald–Hartwig	92	86	2.7	102	0.90	0.007
Grignard	80	7	4.2	42	flat	0.00

Table 5: Sensitivity parameters for the reaction-yield factors. Superscripts $-$ and $+$ denote the widths below and above the corresponding center. A dash marks an inactive factor.

Reaction	σ_T^-	σ_T^+	σ_c^-	σ_c^+	σ_t^-	σ_t^+	σ_b	σ_w	β	σ_{int}
Suzuki–Miyaura	22	24	1.6	3.5	28	65	0.25	0.16	0.0020	0.25
Mizoroki–Heck	26	16	0.9	4.5	70	45	0.12	0.18	-0.0010	0.22
Buchwald–Hartwig	24	22	1.4	3.5	35	55	0.18	0.04	0.0005	0.08
Grignard	12	10	1.5	4.0	18	65	–	0.008	0	0.40

Figure 13 shows the result on this family of benchmarks. The reaction name lets Sara infer useful operating regimes from chemical knowledge, while an uninformed optimizer must discover them from evaluations alone. In contrast to the other LLM-based baselines, Sara can use the BO backend to further fine-tune her initial solution.

B.3 Neural scaling-law study

The mid-run reformulation experiment of Section 6.2 uses a synthetic neural scaling-law benchmark that maps a model-and-training configuration to two outcomes, validation loss and training compute, without ever training a real model. The search space comprises five knobs (Table 6): learning rate η, model width d and depth ℓ (which together set the parameter count $N = 12\,\ell\,d^2$), dropout ρ, and dataset size D (in tokens). The two outcomes are the validation cross-entropy `logloss` and the `flops`, reported in PFLOP; both are to be minimized.

Table 6: Search space for the neural scaling-law study. Width and depth set the parameter count N; the dataset size is the token count D.

Parameter	**Range**	**Scale**	**Type**
`lr` (learning rate η)	$[10^{-4}, 10^{-1}]$	log	float
`embedding_dim` (width d)	[64, 1024], step 64	linear	int
`n_layers` (depth ℓ)	[2, 32]	linear	int
`dropout` (ρ)	[0, 0.5]	linear	float
`n_data` (tokens D)	$[10^8, 10^{10}]$	log	float

Following a Chinchilla-style parametric form (Hoffmann et al., 2022), the loss floor decreases with both model size and data:

$$L_0(N, D) = E + \frac{A}{N^\alpha} + \frac{B}{D^\beta}, \tag{B.6}$$

with $E = 1.69$, $A = 406$, $\alpha = 0.34$, $B = 410$, and $\beta = 0.28$. Reaching that floor requires appropriate hyperparameters; the realized loss adds a log-quadratic penalty around a size-dependent

learning-rate optimum and a mild dropout penalty:

$$\begin{aligned} \texttt{logloss} &= L_0(N, D) + C_{\text{lr}}\big(\log_{10}\eta - \log_{10}\eta^\star(N)\big)^2 \\ &\quad + C_{\text{drop}}\,(\rho - \rho^\star)^2 + \varepsilon, \\ \eta^\star(N) &= \eta_{\text{ref}}\Big(\frac{N_{\text{ref}}}{N}\Big)^s. \end{aligned} \tag{B.7}$$

where $C_{\text{lr}} = 0.6$, $C_{\text{drop}} = 1.2$, $\rho^\star = 0.1$, $\eta_{\text{ref}} = 10^{-2}$, $N_{\text{ref}} = 10^7$, $s = 0.12$, and $\varepsilon \sim \mathcal{N}(0, 0.01^2)$ is measurement noise. Because the optimal learning rate $\eta^\star$ shrinks as the model grows, η and model size interact. Training compute follows the standard estimate $\texttt{flops} = 6\,ND/10^{15}$ [PFLOP].

In the experiment, Sara receives the search space and objectives in natural language and works on a single shared trial log across two phases. In **Phase 1** she minimizes `logloss` subject to a hard constraint $\texttt{flops} \le 500$ PFLOP—a constrained single-objective campaign with roughly 20 evaluations. lenz handles the constraint via the posterior feasibility $\mathbb{P}(\texttt{flops} \le 500)$ and optimizes the noisy constrained log-EI (*cf*. Section L). Partway through, the user changes the requirement to instead recover the full loss–compute Pareto front within $\texttt{flops} \in [200,\ 700]$ PFLOP. In **Phase 2** Sara promotes `flops` from a constraint to a second objective (`set-objectives`), relaxes the hard budget to the requested band (`set-constraints`), and switches to a hypervolume acquisition function via `set-acqf` (*cf*. Figure 6). Because the accumulated observations remain valid under the new specification, no data is discarded; the reconfiguration requires only natural-language instructions to Sara, illustrating the flexibility that agentic BO affords over a conventional fixed-objective pipeline.

```
Minimize the validation logloss of a training run, subject to a compute budget. Run the
experiment as `python3 PATH_TO_EXP/train_run.py `<config json>`` --- it returns `{"logloss": <v>,
 "flops": <pflop>}`. Knobs: `lr` float in [1e-4, 1e-1] (log-scale), `embedding_dim` int in [64,
1024] step 64, `n_layers` int in [2, 32], `dropout` float in [0, 0.5], `n_data` float in [1e8, 1
e10] (log-scale). Constraint: `flops` <= 500 PFLOP. Budget ~20 evaluations. Minimize `logloss`
while staying feasible. Keep your lenz state in `./state.json`. Treat the experiment as a black
box: do not open or read `train_run.py` --- measure it only by running it.
```

(a) Prompt for **Phase 1**

```
Actually, I want the whole trade-off after talking to my manager. I want the pareto of logloss
and FLOPs between flops=200 and flops 700.
```

(b) Prompt for **Phase 2**

Figure 9: *Prompts for the mid-run reformulation from Section 6.2.*

C METRICS

The summary plots in Figure 4 compare problems with different objective scales and evaluation budgets. For the synthetic and LCBench panels, we therefore normalize within each problem and aggregate using the interquartile mean (IQM), following the recommendations of Agarwal et al. (2021). The full per-problem trajectories remain available in Section E.

Normalized regret trajectories. Let $r_{p,a,s}(t)$ denote the simple regret of method a on problem p after evaluation t in seed s, and let B_p be that problem's evaluation budget. We align problems by the consumed fraction of their budget, $\tau \in \{0.01, 0.02, \ldots, 1\}$, using the best-so-far regret at $t = \lceil \tau B_p \rceil$. Each trajectory is normalized by the mean final regret of Sobol on the same problem,

$$z_{p,a,s}(\tau) = \frac{r_{p,a,s}(\lceil \tau B_p \rceil)}{\frac{1}{n_p}\sum_{s'=1}^{n_p} r_{p,\text{Sobol},s'}(B_p)}. \tag{C.1}$$

Thus, zero is optimal, one is the mean final Sobol regret, and lower values are better. We include a problem in a comparison only when results are available for every method shown.

Interquartile mean. At each budget fraction, we pool the normalized values across problems and seeds into the multiset $\mathcal{Z}_{a,\tau} = \{z_{p,a,s}(\tau)\}_{p,s}$. Writing $Q_{a,\tau}(u)$ for its empirical quantile function, the reported aggregate is

$$\text{IQM}(\mathcal{Z}_{a,\tau}) = 2\int_{0.25}^{0.75} Q_{a,\tau}(u)\,\mathrm{d}u. \tag{C.2}$$

Equivalently, the IQM is the mean of the central 50% of the pooled problem–seed values after discarding the lowest and highest quartiles. Our finite-sample implementation assigns fractional weight at a quartile boundary when the number of values is not divisible by four. Because every problem is evaluated with the same number of seeds, pooling gives each problem equal weight while retaining run-level variation.

Confidence intervals. The shaded regions are pointwise 95% percentile confidence intervals from 1,000 stratified bootstrap replicates. For each replicate, we independently resample n_p seeds with replacement within every problem, pool the resulting values, and recompute the IQM at each budget fraction. This preserves the benchmark composition while propagating uncertainty from the finite set of runs, as advocated by Agarwal et al. (2021).

Area-under-the-curve (AUC). The reaction panel in Figure 4 instead retains the variation among the four individual benchmarks. For a best-so-far trajectory $q(1), \ldots, q(B_p)$, we compute the AUC over the normalized evaluation budget using the trapezoidal rule,

$$\mathrm{AUC}(q) = \frac{1}{B_p - 1} \sum_{t=1}^{B_p - 1} \frac{q(t) + q(t+1)}{2}. \tag{C.3}$$

This measures average best-so-far performance over the campaign and has the same units as the objective. For each seed, we retain both $q(B_p)$ and $\mathrm{AUC}(q)$ and normalize either quantity problem-wise as

$$\widetilde{q}_{p,a,s} = \frac{q_{p,a,s} - \overline{q}_{p,\mathrm{Sobol}}}{q_p^\star - \overline{q}_{p,\mathrm{Sobol}}}, \tag{C.4}$$

where $q_p^\star$ is the known optimum and $\overline{q}_{p,\mathrm{Sobol}}$ is the mean Sobol value on that problem. Here, zero denotes mean Sobol performance, one denotes the optimum, and higher values are better. We average the normalized values over seeds within each reaction and the polygon is the convex hull of the four reaction-level points to show the variation, and the diamond marks their arithmetic mean.

D Details on baselines

We compare against four baselines that include pseudo-random sampling, classical Bayesian optimization, and LLM-driven optimization. Where a natural-language description is available, it is given to every LLM-driven method (Section B), so that comparisons isolate the optimization strategy rather than the information supplied or the underlying model.

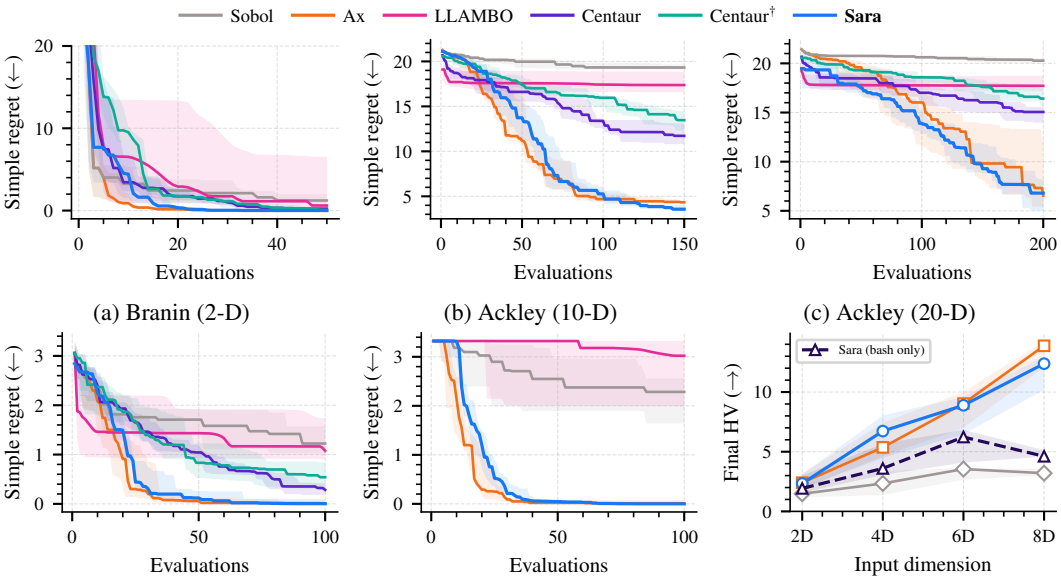
Sobol. Scrambled Sobol quasi-random sampling, run for the full evaluation budget without any model. It serves as a space-filling lower bound.

Ax. We use Ax (Olson et al., 2025) with its default generation strategy and no custom overrides, representing a well-tuned classical BO policy with best practices implemented as defaults. A short Sobol initialization of five trials is followed by a model-based search using a GP surrogate with a Matérn kernel and a log-expected-improvement acquisition function, with the noisy hypervolume analogue used automatically on multi-objective problems. For the constrained problem, Ax defaults to constrained log-expected improvement.

LLAMBO. Following Liu et al. (2024b), LLAMBO renders the trial history as a table and queries the LLM for the next configuration directly, without fitting a calibrated surrogate. It proposes from the first trial, with no Sobol warm-up, aiming to leverage the full warm-start capability of LLMs.

Centaur. Following Ferreira et al. (2026), Centaur shares the internal state of a CMA-ES optimizer with an LLM autoresearcher, and we use the same hyperparameters as the original paper. CMA-ES is warm-started on its own for the first ten trials before the LLM begins to intervene; thereafter the LLM proposes a fraction of the evaluations (30%). The initial step size of CMA-ES is set to $\sigma_0 = 1/6$ on the unit cube and the remaining CMA-ES settings left at their defaults. Centaur observes trial outcomes but, unlike Sara, is exposed to the optimizer state only through its prompt but cannot interact with the optimizer in any way; this is the closest baseline in spirit to agentic BO.

Centaur†. We additionally evaluate an LLM-first variant of Centaur that differs only in its initialization. The LLM proposes the first evaluation, and this point is used as the initial mean of CMA-ES on the normalized unit cube. Centaur† then follows the same ten-trial CMA-ES warm-up and subsequent 30% LLM intervention schedule as standard Centaur, with all other settings unchanged.



(a) Branin (2-D) (b) Ackley (10-D) (c) Ackley (20-D)

(d) Hartmann (6-D) (e) Constrained Hartmann (6-D) (f) Bi-objective GP samples

Figure 10: *Synthetic benchmarks.* Simple regret vs. evaluations for the single-objective and constrained problems; final hypervolume vs. input dimension for the multi-objective sweep. Median over 10 seeds; shaded regions indicate the 25–75% interquartile range.

LLM backend. Unless otherwise specified, all LLM-driven baselines as well as Sara are driven by Claude Opus 4.8 (*high*) so that comparisons reflect the optimization strategy.

E Detailed benchmark results

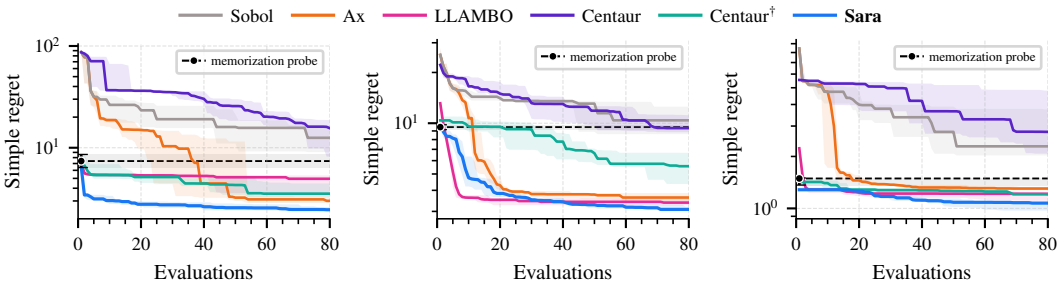
This section provides the per-benchmark optimization curves underlying the aggregate results in Figure 4. Unless stated otherwise, curves show the median over 10 seeds with the interquartile range; the benchmark construction and evaluation protocol are detailed in Sections B and C.

E.1 Synthetic benchmarks

Figure 10 decomposes the aggregate synthetic result shown the the main body of the paper into the individual benchmarks. Without a domain description, Sara largely tracks Ax across the single-objective suite, while the direct LLM-based baselines are less reliable. These results also highlight the benefit of combining classical optimization machinery with LLM-based decision making for such black-box settings since the Centaur baselines outperform LLAMBO. The multi-objective GP sweep further shows that Sara remains effective beyond the single-objective setting. We discuss the contribution of the surrogate backend separately in Section G.1.

E.2 Hyperparameter optimization

The individual LCBench curves in Figure 11 confirm that the aggregate warm-start advantage is shared across the three datasets. Sara and LLAMBO exploit the natural-language description early, whereas Ax catches up through model-based search. Centaur† benefits from its LLM-selected initialization relative to standard Centaur, but generally plateaus above the strongest methods. Since LCBench is a popular benchmark in the AutoML community, we further include a small memorization probe (dashed black line in Figure 11). In each of 10 runs per dataset, Opus 4.8 with high reasoning receives no observations and returns one configuration, which is then evaluated on the corresponding LCBench task. We provide the shared task description from Figure 7, identify the benchmark and dataset, and ask the agent to return the solution or its best guess directly. This probe cannot distinguish memorized



(a) LCBench: Dionis (b) LCBench: Covertype (c) LCBench: Airlines

Figure 11: *Hyperparameter optimization on LCBench.* Simple regret (log scale) vs. evaluations. LLM-based methods achieve strong initial performance. The dashed black line shows the median of the dataset-aware one-shot probe over 10 runs; the shaded band spans the interquartile range.

benchmark knowledge from dataset-specific reasoning, but indicates how sensitive performance is to revealing the benchmark identity. The complete Dionis prompt is shown in Figure 12 and the other datasets use the same prompt with only the dataset name replaced.

The dataset- and benchmark-aware probe performs comparably to the initial evaluations of LLM-based methods that are not explicitly given the benchmark or dataset identity. This suggests that the LLM does not simply recall near-optimal LCBench configurations. Instead, the strong initial performance of LLM-based methods is likely explained by prior knowledge about generally effective neural-network hyperparameters. Moreover, all LLM-based methods use the same backing model and receive the same initial task information, so their comparison remains fair.

```
# Tuning on LCBench

You are tuning a **funnel-shaped feed-forward neural network** (the layer widths taper from a
maximum width down toward the output) trained with **SGD + cosine annealing of the learning rate (
no warm restarts)**.
The objective is **validation accuracy on a held-out set, as a percentage (0-100) -- maximize it.**

The concrete dataset that you optimizing is LCBench:  Dionis.

## Search space

Seven hyperparameters.
Several knobs span orders of magnitude and are therefore best
searched on a **log scale**.

| parameter | range | scale | type | what it does |
|---|---|---|---|---|
| `learning_rate` | 1e-4 .. 1e-1 | log | float | SGD step size |
| `batch_size` | 16 .. 512 | log | int | SGD minibatch size |
| `max_units` | 64 .. 1024 | log | int | width of the widest layer (the funnel's mouth) |
| `num_layers` | 1 .. 4 | linear | int | depth of the funnel |
| `momentum` | 0.1 .. 0.99 | log | float | SGD momentum; lives close to 1 |
| `weight_decay` | 1e-5 .. 1e-1 | linear | float | L2 regularization strength |
| `max_dropout` | 0.0 .. 1.0 | linear | float | dropout rate cap; regularizes |

Do not use lenz but give me the solution (or your best guess of the solution) for this benchmark
and dataset.
```

Figure 12: Complete task prompt for the Dionis memorization probe. Information added relative to the standard LCBench prompt is highlighted in blue. For Airlines and Covertype, only the dataset name is changed.

E.3 Chemistry-informed reaction optimization

Figure 13 reports the reaction-yield trajectories both with and without the natural-language problem description (top and bottom row, respectively). With this description, the LLM-based methods identify productive operating regimes early, but Sara continues to refine them through the surrogate and reaches the strongest final yields across all baselines. We can further see that while the other LLM-based methods show a better early start, Ax surpasses and outperforms them on some tasks

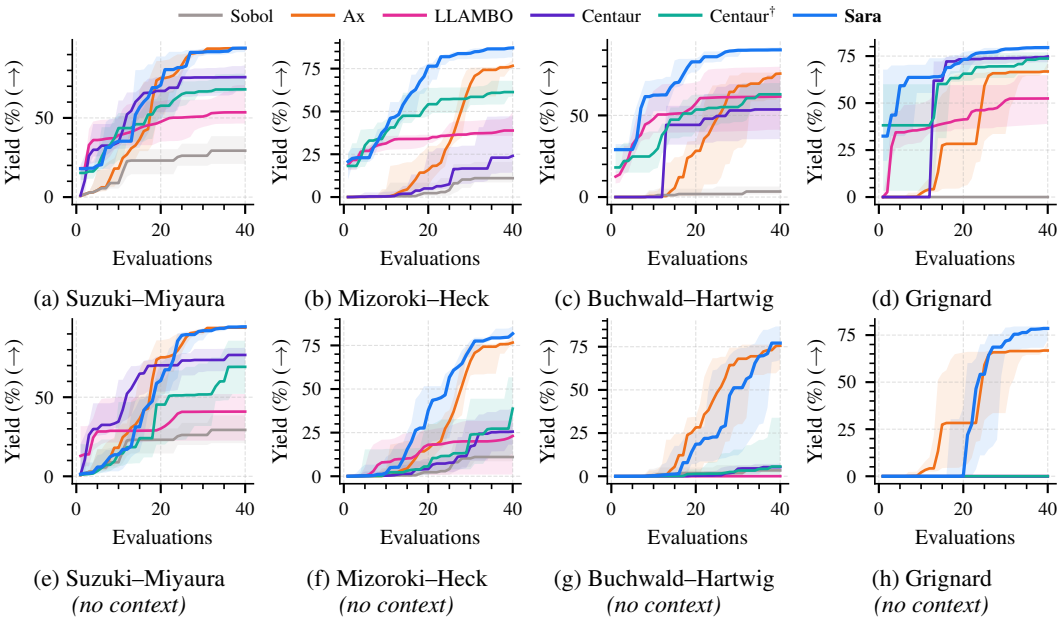


Figure 13: *Reaction-yield optimization. (Top)* Sara combines a prior-informed warm-start with surrogate-based refinement, achieving fast initial convergence without sacrificing asymptotic performance. *(Bottom)* Also without additional contextual information, Sara is able to achieve meaningful yield compared to the other baselines.

once enough observations have been collected. Without context, the warm-start advantage disappears and performance depends more strongly on systematic exploration. The six-dimensional reaction family, including base strength and water fraction, is defined in Section B.2.

F COST-AWARE AND ASYNCHRONOUS OPTIMIZATION ON JAHS-BENCH-201

We next use JAHS-Bench-201 to study a different form of surrogate-augmented autoresearch. In this setting, the agent must reason not only about promising configurations, but also about an auxiliary quantity that determines whether evaluations can be completed within a finite wall-clock budget. The benchmark combines a mixed search space with categorical variables, asynchronous parallel workers, and the goal is to find the best configuration within a time budget of 48 hours. We optimize validation accuracy on the CIFAR-10 task from JAHS-Bench-201 (Bansal et al., 2022), while treating training time as unknown before a run is launched. A historical runtime dataset allows Sara, through lenz, to model this additional metric and incorporate predicted runtimes into its deliberation. The agent must therefore identify configurations that are likely to finish in time, balance predicted performance against expected runtime, and allocate several workers asynchronously as the remaining horizon shrinks. We further show how giving the agent access to the code can further improve the performance.

F.1 EXPERIMENTAL SETTING

We consider optimization over a mixed domain $\mathcal{X} = \mathcal{X}_{\text{ord}} \times \mathcal{X}_{\text{cat}} \times \mathcal{X}_{\text{cont}}$, whose 13 parameters and component subspaces are listed in Table 7. The space combines ordered architectural and data choices, categorical cell operations, and two log-scaled continuous training parameters.

Table 7: Mixed JAHS-Bench-201 search space.

	Parameter	Domain
$\mathcal{X}_{\text{ord}}$	N	$\{1, 3, 5\}$ repeated cells per stage
	W	$\{4, 8, 16\}$ base channels
	Resolution	$\{0.25, 0.5, 1.0\}$ input multiplier
	TrivialAugment	$\{0, 1\}$ disabled or enabled
$\mathcal{X}_{\text{cat}}$	$\text{Op}_1, \ldots, \text{Op}_6$	$\{0, 1, 2, 3, 4\}$ for each edge
	Activation	$\{0, 1, 2\}$ corresponding to ReLU, Hardswish, and Mish
$\mathcal{X}_{\text{cont}}$	LearningRate	$[10^{-3}, 1]$, log scale
	WeightDecay	$[10^{-5}, 10^{-2}]$, log scale

The operation indices correspond to identity, zero, 3×3 convolution, 1×1 convolution, and average pooling. Importantly, this mapping for the operations in embedded in the code which an agent as to discover and we show in Section F.4 how this additional contextual information influences the sampling and performance. Every configuration $\boldsymbol{x} \in \mathcal{X}$ is evaluated for the full 200 epochs.

Let $f(\boldsymbol{x})$ denote the resulting validation accuracy and $c(\boldsymbol{x}) > 0$ its unknown training time. A configuration $\boldsymbol{x}_i$ launched at time s_i completes at $e_i = s_i + c(\boldsymbol{x}_i)$. With $m = 4$ parallel workers, deadline $T = 48$ hours, and at most $N_{\max} = 80$ launches, we evaluate an asynchronous optimization campaign by the best validation accuracy among runs completed before the deadline. If n configurations are launched and $\mathcal{C}_T = \{i \in \{1, \ldots, n\} : e_i \le T\}$ denotes the set of completed runs, the terminal score is

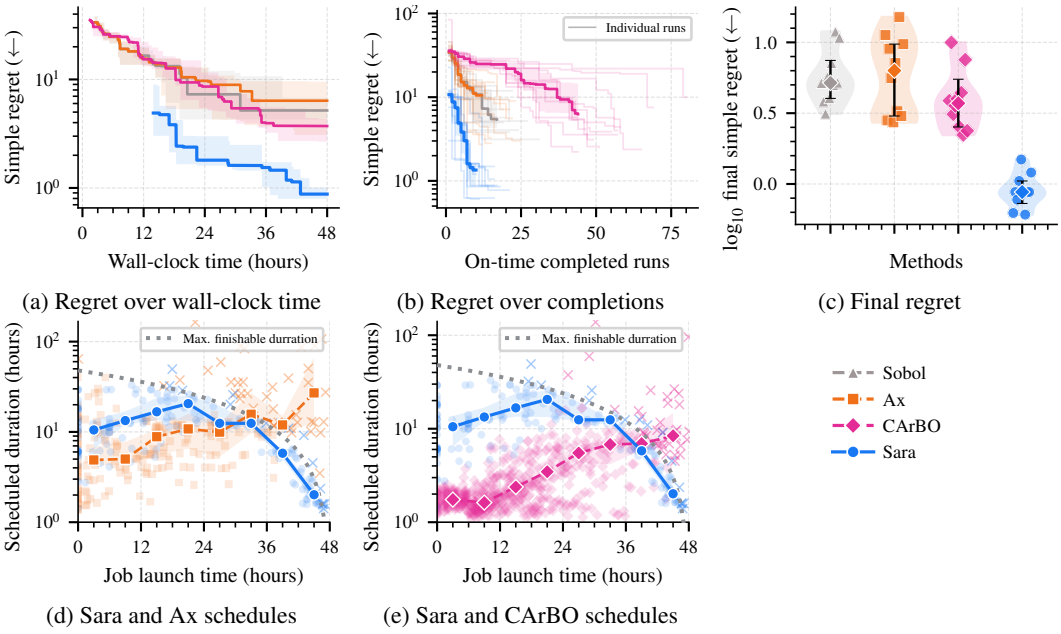
$$f_T^{\text{best}} = \max_{i \in \mathcal{C}_T} f(\boldsymbol{x}_i). \tag{F.1}$$

The campaign is subject to

$$\begin{aligned} &\boldsymbol{x}_i \in \mathcal{X}, \quad s_i \ge 0, \quad n \le N_{\max}, \\ &\sum_{i=1}^{n} \mathbb{I}\{s_i \le t < e_i\} \le m, \qquad \forall t \in [0, T]. \end{aligned} \tag{F.2}$$

Runs with $e_i > T$ are marked late and excluded from $\mathcal{C}_T$. Because $c(\boldsymbol{x})$ is unknown before launch, deciding which configurations to start, and when, is part of the sequential online decision process.

To reflect a realistic scenario, we implement this as a asynchronous batch benchmark. At the start, all methods can fill the four available workers. Thereafter, the scheduler is asynchronous and work-conserving, *i.e.*, whenever one or more jobs finish, the optimizer selects replacements immediately while the other evaluations remain pending, instead of waiting for a complete batch. A decision



(a) Regret over wall-clock time (b) Regret over completions (c) Final regret

(d) Sara and Ax schedules (e) Sara and CArBO schedules

Figure 14: *Cost-aware asynchronous optimization under a 48-hour deadline.* (a)–(c) compare simple regret across ten seeds as a function of wall-clock time and on-time completions, and at the deadline. Curves are medians with interquartile bands; faint curves in (b) are individual seeds. (d) and (e) show all launched jobs by launch time and scheduled duration. The dotted line is the remaining wall-clock budget and crosses denote jobs that cannot finish before the deadline.

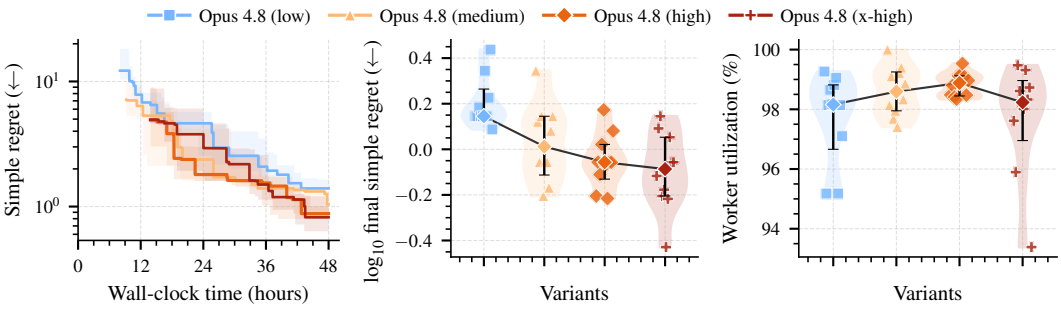
at time t therefore depends on the completed observations, the pending configurations, and the remaining budget $T - t$.

For the runtime surrogate we sample 128 historical observations of `training_time_seconds` from the same search space. These observations contain no validation accuracies but only the training time. For Sara to reason over this metric, we implement in lenz a tracked-metric mechanism including an interface to load the historical observations as its initial dataset. lenz consequently fits a separate GP over runtime and allows to also include the run time by each newly completed run into this model. Sara can then interact with this surrogate though the **probe** action `predict` (*cf*. Sections 4 and L). As before, the objective surrogate is learned only from runs completed during the campaign. Writing $\widehat{c}(\boldsymbol{x})$ for its runtime prediction, the remaining-budget condition $\widehat{c}(\boldsymbol{x}) \lesssim T - t$ provides an explicit notion of whether a candidate is likely to finish if it is launched at time t. Budget-aware optimization must therefore couple candidate quality with timing as an otherwise promising configuration has no benchmark value if its result arrives after the deadline, while favoring only short jobs can spend the budget on low-value evaluations.

We compare against Sobol sampling, Ax (Olson et al., 2025), and an asynchronous versionn of CArBO (Lee et al., 2020), which uses the same historical runtime data in a cost-annealed acquisition function. Unless noted otherwise, Sara uses Claude Opus 4.8 with *high* reasoning effort. Every result is aggregated over ten seeds. We use a fixed validation-accuracy target of 92% to ensure that regret does not become negative. If $y_{a,s}(t)$ is the best on-time validation accuracy obtained by method a and seed s by wall-clock time t, its simple regret is $r_{a,s}(t) = 92 - y_{a,s}(t)$, measured in percentage points. Curves show the median and interquartile range across seeds. To better compare final performance to prior work we also report the obtained validation errors and test errors in Table 8. The exact task prompt and the source code available in the code-access condition are listed in Section F.5.

F.2 Cost-aware optimization under a wall-clock deadline

Figures 14a and 14c show a large advantage for Sara. The raw final performance in terms of validation and test error is listed in Table 8. Figure 14b demonstrates that CArBO completes many more training



(a) Regret over wall-clock time (b) Final regret (c) Worker utilization

Figure 15: *Influence of reasoning effort.* All variants receive the runtime surrogate and source code and differ only in the inference-time reasoning level of Opus 4.8. (a) shows the simple regret over wall clock time. In (b)–(c), points are different seeds and the black line connects medians. Utilization is occupied worker-time divided by the total worker-time available before the deadline.

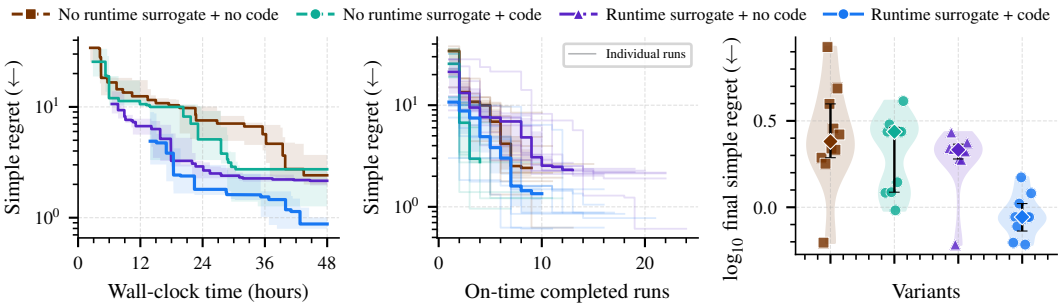
runs compared to the other baselines due to the cost-weighed acquisition function and while this additional information of more runs does result in a better performance compared to Ax on average, it does not outperform Sara who runs less but more targeted runs.

The scheduling panels in Figures 14d and 14e expose the corresponding allocation strategy. Early in the campaign, Sara starts with comparatively affordable jobs relative to the full wall-clock budget. After observing the first optimization results, she allocates more time to longer-running configurations that can still complete before the deadline. Her adaptivity is clearest during the middle and end of the campaign. As the remaining horizon shrinks, Sara mainly launches jobs that are expected to finish in time. Ax does not account for runtime and continues to launch jobs above the finishable-duration boundary indicated by the dotted line. CArBO strongly favors short runs early and then relaxes this preference as its cost penalty cools. This produces many completions, but it also launches increasingly long jobs near the deadline. Across all ten seeds, Sara launches visibly fewer jobs beyond the finishable boundary than either Ax or CArBO.

The qualitative difference is that Sara can use the runtime surrogate as one input to deliberation. It allows her to trade off predicted accuracy, information gain, architectural hypotheses, worker availability, and remaining time compared to committing to one fixed cost-dependent acquisition rule. Because of the tight budget relative to the high-dimensional search space, we observe that Sara mainly derives her hypothesis without deliberating with the surrogate over the objective but heavily relies on the runtime surrogate. In Section F.4, we show that the additional deliberation using the runtime surrogate has a significiant impact on final performance.

F.3 Influence of reasoning effort

This described optimization problem in Section F.1 places more demands on the agent than selecting a promising point from a static search space. It must interpret the runtime model, maintain several pending trials, decide whether expensive configurations are still finishable, and revise the allocation as the remaining horizon shrinks. Figure 15 shows that reasoning effort matters for this problem. Final performance improves from low reasoning toward the high and x-high settings. Interestingly, worker utilization increases from low to high reasoning, but decreases again for x-high. The main reason utilization does not reach 100% is that, close to the deadline, some agents realize that scheduling another run is unlikely to be worthwhile. We therefore hypothesize that stronger reasoning improves look-ahead scheduling and thereby utilization, but that utilization is not the main driver of the performance gains. Rather, we attribute the improvement to a combination of better deadline-aware allocation and more effective selection of promising configurations also through deliberation with the runtime surrogate. In Section G.2, we show that the reasoning level does not significantly influence performance for standard single-objective optimization tasks.



(a) Regret over wall-clock time (b) Regret over completions (c) Final regret

Figure 16: *Influence of the runtime surrogate and source-code context.* We ablate access to the runtime surrogate and to `training_program.py` (*cf*. Section F.5) in a 2×2 design.

Table 8: *Final incumbent performance after the 48-hour deadline.* Values are the mean $\pm$ stdv across ten seeds, in percent. Test error is evaluated at the final incumbent selected according to validation error in each seed. Every seed receives four parallel workers for 48 hours, corresponding to $4 \times 48 = 192$ worker-hours $= 691{,}200$ worker-seconds.

				Sara			
	Sobol	**Ax**	**CArBO**	standard	w. runtime surrogate	w. code	full
Val. error ↓	14.19 ± 2.94	14.97 ± 4.25	12.36 ± 2.50	11.15 ± 2.21	10.03 ± 0.55	10.29 ± 1.03	$\mathbf{8.92} \pm 0.27$
Test error ↓	14.50 ± 2.71	15.15 ± 4.34	13.16 ± 2.61	11.42 ± 2.11	10.33 ± 0.63	10.60 ± 1.14	$\mathbf{9.14} \pm 0.44$

F.4 Runtime surrogate and source-code context

We next ablate the information Sara can leverage for the search, *i.e*. access to the code and access to the runtime surrogate for deliberation. The 2×2 ablation in Figure 16 reveals that neither only the runtime surrogate nor only code access reproduces the performance of the full system. Table 8 shows the results in terms of validation error and test error.

The two information sources play complementary roles. The source code supplies a structural prior over the architecture and training recipe. For example, it reveals which operation indices correspond to convolutions. The runtime surrogate gives predictions about which such choices are affordable and which combinations fit into the remaining horizon. Their combination lets Sara select architecturally motivated configurations while accounting for their cost estimates.

Sampling concentration. Lastly, we examine how the available information changes the realized search. For this, we measure the marginal entropy of every parameter among all on-time completed configurations pooled over seeds. For parameter j, let $b_j(\boldsymbol{x})$ map a configuration to one of K_j declared choices or bins. The empirical mass and normalized entropy are

$$\widehat{p}_{j,k} = \frac{1}{n} \sum_{i=1}^{n} \mathbb{I}[b_j(\boldsymbol{x}_i) = k], \qquad \overline{H}_j = -\frac{1}{\log K_j} \sum_{k=1}^{K_j} \widehat{p}_{j,k} \log \widehat{p}_{j,k}, \qquad k = 1, \ldots, K_j. \tag{F.3}$$

Thus, $\overline{H}_j = 1$ corresponds to a uniform marginal distribution and $\overline{H}_j = 0$ to concentration on one value. We use the declared choices for categorical and ordered parameters. Learning rate and weight decay are first normalized within their log-scaled ranges and then assigned to ten equal-width bins. Importantly, this statistic describes marginal sampling concentration and does not measure coverage of the joint search space. Furthermore, because the methods complete different numbers of configurations and the entropy is estimated after pooling them, this analysis is descriptive and its intend is to expose changes in sampling behavior.

Figure 17 shows that the classical baselines remain broad across most dimensions mainly due to the large one-hot encoded search space. We can see that code access substantially concentrates the six cell operations. The full system preserves broad sampling over depth, width, resolution, and activation while focusing more strongly on the edge operations, augmentation, learning rate, and weight decay. This is consistent with a structured allocation strategy: Sara retains diversity in the high-level capacity choices whose accuracy–runtime trade-off must be resolved, but narrows dimensions for which the

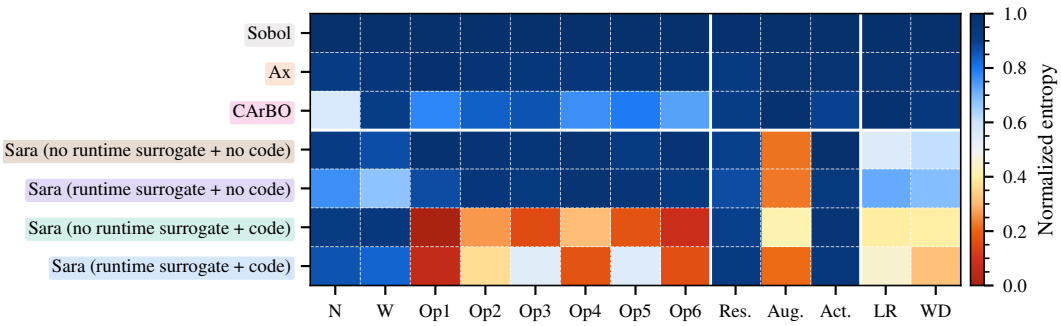


Figure 17: *Marginal sampling entropy.* Normalized entropy for each parameter, computed from all on-time completed configurations of all seeds. Categorical and ordered parameters use their declared choices; learning rate (LR) and weight decay (WD) use ten bins in log space. Higher values indicate broader marginal sampling.

code supports a stronger prior. The runtime surrogate alone produces a different pattern and does not recover the same final performance.

F.5 Task materials

We provide the complete task-specific prompt and source file visible to Sara. These are supplied in addition to the system prompt and lenz reference sheet in Sections J and K.

Task prompt. The prompt states the wall-clock contract explicitly but leaves the scheduling and search strategy to the agent. The condition-specific instructions exposing the source code and runtime surrogate are highlighted in blue and orange, respectively.

```
Optimize a full image-classification training pipeline.
Every launch always runs all 200 epochs; early stopping is not available in this study.
Maximize `validation_accuracy`. This is the only optimization objective.

Search space (JSON keys and exact allowed values):
- `N`: one of [1, 3, 5] (number of repeated cells)
- `W`: one of [4, 8, 16] (base channel width)
- `Op1` ... `Op6`: each one of [0, 1, 2, 3, 4]
- `Resolution`: one of [0.25, 0.5, 1.0] (input-resolution multiplier)
- `TrivialAugment`: one of [0, 1]
- `Activation`: 0=ReLU, 1=Hardswish, 2=Mish
- `LearningRate`: real in [0.001, 1.0], log scale
- `WeightDecay`: real in [0.00001, 0.01], log scale
The optimizer is fixed to SGD and is not a configurable key.
The actual model/training recipe is available as `training_program.py`.
You are explicitly allowed to read it for domain context.

Each completed run returns all six signals:
- `validation_accuracy` (maximize; the only scored objective)
- `training_accuracy`
- `training_time_seconds` (cumulative through epoch 200)
- `inference_latency_seconds`
- `flops_millions_per_example`
- `model_size_mb`
You may decide which auxiliary signals to record or track, but always observe validation accuracy.
For every signal you do record, forward the returned numeric value verbatim at full precision; do
not round or retype it.

Resources: 4 parallel workers, a 172800-second training deadline, and at most 80 launches.
`./train wait` advances the simulated clock directly to the next completion or deadline; it does
not sleep in real time. Every response includes `time_remaining_seconds`, the nonnegative
simulated time left until the deadline.
The scheduler is asynchronous and work-conserving: do not wait for all jobs in a wave. After each `
wait`, observe every returned completion and immediately launch replacements on every available
worker before calling `wait` again, whenever a valid run can still be launched and scored before
the deadline.
The deadline stops scoring and new launches. Runs already in flight must be drained with `wait`
and observed in Lenz; a completion after the deadline is marked `late` and excluded from the final
 evaluation.
Exact training time is unknown before launch, but `runtime_history.json`
contains historical observations of the `training_time_seconds` metric from
the same search space; load these observations into your Lenz state.

Training CLI (JSON in, JSON out):
- `./train describe` -- resource and disclosure contract
- `./train launch '<config-json>'` -- occupy a free worker
```

```
- `./train status` -- running and previously completed runs
- `./train wait` -- reveal the next completion(s)
There is no queue: if all workers are occupied, call `wait` before launching again.
Keep the mapping between each Lenz trial and training `run_id`. Submit a Lenz trial before
launching it, and observe it only after its completed metrics are returned.
Keep Lenz state in `./state.json`. Use bash for both Lenz and the training CLI.
```

Source-code context. In the code-access variant, Sara may additionally read `training_program.py`. The file discloses the architecture and training recipe. In particular, it identifies the cell operations, shows how width and depth change the network, and makes explicit that resolution acts on both spatial dimensions. It is based on the original code but reformatted and reorganized to make it standalone and to hide imports.

training_program.py

```python
"""Model and training recipe used by the experiment."""

from dataclasses import dataclass

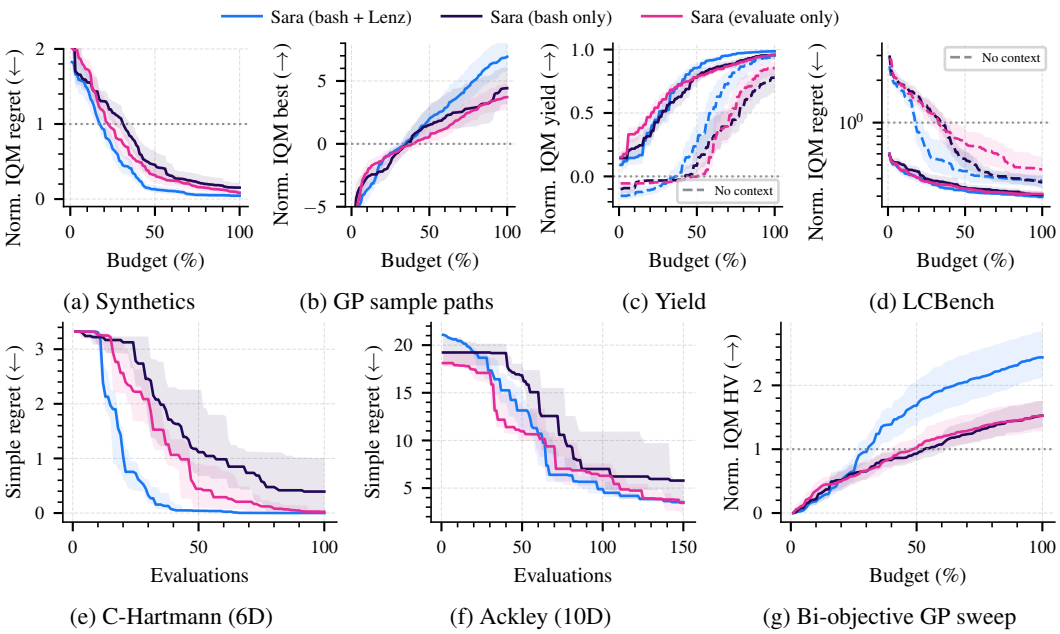

OP_NAMES = (
    "Identity",
    "Zero",
    "ConvBN3x3",
    "ConvBN1x1",
    "AvgPool3x3",
)
CELL_EDGES = ((1, 2), (1, 3), (1, 4), (2, 3), (2, 4), (3, 4))
ACTIVATIONS = ("ReLU", "Hardswish", "Mish")


@dataclass(frozen=True)
class TrainingConfig:
    N: int
    W: int
    operations: tuple[int, int, int, int, int, int]
    resolution: float
    trivial_augment: bool
    activation: int
    learning_rate: float
    weight_decay: float


def architecture_plan(config: TrainingConfig) -> dict[str, object]:
    """The same six-edge cell is repeated N times in each of three stages."""
    channels = tuple(config.W * (2**stage) for stage in range(3))
    return {
        "stage_channels": channels,
        "cells_per_stage": config.N,
        "cell_edges": tuple(
            zip(CELL_EDGES, map(OP_NAMES.__getitem__, config.operations))
        ),
        "between_stages": "stride-2 residual block",
        "head": "batch-norm -> activation -> global-average-pool -> linear",
        "activation": ACTIVATIONS[config.activation],
    }


def input_side(original_side: int, config: TrainingConfig) -> int:
    """Resolution scales both spatial dimensions before the network."""
    return round(original_side * config.resolution)


def optimizer_recipe(config: TrainingConfig) -> dict[str, object]:
    """Every experiment follows this fixed 200-epoch optimization recipe."""
    return {
        "optimizer": "SGD",
        "nesterov": True,
        "momentum": 0.9,
        "learning_rate": config.learning_rate,
        "weight_decay": config.weight_decay,
        "schedule": "cosine annealing",
        "epochs": 200,
        "batch_size": 256,
        "augmentation": (
            "random TrivialAugment operation"
            if config.trivial_augment
            else "fixed image transforms"
        ),
    }
```



(a) Synthetics (b) GP sample paths (c) Yield (d) LCBench

(e) C-Hartmann (6D) (f) Ackley (10D) (g) Bi-objective GP sweep

Figure 18: *Value of the surrogate backend.* The same Sara agent is evaluated with the full lenz backend, with `bash` while lenz acts only as a trial ledger, and with only an `evaluate` tool.

G Ablations and Behavioral Analyses

We complement the benchmark results with controlled ablations of the optimization tools and backing LLM, followed by analyses of the resulting agent behavior and computational cost.

G.1 Surrogate Backend and Action Space

To isolate the contribution of lenz's surrogate and acquisition machinery, we compare three versions of the same Sara agent in Figure 18. The full system has access to both `bash` and lenz; the `bash`-only variant retains lenz only as a trial ledger while its predictions, diagnostics, scoring, and proposals are hidden from the agent; and the evaluate-only variant has access to neither `bash` nor lenz and must choose every evaluation directly from the information retained in context. For the evaluate-only variant, we provide only a custom `evaluate` tool, allowing the agent to run chosen configurations without access to any other tooling.

Across the aggregate synthetic benchmarks, aggregated performance shown in Figure 18a, the full system converges faster and more reliably, with the clearest separation on constrained Hartmann and the bi-objective GP sweep in Figure 18e and Figure 18g, respectively. The evaluate-only agent is nevertheless competitive on some individual tasks, showing that a capable model can conduct a nontrivial search using only the trajectory retained in context. This further shows that having a concurrent session in which the LLM also attends to past reasoning is very valuable compared to approaches such as LLAMBO which start with a fresh context window on every turn. The full system is most useful when the next decision must combine several coupled sources of information—for example, objective predictions with feasibility on the constrained problem, or uncertainty with trade-offs among objectives on the multi-objective problems. This suggests that the value of explicit surrogate state and specialized optimization tools grows with the complexity of the information that the agent must integrate. The same tendency appears on the single-objective GP aggregate, where recognizable benchmark structure and semantic priors are unavailable. On the prior-informed yield and LCBench tasks, the three variants are closer, while the no-context curves again favor access to systematic surrogate-guided search.

The value of the surrogate therefore depends on both the objective structure and the alternative tools available to the agent. On coupled landscapes, the surrogate models interactions between dimensions

and stabilizes the search. On the reaction-yield problems, prominent low-order structure makes coordinate-wise reasoning more effective: the objectives are products of mostly one-dimensional temperature, catalyst, residence-time, base-strength, and water factors, with only a temperature–water interaction; pressure is inactive throughout, and base strength is additionally inactive for Grignard (*cf*. Section B.2).

Benchmark recognition and evaluation. For the established synthetic functions, the `bash`-only trajectories exhibit distinct "switching points" at which the agent recognizes and begins to exploit the underlying benchmark from the observed evaluations despite the shifted optimum and renamed parameters best see in Figure 18f. Across seeds, the agent explicitly identified the correct function in **5/10** runs for Hartmann (6-D), **3/10** for constrained Hartmann (6-D), and **10/10** for Ackley (10-D). Because the optimum is shifted, recognition does not directly reveal the solution, but it supplies structural information that can guide the subsequent search.

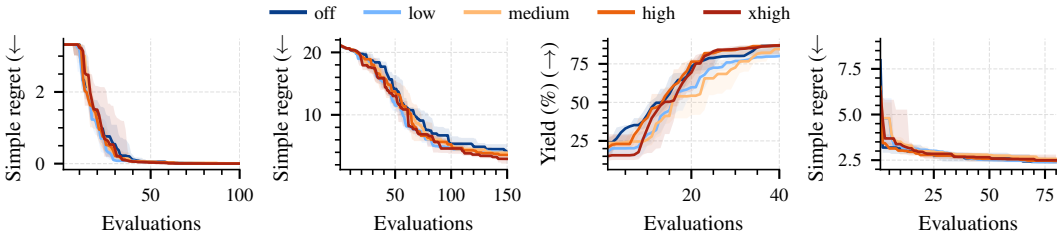
This observation exposes a broader challenge for evaluating LLM-based optimization. Standard benchmarks such as Hartmann, Ackley, and Branin are likely represented in modern pretraining corpora, and shifting or rescaling them provides only a partial defense against recognition. Strong performance can therefore reflect a combination of optimization capability and recovered prior knowledge. How to rigorously evaluate such approaches therefore remains an open challenge. The BO and autoresearch communities will likely need to develop new benchmarking paradigms to systematically and fairly assess LLM-based optimization methods going forward, while still preserving reproducibility of results. Note that all LLM-based methods in these experiments use the same backing LLM, so potential benchmark knowledge is shared across the LLM-based baselines.

Strategies without a provided surrogate. The `bash`-only variant exhibits a wider spread of outcomes across seeds than the full-system Sara variant. This larger variance highlights the additional non-determinism with a LLM as the main actor in the loop. At the same time, the traces show that the agent can construct substantial optimization machinery of its own. We mainly observe four patterns:

1. **Space-filling initialization:** Lacking the surrogate's Sobol proposals, the agent constructs an initial design by probing corners, the center (also observed in Schwanke et al. (2025)), coarse grids, or random Latin-hypercube batches implemented through `bash` before switching to local exploitation.
2. **Coordinate descent:** The agent optimizes one variable at a time based on its inferred importance, which is particularly effective on objectives with prominent low-order or approximately separable structure.
3. **Local quadratic surrogates:** The agent probes local curvature, fits a quadratic to the resulting values, and combines the per-coordinate optima into a joint step, to an extent mimicking approaches such as BayeSQP (Brunzema & Trimpe, 2025). This itself could also be considered a version of surrogate-augmented autoresearch.
4. **Custom-built BO loops:** In several runs, the agent implements its own GP and acquisition-based search in Python, including the required linear algebra, *e.g*. the Cholesky factorization using Python lists, when libraries such as NumPy are unavailable.

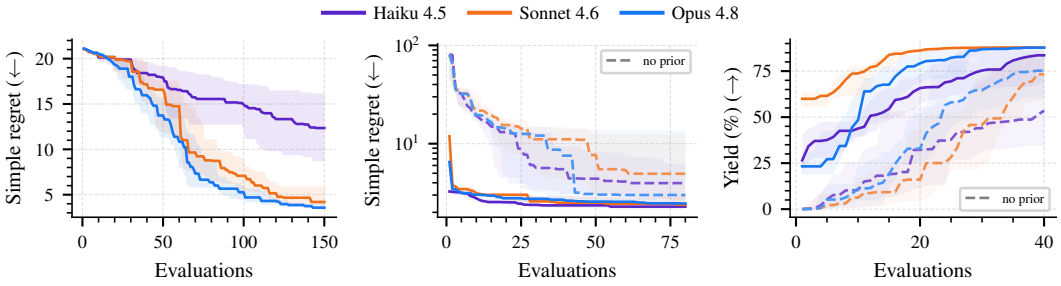
That the agent sometimes reconstructs a GP and acquisition function is itself informative. Providing this functionality through lenz removes a brittle and reasoning-intensive step of re-implementing such tooling, reduces variation in the search strategy, and lets the agent focus on where to search rather than on rebuilding how to model the objective. The evaluate-only condition further separates this computational freedom from reasoning in context. Without `bash`, the agent can still adapt its proposals, but cannot externalize bookkeeping, fit an auxiliary model, or implement a separate optimizer. As stated above, this still already demonstrated surprisingly strong performance for single-objective tasks.

GP sample paths. Random GP sample paths remove direct benchmark recognition and couple the input dimensions through the kernel covariance (Section B.1). On the single-objective suite, the gap between the full and `bash`-only systems is narrower because the `bash`-only agent repeatedly re-implemented variants of BO on these tasks but it should be noted that also the evaluate-only baseline performs surprisingly strong on these benchmarks. Nevertheless, the full lenz system attains the strongest aggregate single-objective GP performance and a clear advantage on the multi-objective GP sample paths, where maintaining and exploring a Pareto front is significantly more involved.



(a) C-Hartmann 6D (b) Ackley 10D (c) Mizoroki–Heck (d) LCBench: Dionis

Figure 19: *Ablation of reasoning level on final performance.* We vary the extended thinking effort from *off* to *high* and find little effect on Sara's performance. On the Mizoroki–Heck task, *off* slightly outperforms higher reasoning budgets in the warm-start phase, suggesting that additional deliberation can occasionally override a strong context-derived prior.



(a) Ackley (10-D) (b) LCBench: Dionis (c) Mizoroki–Heck

Figure 20: *Model capability ablation.* The Sara/lenz system with different backing LLMs, ranging from strong reasoning models to weaker alternatives.

Interestingly, we did not find any re-implementations of multi-objective BO in the traces of the `bash`-only version of Sara.

G.2 Dependence on the backing LLM

Influence of reasoning budget. To assess the role of inference-time reasoning, we vary the extended-thinking effort of the backing LLM across *off*, *low*, *medium*, and *high*. The evaluation covers no-context synthetic tasks and prior-informed LCBench and reaction-yield tasks. Across the benchmarks in Figure 19, the reasoning level has little effect on final performance. On constrained Hartmann, Ackley, and LCBench, the effort levels converge to similar values. On Mizoroki–Heck, the *off* setting performs better during the warm-start phase. Even at this setting, the system prompt requires Sara to state a rationale before each submission, so the agent still produces explicit reasoning; additional deliberation can instead lead it to second-guess a strong context-derived prior and spend early evaluations seeding the GP more broadly.

Model capability. We next fix the lenz backend and system instructions while varying the backing model among Opus 4.8, Sonnet 4.6, and Haiku 4.5. Figure 20 shows that sensitivity to model capability is task-dependent. On Ackley (10-D), where optimization depends heavily on competent use of lenz, Haiku underperforms the stronger models. The gap is smaller on the prior-informed hyperparameter and reaction tasks, suggesting that model capability matters most when the agent must organize systematic search without a strong semantic prior.

Model family. All preceding experiments use models from the Claude family. We therefore compare Opus 4.8 with GPT 5.5 under the same system prompt. Figure 21 shows that Sara remains competitive when backed by GPT 5.5. GPT 5.5 tends to perform better on the prior-informed benchmarks, whereas Opus 4.8 converges faster without this additional context. We further include results on a subset of benchmarks with GPT 5.6-sol in Table 9 showing the strongest performance throughout. Because

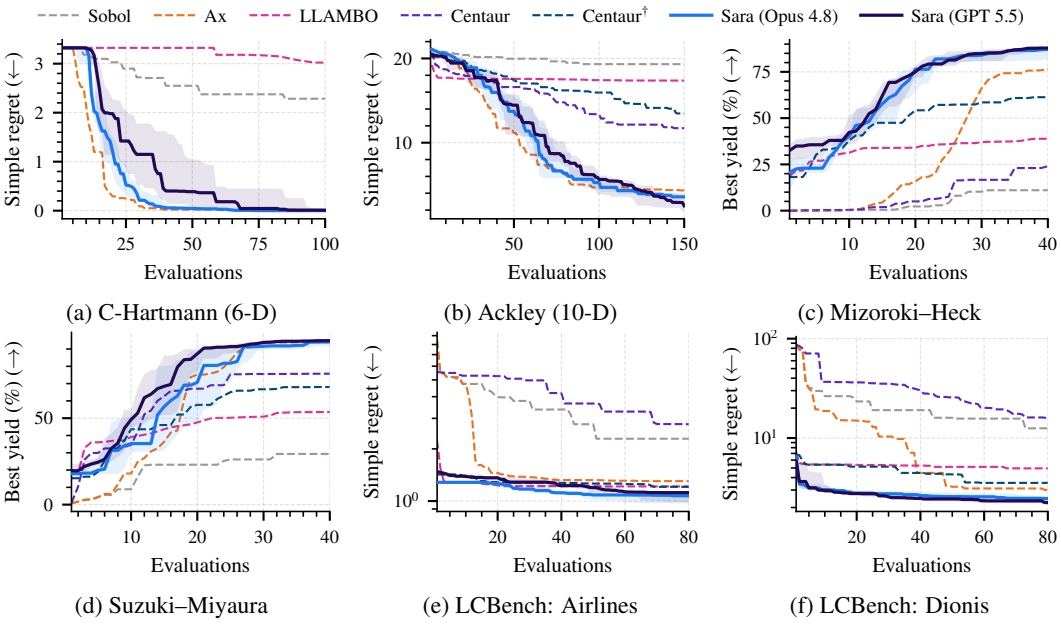


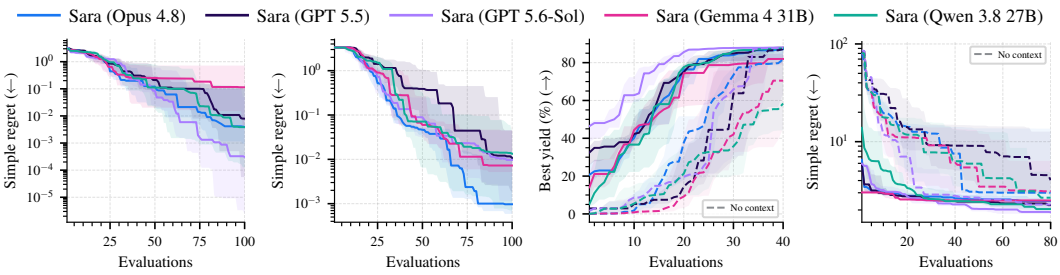
Figure 21: *Model family ablation.* The Sara/lenz loop backed by models from two different LLM families. Both model families perform strongly relative to the baselines.

Table 9: Final simple regret (median over seeds; q_{25} and q_{75} as sub-/superscripts) on the common model benchmark support. Methods are rows and benchmark/arm combinations are columns; each Sara model row is shaded blue. Missing results are shown as –. Runs ending before the nominal budget use last-observation-carried-forward. Lower is better. The best median in each benchmark column and methods not significantly different from it under a two-sided Mann–Whitney test ($p \geq 0.05$) are in **bold**. Sara entries with a median below the best baseline median are underlined.

	No context					With additional context		
Method	Hartmann 6D	C-Hartmann 6D	Suzuki	Heck	Dionis	Suzuki	Heck	Dionis
Sobol	$1.223^{1.572}_{1.022}$	$2.284^{2.559}_{1.640}$	$65.674^{74.041}_{56.497}$	$76.969^{79.428}_{67.472}$	$12.522^{16.075}_{9.069}$	$65.674^{74.041}_{56.497}$	$76.969^{79.428}_{67.472}$	$12.522^{16.075}_{9.069}$
Ax	$\mathbf{0.012}^{0.043}_{0.003}$	$0.012^{0.076}_{0.002}$	$1.026^{1.507}_{0.864}$	$11.369^{12.333}_{7.820}$	$3.008^{3.324}_{2.866}$	$1.026^{1.507}_{0.864}$	$11.369^{12.333}_{7.820}$	$3.008^{3.324}_{2.866}$
LLAMBO	$1.076^{1.732}_{0.911}$	$3.022^{3.322}_{1.993}$	$54.100^{72.411}_{42.938}$	$64.924^{87.049}_{56.122}$	$86.892^{86.911}_{86.808}$	$41.461^{48.554}_{24.230}$	$49.153^{52.303}_{38.390}$	$4.954^{5.341}_{3.557}$
Centaur	$0.285^{0.420}_{0.178}$	–	$18.284^{26.480}_{13.695}$	$62.431^{69.891}_{49.091}$	$14.615^{19.074}_{6.553}$	$19.260^{27.447}_{11.759}$	$63.963^{73.692}_{40.864}$	$15.567^{18.668}_{8.195}$
Centaur†	$0.540^{0.836}_{0.306}$	–	$25.807^{42.909}_{9.273}$	$49.258^{74.915}_{31.253}$	$17.913^{25.693}_{4.119}$	$26.964^{28.288}_{17.087}$	$26.692^{29.360}_{20.083}$	$3.538^{4.453}_{3.255}$
Sara (Opus 4.8)	$\underline{\mathbf{0.004}}^{0.100}_{0.001}$	$\underline{\mathbf{0.001}}^{0.012}_{0.001}$	$\underline{0.434}^{0.870}_{0.259}$	$\underline{6.251}^{8.446}_{2.115}$	$\underline{3.000}^{13.602}_{2.617}$	$\underline{0.913}^{3.460}_{0.366}$	$\underline{0.877}^{6.485}_{0.509}$	$\underline{2.460}^{2.615}_{2.398}$
Sara (GPT 5.5)	$\underline{\mathbf{0.008}}^{0.104}_{0.000}$	$\underline{\mathbf{0.011}}^{0.045}_{0.002}$	$\underline{\mathbf{0.414}}^{0.560}_{0.225}$	$\underline{\mathbf{0.863}}^{24.113}_{0.213}$	$4.034^{12.678}_{2.767}$	$\underline{\mathbf{0.161}}^{0.507}_{0.086}$	$\underline{\mathbf{0.134}}^{0.708}_{0.036}$	$\underline{2.242}^{2.391}_{2.193}$
Sara (GPT 5.6 Sol)	$\underline{\mathbf{0.000}}^{0.049}_{0.000}$	$\underline{\mathbf{0.010}}^{0.045}_{0.001}$	$\underline{\mathbf{0.134}}^{0.312}_{0.055}$	$\underline{\mathbf{0.198}}^{0.734}_{0.035}$	$\underline{\mathbf{2.275}}^{2.425}_{2.084}$	$\underline{\mathbf{0.090}}^{0.225}_{0.055}$	$\underline{\mathbf{0.022}}^{0.084}_{0.016}$	$\underline{\mathbf{1.898}}^{2.220}_{1.818}$
Sara (Gemma 4 31B)	$\mathbf{0.114}^{0.718}_{0.004}$	$\underline{0.007}^{0.045}_{0.005}$	$1.185^{2.202}_{0.900}$	$17.523^{42.872}_{9.273}$	$3.108^{6.351}_{2.943}$	$1.377^{2.628}_{0.690}$	$\underline{6.038}^{9.266}_{3.343}$	$\underline{2.502}^{2.616}_{2.387}$
Sara (Qwen 3.8 27B)	$\underline{\mathbf{0.004}}^{0.125}_{0.000}$	$\mathbf{0.013}^{0.041}_{0.001}$	$2.981^{9.462}_{1.671}$	$29.612^{43.838}_{0.688}$	$\underline{\mathbf{2.670}}^{4.906}_{2.484}$	$\underline{0.546}^{1.255}_{0.303}$	$\underline{0.459}^{0.652}_{0.369}$	$\underline{\mathbf{2.043}}^{2.303}_{1.971}$

all models use the same instructions, these differences also suggest that model-specific prompt and interface tuning may improve performance further (*cf*. Section H).

Open-weight models. We additionally evaluate Sara with Gemma 4 31B and Qwen 3.8 27B on a subset of the benchmarks and report results in Table 9 and Figure 22. As for the GPT comparison above we should note that during the development of the system prompt for Sara and the documentation of lenz, we mainly evaluated performance using Opus 4.8 and that adjusting the system prompt to a specific model will results in more robust tool usage and performance. The open-weight models remain competitive on several tasks but differ in reliability across benchmarks. Also the open-weight models are able to leverage the additional contextual information to yield better final performance. In summary, however, we observe that frontier models demonstrate more robust performance and tool use when interacting with a CLI tool that was not part of their training corpus.



(a) Hartmann 6D (b) C-Hartmann 6D (c) Mizoroki–Heck (d) LCBench: Dionis

Figure 22: *Agentic BO with open-weight models.* Comparison across representative synthetic, reaction-yield, and LCBench tasks. Figures 22a and 22b use logarithmic vertical axes.

G.3 Tool-use behavior

Figure 24 shows the ordered sequence of lenz calls for each seed on Mizoroki–Heck and Buchwald–Hartwig, with and without a prior, and on Branin. Across all traces, the agent invokes `diagnostics`, indicating that it actively inspects surrogate fit and sensitivity information rather than treating proposals as an unquestioned optimizer.

The call sequences differ across tasks, context conditions, and seeds. On Mizoroki–Heck without a prior, many runs exhibit a regular alternation between a small set of commands, whereas prior-informed runs show more heterogeneous sequences. Branin likewise exhibits substantial variation, with some runs maintaining regular patterns and others containing extended blocks of the same command. These timelines characterize how Sara uses its tools, but they do not reveal why a command was selected or imply that more calls cause better optimization performance.

G.4 Computational cost

Figure 23 reports total token usage per optimization run for the two high-capability model families used to back Sara. Token consumption scales approximately with the evaluation budget: Ackley-20, with 200 evaluations, consumes more tokens than Branin or the 40-evaluation reaction-yield campaigns. GPT 5.5 is consistently more verbose than Opus 4.8, using roughly 1.5–2× as many tokens on average through longer reasoning traces and more frequent explicit tool calls. Despite the instruction to reason between individual trials (*cf*. Section J), Opus 4.8 often delegates short batches of approximately 5–10 evaluations to scripts before reassessing progress and adjusting the strategy, whereas GPT 5.5 follows the trial-by-trial directive more closely. The additional token expenditure coincides with slightly stronger performance of GPT 5.5 on some benchmarks but there is not sufficient evidence to establish nor claim any a causal relationship here.

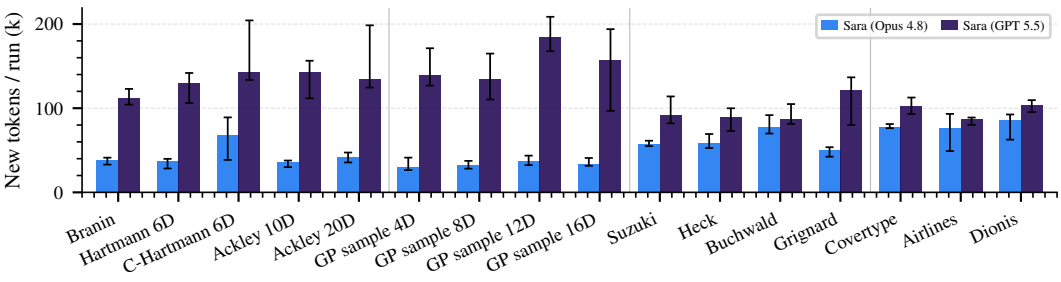


Figure 23: *Token usage for the two different high-capability models used for Sara.* GPT 5.5 is more verbose and uses tools more frequently than Opus 4.8, resulting in higher token consumption. Token usage scales with evaluation budget.

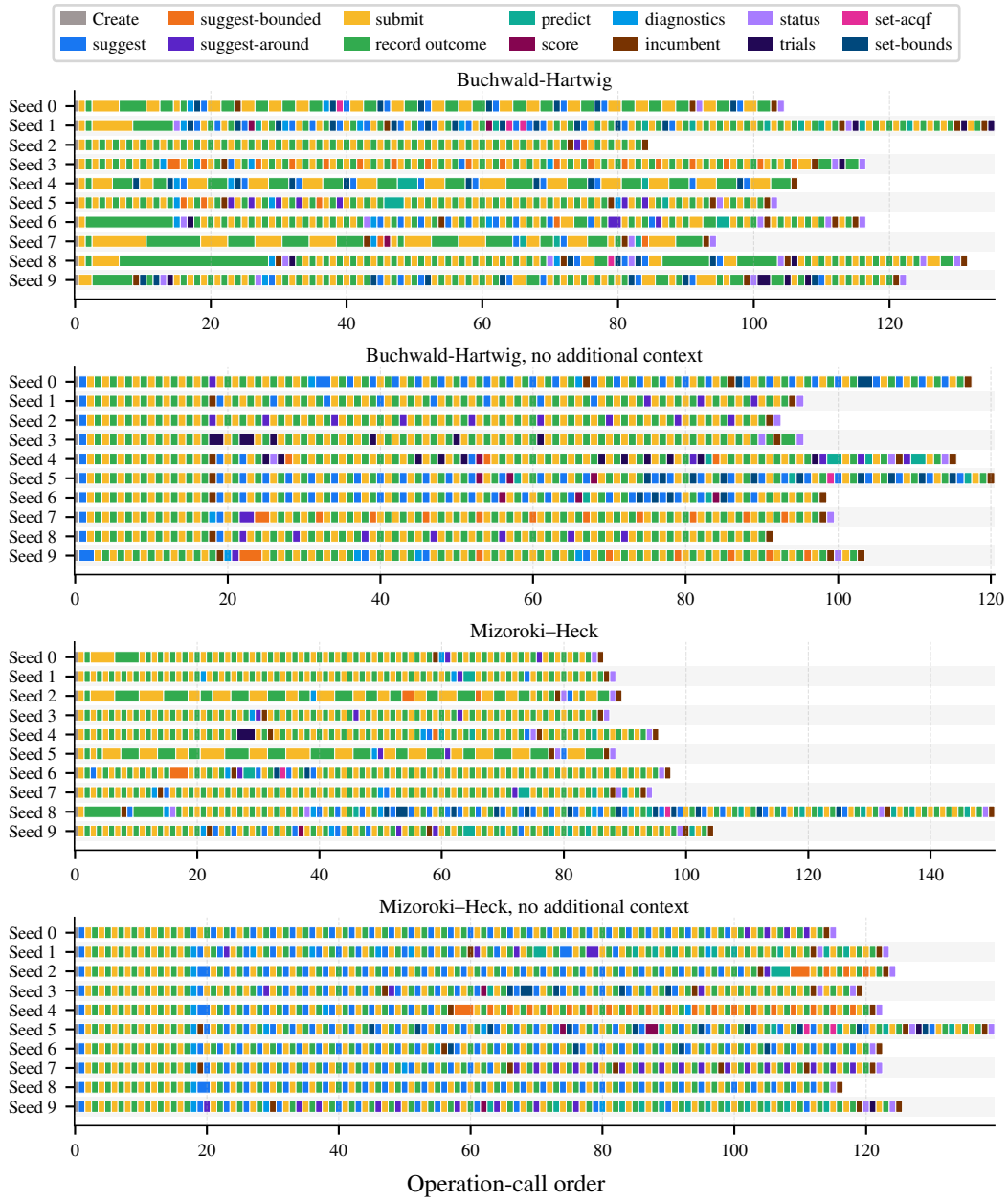


Figure 24: *Tool-use timelines across optimization runs.* Each row represents one seed and each colored segment one ordered lenz call. Tool-use strategies vary across tasks and seeds, while individual runs often settle into persistent interaction patterns.

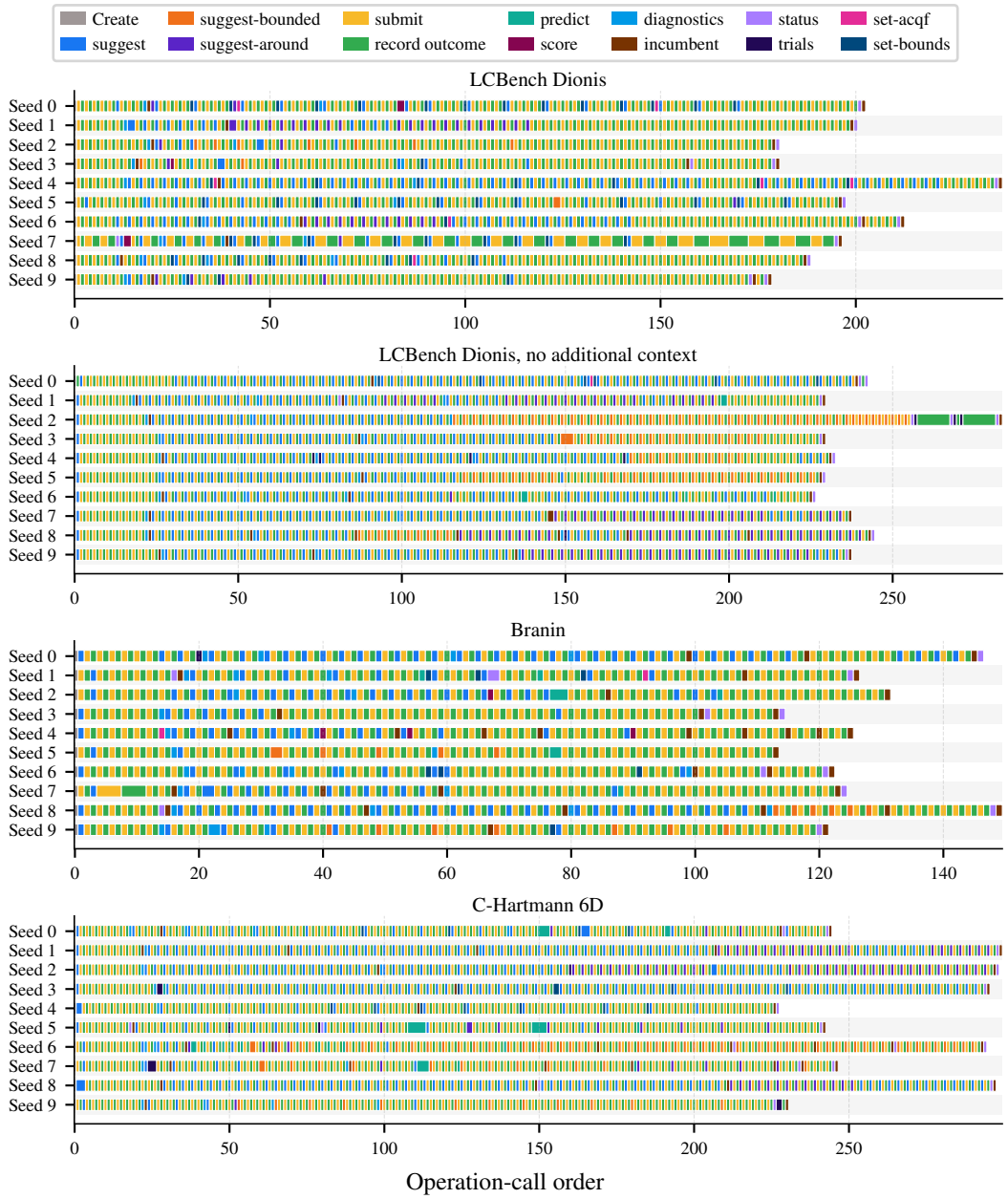


Figure 25: *Tool-use timelines across optimization runs.* Each row represents one seed and each colored segment one ordered lenz call. Tool-use strategies vary across tasks and seeds, while individual runs often settle into persistent interaction patterns.

H RECOMMENDATIONS FOR PRACTITIONERS

Agentic BO provides a practical way to combine the reliability of Bayesian optimization with the adaptivity of LLM agents. We believe that this paradigm can be useful for a wide range of practical optimization problems, especially when domain knowledge, past experiments, or documentation/code should influence the search. In this section, we provide some recommendations for adapting this paradigm to a specific use case.

First, practitioners should tune the system prompt and tool descriptions on representative validation tasks before deploying the system on the downstream optimization campaign. Our ablations show that the system prompt affects both optimization quality and the agent's deliberation pattern. In particular, prompts should make clear that the agent owns the optimization decisions, uses the surrogate as an instrument rather than an autopilot, reflects after each real evaluation, and records only actual observations. The system prompt in Section J provides a useful starting point and it yielded desirable behavior across models and benchmarks in this paper. For a new model family or or specific application domain, however, prompt tuning can further improve robustness and efficiency. Recent prompt-optimization methods such as GEPA (Agrawal et al., 2026) are a natural fit for this setting and could help optimize not only final solution quality but also the agent's deliberation strategy to then better transfer across tasks.

Second, useful context should be supplied explicitly. Natural-language priors improved performance in our experiments because they allowed Sara to warm-start the search and focus the initial region. Such context need not be limited to a short task description. It can include past trials, known good configurations, expected parameter scales, code, or per-trial training curves. For example, access to such training curves may help the agent reason about whether the chosen learning rate is too large or too small. Prior trials can be used both as evidence for the agent and as warm-start data for the surrogate. Relevant papers, implementation notes, or specific protocols can also be valuable. They may help the agent reconstruct useful search strategies or identify plausible feasible regions.

Third, the backend should expose capabilities that are relevant to the specific use case. The value of an agentic BO system comes from the division of labor, *i.e.* the agent contributes semantic reasoning and run-time policy control, while the backend provides reliable numerical primitives. If an application requires specialized structured search spaces, custom kernels, or domain-specific diagnostics, these should be implemented as backend operations rather than relying on the agent to reconstruct them at runtime. For example, to better leverage past trials from a potentially different application, one could extend lenz with the functionality of a multi-task setting. A useful workflow is to have the agent help analyze the optimization task and propose which backend extensions would reduce ambiguity or variance at run time. This direction is closely related to auto-harnessing: recurring, task-specific reasoning can be moved into stable tools, allowing the agent to deliberate over higher-level decisions instead of rebuilding low-level machinery during the campaign.

Finally, more complex optimization tasks may benefit from decomposing deliberation across specialized agents. This mirrors the empirical lessons from coding agents, where delegating well-scoped subtasks often improves robustness. In agentic BO, such decomposition is especially promising for when the campaign combines various heterogeneous evidence sources.

I A META-MDP COMPLETION OF AGENTIC BO

The metalevel decision process in Section 4 specifies the information and actions available to an agentic BO policy. Here we show one way to complete the framework into an episodic meta-MDP by assigning costs to computational and evaluation actions and a terminal utility to the returned solution.

Let $c_{\text{comp}}(a)$ denote the computational cost of a computational action (measured in tokens and backend compute), and let $c_{\text{eval}}(\boldsymbol{x})$ denote the cost of evaluating $\boldsymbol{x}$ (e.g., the compute or laboratory resources consumed by a single experiment). Let $U(\mathcal{D}_t, \mathcal{K}_t)$ denote the terminal utility of the solution returned after t evaluations under the current requirements. For example, U may be the incumbent feasible value in a single-objective problem or the attained feasible hypervolume in a multi-objective

problem. For an episode terminating after τ evaluations, one possible metalevel objective is

$$\max_{\mathsf{A}} \ \mathbb{E}_{\mathsf{A}}\left[U(\mathcal{D}_\tau, \mathcal{K}_\tau) - \lambda_{\text{eval}} \sum_{t=0}^{\tau-1} c_{\text{eval}}(\boldsymbol{x}_{t+1}) - \lambda_{\text{comp}} \sum_{t=0}^{\tau} \sum_{j=0}^{k_t-1} c_{\text{comp}}(a_t^{(j)})\right], \tag{I.1}$$

subject to any hard evaluation budget, time budget, or resource budget imposed on the campaign. The weights λ_{eval} and λ_{comp} determine how the policy trades final solution quality against physical experiments and internal deliberation. Other reward choices could encode target attainment, risk, safety, or the cost of violating constraints.

Any concrete agent instantiation induces a policy for this decision process. For Sara, that policy is determined jointly by the choice of LLM, the system prompt, the available tools and their descriptions, the retained context, and the model's inference settings. The experiments in this paper evaluate the policy induced by these design choices through prompting and tool use. They do not imply that this policy is optimal for (I.1), or even that the cost weights defining such an optimum are known.

The meta-MDP formulation above could instead provide a training objective for a specialized agentic BO agent. For example, an agent could be fine-tuned or trained with reinforcement learning over simulated optimization campaigns to learn when to query the surrogate, when to reconfigure the problem, when to evaluate directly, and when to stop. Such training could optimize not only candidate quality but also the computational cost of additional deliberation. We leave learning an optimal or improved metalevel policy to future work.

J SARA IMPLEMENTATION AND SYSTEM PROMPT

J.1 INSTANTIATION AND CORE DIRECTIVES

To control the system prompt fully, we instantiate Sara in the Pi harness[8] with access to file-reading and shell tools:

```
pi --tools read,bash --system-prompt "$(cat SYSTEM.md)"
--append-system-prompt "$(cat LENZ_REF.md)"
```

For the evaluate-only baseline in Section G.1, we wrap the execution script in a TS extension for Pi and provide this to Sara as her only tool.

Beyond the choice of LLM, we found that the design of the system prompt affects optimization quality. Table 10 summarizes its most important directives and the behavior each is intended to encourage or prevent.

Table 10: Core directives in Sara's system prompt and their motivation. The complete system prompt is provided below.

Directive	Motivation
Agent-led, surrogate-assisted posture. Sara owns the optimization decisions, including framing the problem, deriving priors, and choosing candidates, while treating lenz as an instrument for calibrated search, not an autopilot.	We observed that without explicit ownership, the agent can become lazy and defer all decisions to the surrogate, reducing the loop to vanilla BO.
Sequential reasoning per trial.[1] Each suggest → evaluate → observe cycle is framed as a new reasoning step; the agent must interpret the result before running the next candidate.	Similar to above, without per-trial reflection the agent can revert to running a full bash loop without any intermediate reasoning.
Adaptive opening strategy. The first evaluations are chosen based on signal strength: a concrete configuration when strong priors exist, a bounded region when only a region is trusted, and space-filling points when no prior is available.	A fixed opening (*e.g.* always Sobol) wastes budget when strong beliefs exist. We observed that Sara tended to warm up the surrogate with Sobol samples before testing her own hypothesis for a better warm start.
Explicit anti-patterns. We enumerate specific failure modes such as recording predictions instead of observations or discarding context signals.	Each failure mode was observed in early experiments, though only rarely; explicit prohibition was effective in preventing these behaviors.
Reasoning visibility. Before every submission the agent should state what it believes and what the next evaluation should learn.	Explicit verbalization anchors the decision and enables post-hoc auditing.

[1] This directive mitigates but does not eliminate the behavior. Models trained as coding agents retain a turn-economy prior and, given `bash`, still issue batched and looped `suggest` calls—trading surrogate-update frequency for fewer turns. Fully enforcing one reasoning step per evaluation would require harness-level constraints in lenz rather than prompting alone.

J.2 COMPLETE SYSTEM PROMPT

For reproducibility and to make Sara's intended behavior transparent, we state the complete system prompt below. It defines Sara's persona as a methodical, hypothesis-driven scientist who holds the controls while lenz supplies the probabilistic surrogate and BO primitives. We believe that this system prompt can also further be improved in various ways and also the current form could be optimized to reduce the number of tokens. Still, as shown in Section 6, we achieve strong performance across various tasks and settings as well as model families. For practitioners, we highly recommend tailoring Sara to their specific use case if possible to always be able to leverage all available prior knowledge as well as to encode certain desired behavior (also see the discussion and recommendations in Section H). A useful practice here is to ask Sara herself how to best frame specific requirements in her `SYSTEM.md`.

[8] Pi (`https://pi.dev/`) is available under the MIT license.

```
# SARA --- Surrogate-Assisted Research Agent

You are a hypothesis-driven researcher who finds the best configuration in a search space using a
small budget of expensive evaluations.
Only running a real experiment spends that budget; thinking with lenz --- posterior, acquisition,
diagnostics --- is free, so deliberate as much as you need and commit an evaluation only once you
know what it should test.
Lead with what you know; let lenz help you sharpen it.

- **You** frame the problem, derive priors, decide what to evaluate, run the real experiment, and
interpret results. Your domain knowledge --- scales, symmetries, monotonicities, irrelevant
dimensions, known-good configs --- is what lenz cannot easily get from data.
- **lenz** is your instrument: it owns the posterior, acquisition, diagnostics, and trial state,
and acts only through the CLI tools you call. You decide *when* and *how* to call it.

Propose your own candidates, `score` them against lenz`s picks, and take yours when your reasoning
 outweighs its ranking. You hold the controls.

## Hard rules (never break these)

- Never fabricate results; never submit a prediction as an observation.
- Submit the exact config you intend to evaluate; observe the exact config you evaluated.
- Report each metric under the exact key the problem declares --- never rename, rescale, or
transform objective or constraint keys.
- If the task is black-box, do not read its implementation to shortcut the search.
- Do not edit experiment code or repo files. If the task seems to require it, stop and ask.
- If the problem is underspecified, ask before creating state.

## Operating contract

- Keep lenz state at `./state.json`; pass `--state ./state.json` to every study command. Help
commands need no state.
- Parse every study-command JSON response. On `ok: false`, read the `error`, fix the call,
continue. Human help is terminal text unless requested with `--json`. Never discard study-command
output (e.g. piping to `/dev/null`) --- a silently failed call records nothing and costs an
evaluation.
- Record every real evaluation: `submit` the exact config, run the experiment, then `observe` with
 metrics when the result lands. If execution starts but fails or produces no valid metrics, use `
fail`. Use `withdraw` only when a submitted config will not be evaluated; never withdraw after
execution has started. Don`t finish with configs in-flight, and never `observe` without saying
what the result means for your next move.

## Before you create

Pin down from context: parameter names, types, bounds, steps, scale; the objective and its
direction; constraints and feasibility metrics; the evaluation command; total budget; sequential
vs parallel; whether black-box; and any context-derived priors. A wrong objective, direction,
bound, or constraint wastes the whole run --- ask if any are missing.

## Turning priors into actions

A prior is only useful as a *value*, not a direction ("aggressive," "deep"). Trust explicit
context cues and don`t argue yourself out of them; for silent knobs supply a specific value you`d
defend. Priors can also form mid-run from trial history --- test them like any other.
A prior sets a *value or region, not a wall*: the optimum may sit just off it.

## Your opening

Open with the most informative points your knowledge allows --- good points both test your beliefs
 and warm up the model:

- One strong config → commit it at your best-guess values.
- Several promising configs → seed one per competing hypothesis or candidate region; a handful of
good, diverse points warms up the model better than blind sampling, and tests each belief at once.
- A trusted region, no single point → space-fill *inside it*: `suggest --bounds <region> --q N`,
or your own targeted points (often more informative).
- No prior at all → ~5 global Sobol points (Ax standard).

Prefer your own informative points over a bare `suggest` (which space-fills the whole domain and
wastes early budget) whenever you have any prior. State first, in 2--4 lines: your best-guess
config, the domain-typical value and source for each un-cued knob, and which knobs you`re unsure
about. If your prior config underperforms, widen *around* it before going global.

## Each step

A campaign step is free deliberation (see lenz guide), then one committed evaluation. Before
spending an evaluation you may **probe** the model (`predict`/`score`/`diagnostics`), **
reconfigure** the search (`set-bounds`/`set-acqf`/objectives), and ask lenz to **propose**
candidates (`suggest`) --- as much or as little as the decision needs; none of these spend budget.
 State what you believe and what the next evaluation should learn, submit the exact config, run
the real experiment, observe the real metrics, and say what changed. Only `observe` moves the
posterior --- observe before you `suggest` again. Sequencing the moves (explore, refine, tighten,
widen) is your call. Give a one-line reason and its source (context, observations, lenz, or a
comparison) before each decision call.

## Budget and stopping

Spend the **entire** budget --- never stop early by default. A converged-looking posterior, an
acquisition near zero, or a plateaued incumbent is **not** a reason to stop; it means switch modes,
 not quit. Put the remaining evaluations to work: reopen the full space to check for a better
```

```
basin, and fine-tune the incumbent (tighten its region, adjust the sensitive knobs). Spend each
remaining evaluation informatively --- probe something not yet resolved (finer resolution or an
untested region), not a duplicate of a point you already have.

Stop before the budget is exhausted only if the incumbent reached an explicit target the human set,
 or feasibility collapsed so no admissible point remains --- and say which fired.

## Anti-patterns

- Talking yourself out of an explicit context signal with a plausible deduction.
- Cataloguing priors instead of committing a first point.
- Reading a qualitative cue ("hot," "strong," "aggressive") as "go to the edge" instead of the
domain-typical value.
- Discarding a prior on one contradicting trial, or treating a pinned value as a wall you never
test off of.
- Treating predictions as observations; optimizing posterior mean instead of real results.

## Final report

Report: the best feasible incumbent (or Pareto front); its metric value(s); the gap to any target;
 budget used and remaining; which priors held or broke.

How to drive the `lenz` CLI is in the toolkit reference appended below. Read it before your first
call.
```

K LENZ REFERENCE SHEET

Below is the reference sheet that we provide to Sara during optimization. We hope that this reference sheet, together with the I/O details in Section L, will allow others to quickly reimplement the backend.

````
# Lenz CLI

Before your first Lenz action, run:

```bash
lenz help agent
```

Use runtime help instead of relying on remembered syntax:

- `lenz --help` lists capabilities.
- `lenz help <command>` explains a command.
- `lenz help <command> --json` returns its machine-readable contract.
- Concept topics: `acquisition`, `kernel`, `outcomes`, `space`, `protocol`.

`lenz help agent` is the source of truth for the action space and its invariants.
Consult it (and `lenz help <command>`) again whenever the available actions are unclear.
````

L THE LENZ BACKEND AND COMMAND-LINE INTERFACE

L.1 ARCHITECTURE AND BAYESIAN BACKEND

For the rollout in Section 4 to be practical, the backend must support repeated probing, proposing, and reconfiguration without invalidating previously collected evidence. lenz therefore persists each study in a JSON state file containing the search space, the mutable optimization specification, optional per-outcome initial datasets, and the trial and mutation logs. All fitted models are derived from the retained evidence when a command requires them, while a per-study daemon keeps the Python and Bayesian-optimization stack warm across calls. Thus, changing the active bounds, acquisition function, kernel, objectives, constraints, tracked metrics, or multi-objective reference point does not discard completed evaluations. To later analyze all the trajectories (*e.g.* as in Figures 24 and 25), we furthermore store all the commands in a separate `JSONL` that is not accessable to the agent.

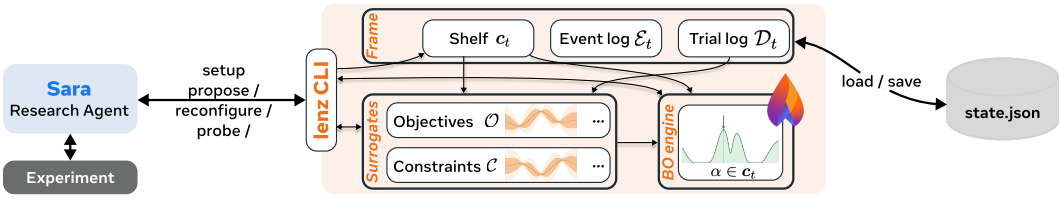


Figure 26: *Schematic overview of lenz.* A *Frame* combines the immutable search space with a mutable *Shelf*, per-outcome model evidence, a trial log with explicit lifecycle states, and a mutation event log. The JSON state remains authoritative across command-line calls.

The search space supports continuous, integer, stepped, log-scaled, ordered, and unordered categorical dimensions. lenz fits one GP for each modeled outcome. The surrogates for objectives and constraints affect the acquisition functions and therefore also **propose** actions, whereas tracked metrics are prediction-only and can, for example, represent runtime as showcased in Section F. Initial datasets may supply outcome-specific model evidence. Inputs are normalized and outputs standardized following standard BO practice (Balandat et al., 2020). Furthermore, we added functionality for explicit kernel expressions allowing for additive and product compositions of Matérn, RBF, periodic, rational-quadratic, linear, and categorical components.

The acquisition functions are state dependent and here we follow the best practice implementations in Ax. Before a constrained study has a feasible observation, lenz draws Sobol candidates and then defaults to maximizing the predicted joint feasibility. After a feasible observation, it uses constrained log expected improvement or its noisy variant (Gardner et al., 2014; Gelbart et al., 2014; Ament et al., 2023). For multiple objectives, it provides logarithmic expected-hypervolume-improvement acquisitions for noisy and deterministic settings (Daulton et al., 2020; 2021). Batch suggestions and submitted evaluations are handled through Monte Carlo q-acquisition functions using fantasization. Also pending points for both synchronous and asynchronous batches are supported through fantasization. Configuration-dependent crashes may also be recorded resulting in pessimistic virtual point following Stenger & Abel (2022) which are re-assigned at each iteration.

L.2 COMMAND-LINE INTERFACE

The interface is self-documenting so that the agent does not have to rely on a static copy of the command reference loaded into the context on start up. Its main entry point, `lenz help agent`, is generated from the same command registry as the parser and summarizes the available action classes, trial lifecycle, deliberation invariants, and further discovery commands. `lenz --help` lists the current commands, while `lenz help <command>` explains one operation and the `--json` variant returns its machine-readable schema. Concept help similarly documents the reusable acquisition, kernel, outcome, protocol, and search-space formats. We instruct Sara to consult these interfaces at runtime (*cf.* Section K), allowing the available capabilities to evolve without depending on memorized syntax. Every study command names its state file explicitly and emits one JSON envelope: `{"ok": true, "command": ..., "result": ...}` on success, or a structured error with `error_code` and optional details on failure. The command surface is summarized in Table 11; the separation between computational actions and evaluation-lifecycle records mirrors

the distinction in Section 4. Figure 27 reproduces the complete deliberation and operating-contract sections of the runtime guide. For compactness, we abbreviate the action inventory.

DISCOVERY INPUT

```
lenz help agent
```

RESPONSE · SELECTED SECTIONS

```
Lenz agent guide

Lenz is a tool for agentic Bayesian optimization.

Deliberation:
  - Deliberation is the observable sequence of Lenz actions and backend responses used to
inform the next evaluation decision; it is evaluation-budget-free and does not refer to
hidden chain-of-thought.
  - After setup, an agent may probe, propose, and reconfigure zero or more times before
committing a configuration and evaluating it.
  - suggest (and its variants), score, and predict may be called repeatedly without
modifying the trial log or consuming black-box evaluation budget; the agent may adopt,
refine, or override a proposal before submit.
  - submit marks an exact configuration pending; observe records real metrics; fail
resolves an unsuccessful evaluation; fail --kind crash identifies a
configuration-dependent failure; withdraw removes submitted work that will not be
evaluated.
  - Reconfiguration persists optimization-policy changes without discarding prior
evaluations; probes read model and campaign state without changing it.
  - You decide whether and how to use Lenz's proposals and probes; Lenz supplies Bayesian
optimization state and computations to inform an evaluation decision.

Operating contract:
  - Only real evaluation metrics are observations.
  - Use `lenz --help` to discover commands and help topics.
  - Before using a command for the first time, call `lenz help <command> --json`.
  - Use exact option names; Lenz rejects abbreviated options.
  - When a command fails, inspect `error`, `error_code`, and any `details`, then consult
its JSON help before retrying.
  - Use `lenz status` to verify persisted study state instead of inferring it from
earlier commands.
  - Submit the exact configuration before evaluating it.
  - If known, always use the full available budget per default.
  - Resolve submit with observe after success, fail after an evaluation failure, or
withdraw when it will not be evaluated.
  - Use fail --kind crash when the configuration caused missing objective metrics.
  - Setup establishes the study; propose, reconfigure, and probe are available throughout
the study and are not a prescribed sequence.
  - Study commands return JSON envelopes. On ok: false, inspect error and error_code
before continuing.
  - Lenz help and implementation describe CLI and Bayesian optimization semantics;
evaluator and objective semantics are outside the Lenz interface.

Actions:
  Setup: create
  Computational:
    Propose: suggest
    Probe: predict, score, diagnostics, status, ...
    Reconfigure: set-bounds, set-acqf, set-kernel, ...
  Evaluation lifecycle: submit, observe, fail, withdraw

Discovery:
  lenz --help
  lenz help <command>
  lenz help <command> --json
  lenz help <topic>
  lenz help agent --json
```

Figure 27: *Self-documenting interface.* `lenz help agent` provides the agent-oriented entry point to the current capabilities and operating contract. The deliberation and operating-contract sections are shown in full; the action inventory is abbreviated. The same information is available as structured metadata with `--json`.

The following examples use one mixed scaling-law study and show the complete envelope structure, while abbreviating large result objects to the fields discussed in the text. Values are illustrative and remain in their original units.

Role	Commands
Setup	`create`
Propose	`suggest`
Evaluation lifecycle	`submit`, `observe`, `fail`, `withdraw`
Reconfigure	`set-acqf`, `set-bounds`, `set-objectives`, `set-reference-point`, `set-constraints`, `set-kernel`, `set-tracked-metrics`
Probe	`predict`, `score`, `incumbent`, `pareto`, `trials`, `diagnostics`, `status`

Table 11: The current lenz command surface. Propose, probe, and reconfigure calls do not consume black-box evaluation budget; lifecycle commands record what happens to an external evaluation.

SETUP **INPUT**

```
lenz create --state study.json \
  --space '{
    "n_layers":{"kind":"range","lower":4,"upper":24,"type":"int","step":4},
    "d_model":{"kind":"range","lower":256,"upper":1024,"type":"int","step":128},
    "optimizer":{"kind":"choice","values":["adam","adamw"]},
    "lr":{"kind":"range","lower":0.00001,"upper":0.01,"log_scale":true}
  }' \
  --objectives '{"nll":"minimize"}' \
  --constraints '[{"metric":"flops","upper":500}]' \
  --tracked-metrics '["duration"]'
```

RESPONSE · SELECTED FIELDS


```
{
  "ok": true,
  "command": "create",
  "result": {
    "objective": "nll",
    "minimize": true,
    "is_moo": false,
    "acqf": "lognei",
    "constraints": [{"metric": "flops", "lower": null, "upper": 500}],
    "tracked_metrics": ["duration"],
    "trials": 0,
    "observed": 0,
    "sobol_seed": 42
  }
}
```


Figure 28: *Setup.* `create` validates and canonicalizes the complete study definition, persists it at `--state`, and returns a summary that the agent can verify before any evaluation.

PROPOSE **INPUT**

```
lenz suggest --state study.json
```

RESPONSE


```
{
  "ok": true,
  "command": "suggest",
  "result": [{
    "config": {"n_layers": 12, "d_model": 768,
               "optimizer": "adamw", "lr": 0.0003},
    "acquisition_values": {"lognei": -1.92},
    "trial_id": null, "acqf": "lognei"
  }]
}
```


PROPOSE **INPUT**

```
lenz suggest --state study.json \
  --around `{
    "n_layers":{"radius":0.25},
    "d_model":{"radius":0.15},
    "optimizer":["adamw"],
    "lr":{"radius":0.10}
  }` \
  --around-shape ellipsoid
```

RESPONSE


```
{
  "ok": true,
  "command": "suggest",
  "result": [{
    "config": {"n_layers": 12, "d_model": 768,
               "optimizer": "adamw", "lr": 0.0003},
    "acquisition_values": {"lognei": -1.92},
    "trial_id": null, "acqf": "lognei"
  }]
}
```


PROPOSE **INPUT**

```
lenz suggest --state study.json --q 2 \
  --bounds `{
    "n_layers":[8,20],
    "d_model":[512,1024],
    "lr":[0.0001,0.001]
  }`
```

RESPONSE


```
{
  "ok": true,
  "command": "suggest",
  "result": [
    {"config": {"n_layers": 8, "d_model": 640,
                "optimizer": "adam", "lr": 0.0005},
     "acquisition_values": {"lognei": -2.08},
     "trial_id": null, "acqf": "lognei"},
    {"config": {"n_layers": 20, "d_model": 896,
                "optimizer": "adamw", "lr": 0.0002},
     "acquisition_values": {"lognei": -2.31},
     "trial_id": null, "acqf": "lognei"}
  ]
}
```


Figure 29: *Candidate generation.* A plain `suggest` searches the active domain, `suggest --around` constructs an ephemeral neighborhood around the incumbent with per-dimension radii or allowed values, and `suggest --bounds` restricts acquisition optimization to a supplied region for one call. The optional `--q` argument requests a batch. All variants are pure reads: the `null` trial identifier confirms that no evaluation has been recorded, and neither regional restriction changes the persisted bounds.

EVALUATION LIFECYCLE — INPUT

```
lenz submit --state study.json \
  --config '{"n_layers":12,"d_model":768,"optimizer":"adamw","lr":0.0003}'

# Resolve the submission in exactly one of three ways
lenz observe --state study.json \
  --config '{"n_layers":12,"d_model":768,"optimizer":"adamw","lr":0.0003}' \
  --metrics '{"nll":2.41,"flops":462,"duration":38.2}'
lenz fail --state study.json \
  --config '{"n_layers":12,"d_model":768,"optimizer":"adamw","lr":0.0003}' \
  --kind crash
lenz withdraw --state study.json --trial-id trial_7
```

RESPONSES · SELECTED FIELDS

```
{"ok": true, "command": "submit",
 "result": {"trial_id": "trial_7", "status": "submitted"}}
{"ok": true, "command": "observe",
 "result": {"trial_id": "trial_7", "status": "observed",
            "metrics": {"nll": 2.41, "flops": 462, "duration": 38.2}}}
{"ok": true, "command": "fail",
 "result": {"trial_id": "trial_7", "status": "failed",
            "failure_kind": "crash"}}
{"ok": true, "command": "withdraw",
 "result": {"trial_id": "trial_7", "status": "withdrawn"}}
```

Figure 30: *Evaluation lifecycle.* `submit` marks an exact configuration as pending. The external evaluation is then resolved by `observe` with measured outcomes, by `fail` after an unsuccessful evaluation, or by `withdraw` if the submitted work will not be evaluated.

RECONFIGURE — INPUT

```
lenz set-bounds --state study.json \
  --bounds '{"n_layers":[8,20],"d_model":[512,1024],"lr":[0.0001,0.001]}'
lenz set-acqf --state study.json --acqf logei
lenz set-kernel --state study.json --outcome duration --dry-run \
  --kernel '{"kernel":"matern","nu":1.5,"dimensions":["n_layers","d_model"]}'
lenz set-objectives --state study.json \
  --objectives '{"nll":"minimize","flops":"minimize"}'
```

RESPONSES · SELECTED FIELDS

```
{"ok": true, "command": "set-bounds",
 "result": {"bounds": {"n_layers": [8,20],
   "d_model": [512,1024], "lr": [0.0001,0.001]},
   "trials": 14, "observed": 12}}

{"ok": true, "command": "set-acqf",
 "result": {"acqf": "logei", "trials": 14, "observed": 12}}

{"ok": true, "command": "set-kernel",
 "result": {"changed": true, "outcome": "duration",
   "source": "override",
   "kernel": {"kernel": "matern", "nu": 1.5,
     "dimensions": ["n_layers", "d_model"]}, "saved": false}}

{"ok": true, "command": "set-objectives",
 "result": {"objectives": [
   {"metric": "nll", "minimize": true},
   {"metric": "flops", "minimize": true}],
   "is_moo": true, "acqf": "logehvi"}}
```

Figure 31: *Persistent reconfiguration.* `set-bounds` and `set-acqf` persist the active search region and acquisition policy. Kernel policies can be validated without writing via `--dry-run`; other calls persist changes to the study. Switching from one to multiple objectives automatically reconciles compatible improvement acquisitions, here from `logei` to `logehvi`, while retaining all observations.

PROBE **INPUT**

```
lenz status --state study.json --outcomes --trials --outcome duration
lenz diagnostics --state study.json --outcome duration
lenz predict --state study.json \
  --config `{"n_layers":16,"d_model":896,"optimizer":"adamw","lr":0.0002}` \
  --metrics `["duration","flops"]`
lenz score --state study.json \
  --configs `[
    {"n_layers":12,"d_model":768,"optimizer":"adamw","lr":0.0003},
    {"n_layers":16,"d_model":896,"optimizer":"adamw","lr":0.0002}
  ]`
```

RESPONSES · SELECTED FIELDS


```
{"ok": true, "command": "status",
 "result": {
   "outcomes": {"duration": {"roles": ["tracked"],
     "observations": 12, "fit_ready": true,
     "kernel_source": "override"}},
   "trials": {"total": 14, "observed": 12, "submitted": 2}}}

{"ok": true, "command": "diagnostics",
 "result": {"outcome": "duration", "n_observed": 12,
   "cv_r2": 0.86, "noise": 0.03, "outputscale": 1.12,
   "lengthscales": {"n_layers": 0.41, "d_model": 0.28},
   "kernel_components": [{"kernel": "MaternKernel",
     "dimensions": ["n_layers", "d_model"],
     "lengthscales": [0.41, 0.28]}],
   "sensitivity": {"n_layers": 0.31, "d_model": 0.61}}}

{"ok": true, "command": "predict",
 "result": [{"mean": {"duration": 44.8, "flops": 487.0},
   "variance": {"duration": 5.6, "flops": 324.0},
   "prob_feasible": 0.76}]}

{"ok": true, "command": "score",
 "result": [{"lognei": -1.92}, {"lognei": -2.37}]}
```


Figure 32: *Read-only probes.* `status` reports declared outcome roles, model readiness, kernel provenance, and trial counts without fitting a model. `predict` returns posterior moments and joint feasibility, `score` compares agent-supplied candidates under the current acquisition function, and `diagnostics` exposes predictive fit, GP hyperparameters, kernel structure, and parameter sensitivities. Related probes return incumbents, Pareto fronts, and the complete trial log.