ASAP: AGENT SELECTION ACROSS PROPOSERS FOR HYPERPARAMETER OPTIMIZATION UNDER WALL-CLOCK BUDGETS

Anonymous authors
Paper under double-blind review

ABSTRACT

Hyperparameter Optimization (HPO) is essential for maximizing machine learning (ML) model performance. Because every HPO tool relies on a surrogate prior that imparts its own inductive bias, individual tools struggle once problems grow diverse and drift from these priors. Recent work therefore explores LLMs for HPO and reports improved per-iteration performance, yet shares two limitations with a common origin: the LLM is used as a *single-tool replacement evaluated by iteration count*. (i) The LLM replaces rather than complements prior tools, yet is itself constrained by pretraining to one family of inductive-biased proposals, so a single source still cannot cover diverse problems. (ii) Per-iteration evaluation ignores the LLM and tool latency paid serially every round, so iteration gains do not translate into real **end-to-end wall-clock** gains. We present **ASAP**, which addresses both. Its core is *integration*: the LLM acts as a Judge over a diverse pool of inductive-biased optimizers and selects among their proposals each round. Even in a serial loop that pays the full LLM and tool latency every round, integration reaches markedly lower regret per unit wall-clock than statistical, LLM-based, and frontier-LLM-agent baselines across 28 modern HPO tasks. On top of it, ASAP co-designs the loop for wall-clock: a *prefix-stable prompt* keeps the Judge's prompts cache-friendly, and *speculation parallelism* hides the LLM and tool latency under model evaluation when the two are of similar size, committing a speculative round only when its predicted outcome passes a relative-error accept test and otherwise falling back to the serial loop; on such tasks it cuts wall-clock by 12–24% at matching regret.

1 INTRODUCTION

Hyperparameter Optimization (HPO) is essential for maximizing machine learning (ML) model performance. It is sequential decision-making: given past (hyperparameter, performance) observations, the algorithm selects the next configuration to evaluate (Bergstra & Bengio, 2012). Existing methods combine a surrogate of the black-box objective, a candidate sampler, and an acquisition function, and fall into (1) *conventional statistical* approaches such as Scikit-Optimize (SKopt) (scikit optimize) and Optuna (Akiba et al., 2019), built on Gaussian processes (Williams & Rasmussen, 1995) or Tree-structured Parzen Estimators (TPE) (Watanabe, 2026) with random or Bayesian sampling (Ding et al., 2014) and expected improvement or upper confidence bound; and (2) *LLM-based* approaches (Liu et al., 2024; 2025), in which an LLM plays one or more of these roles.

Real-world HPO tasks vary widely in data and architecture, and no single optimizer dominates: each tool (GP-based Bayesian optimization, TPE, SMAC, or an LLM proposer) is built around a prior that fits some landscapes and not others (Wolpert & Macready, 2002; Li et al., 2024). **The practical problem is therefore tool selection.** On our 28-task suite the best tool changes from task to task, and picking it per task would more than halve regret relative to an uninformed choice (0.039 vs. 0.090, Sec. 3.1), yet at round 0 there is no evaluation to base that choice on. LLM-based optimizers such as LLAMBO (Liu et al., 2024) do not escape this: they *replace* a conventional tool with the LLM, itself one more fixed-prior proposer, leaving the selection problem where it was. A second, largely ignored drawback is that HPO is evaluated almost exclusively by iteration count, while in real runs LLM inference and tool execution are paid serially on top of the model evaluation T_{ML}

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.

every round; summed over iterations, this overhead can rival the cost the optimizer tries to amortize, so iteration-count gains need not translate into **end-to-end wall-clock** gains (Fig. 1).

Our design follows two principles. (1) **Integration over replacement**: we turn task-level tool selection, impossible before the first evaluation, into a per-round selection among the proposals of a diverse pool (statistical tools and LLM-as-tool alike), made by one LLM Judge that conditions on the shared history and extends to any new tool. Selecting per round from complementary priors reduces per-tool fragility and yields robust optimization across the diverse tasks we test. (2) **Wall-clock as the design objective**: once end-to-end wall-clock rather than iteration count is the target, prompt structure, tool scheduling, and execution policy become coupled and must be designed together with the agent.

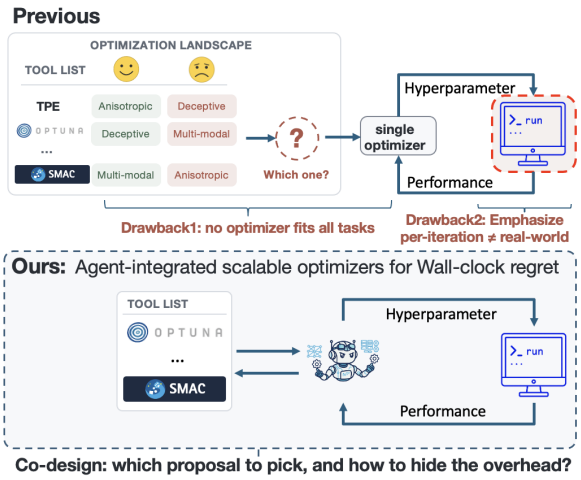


Figure 1: Motivation. Prior LLM-HPO replaces a *single* optimizer, though none fits all tasks, and is judged per iteration rather than by wall-clock; ASAP integrates a pool of optimizers under one LLM agent and co-designs the system for wall-clock regret.

Guided by these principles, we present **ASAP** (Agent Selection Across Proposers), whose core is agent-integrated selection, complemented by a wall-clock co-design. (1) **Agent-integrated optimizers**: each round, a diverse pool of tools proposes candidates in parallel and an LLM-as-Judge selects among them, so complementary inductive biases cover problems no single proposer covers alone. (2) **Reducing the wall-clock cost of selection**, built on top of integration: a *prefix-stable prompt* keeps an immutable prefix so the Judge's prompts are cache-friendly, and *speculation parallelism* predicts the current round's outcome with the LLM, runs the next round's tools and Judge during the current evaluation, and commits under a relative-error accept test that never re-runs the model evaluation. Integration alone already gives the lowest regret per unit wall-clock across our suite; speculation adds a further saving where evaluation and overhead are of similar size. The contributions of this work are summarized as follows:

- New problem framing: HPO as per-round selection across the proposals of diverse optimizers, evaluated under wall-clock budgets, exposing two ignored drawbacks of LLM-based HPO: tools that substitute rather than compose, and iteration-centric evaluation that ignores the overhead dominating wall-clock.
- ASAP framework: (1) agent-integrated optimizers, whose complementary biases jointly cover diverse problems; and (2) wall-clock co-design on top of it, a cache-friendly prompt and speculation parallelism with a latency-safe fallback, which cuts wall-clock by 12–24% at matching regret where evaluation and overhead are of similar size.
- Empirical validation: experiments spanning tabular, vision, language, and protein models show ASAP's effectiveness and efficiency, and a landscape analysis shows that the best fixed tool changes with the landscape while ASAP stays in the top group on every axis.

2 Related Work

LLMs for AutoML and HPO LLMs are increasingly applied to autonomous machine learning (AutoML) (Tornede et al., 2024), but most work targets broader tasks such as dataset processing and workflow orchestration (Hong et al., 2024; Guo et al., 2024; 2025; Chi et al., 2024), letting the LLM generate hyperparameters directly without optimization and sacrificing peak model performance. Among LLM-HPO methods, LLAMBO (Liu et al., 2024) uses an LLM as both surrogate and acquisition function; others (Zhang et al., 2024; Liu et al., 2025; Guo et al., 2026) use LLM agents that analyze results and directly suggest the next hyperparameters. All of these design the

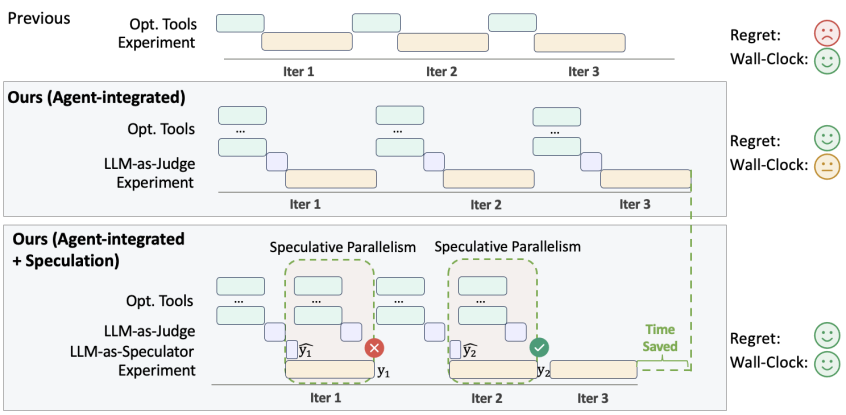


Figure 2: ASAP execution loop. After a serial warm-up, the Judge's own prediction $\hat{y}_n$ for the selected configuration stands in for round n's outcome, and the next round's tools and Judge run on it while the evaluation (T_{ML}) is in flight. On accept (relative error below τ), the LLM+tool cost is hidden under T_{ML}; on reject, the tool+Judge stage re-runs with the true y_n, matching the serial wall-clock.

LLM to *replace a single* HPO tool and ignore the LLM inference cost paid serially every round. Choosing among several optimizers is also a classical BO question: portfolio methods such as GP-Hedge (Brochu et al., 2011) and entropy-search portfolios (Shahriari et al., 2015) use a bandit over past rewards to pick one *acquisition function* per round over a single shared surrogate, not among heterogeneous optimizers that each bring their own surrogate and sampler, so they do not apply directly to such a pool; the closest surrogate-based counterpart in our setting is a GP-EI selector over the pool (Table 1a). ASAP's LLM Judge instead ranks the *concrete proposals* of heterogeneous tools (statistical and LLM alike) against the shared history and the textual task context, so a new tool is usable from its first proposal without a learning phase. *ASAP instead (i) integrates a* pool *of HPO tools under one LLM-as-Judge, mitigating any single tool's inductive bias, and (ii) is evaluated under* end-to-end wall-clock*, where it hides the agent cost under model evaluation whenever the two are of similar size.*

Speculative Acceleration for LLM Agents Speculation originated in decoding acceleration, where a draft model proposes tokens verified in parallel under a distribution-preserving rule (Leviathan et al., 2023). Recent work lifts it to the agent loop, speculating next actions or tool calls executed ahead of verification (Ye et al., 2026; Guan et al., 2025), or downstream workflow nodes verified asynchronously with rollback (Ro et al., 2025); all of these speculate *discrete* targets under exact, distributional, or binary acceptance tests. Running the next round on a predicted outcome also has a BO counterpart: Kriging believer and constant liar (Ginsbourger et al., 2010) and asynchronous BO with fantasized pending evaluations (Snoek et al., 2012; Kandasamy et al., 2018) impute unfinished evaluations so that *more evaluations* can run in parallel, and cost-aware acquisition functions (Lee et al., 2020) trade expected improvement against evaluation cost. *ASAP differs in object and test: evaluations stay serial and the target is the LLM agent's own overhead, speculating a* continuous *model-performance value driving tool inputs and committing under a relative-error gate whose fallback re-runs only the cheap tool+Judge stage: the worst-case round equals the non-speculative baseline, while accepted rounds hide the LLM+tool overhead under the evaluation.*

3 ASAP Module 1: Agent-Integrated Optimizers

3.1 Why Integrate Rather Than Replace: HPO as Tool Selection

Setup and notation. Let $f : \mathcal{X} \to \mathbb{R}$ be the black-box objective of an ML task and $D_n = \{(x_t, y_t)\}_{t=0}^{n-1}$ the history at the start of round n, with $y_t = f(x_t)$ (we write $D_n(f)$ and $C_n(f)$ when the task matters). An HPO tool $i \in [K]$ is a proposer: given the history it returns a small batch of candidates $G_i(D_n) \subset \mathcal{X}$, whose top element $g_i(D_n)$ it would evaluate next. Each tool (GP-based BO (Snoek et al., 2012), TPE (Bergstra et al., 2011), SMAC (Lindauer et al., 2022), or an LLM

proposer such as LLAMBO (Liu et al., 2024)) is built around its own family of landscapes, so none dominates across families (Wolpert & Macready, 2002; Eggensperger et al., 2022); the question is *which tool to use on which task*. Running tool i alone for n rounds on f incurs the simple regret

$$R_i(f,n) \;=\; f^* - \max_{t\le n} f(x_t), \qquad x_t = g_i(D_t), \tag{1}$$

where $f^* = \max_{x\in\mathcal{X}} f(x)$.

Static tool selection is the real problem, and it cannot be solved before the first evaluation. An HPO service faces a *suite* $\mathcal{F} = \{f^{(1)},\dots,f^{(M)}\}$. Selecting one tool per task, $i^*(f) = \arg\min_i R_i(f,n)$, attains the *oracle* regret $R_{\mathrm{orc}}(\mathcal{F},n) = \frac{1}{M}\sum_f \min_i R_i(f,n)$; one tool for all tasks gives $\bar{R}_i(\mathcal{F},n) = \frac{1}{M}\sum_f R_i(f,n)$; and picking a tool uniformly at random gives the *grand-mean* regret $\bar{R}(\mathcal{F},n) = \frac{1}{K}\sum_i \bar{R}_i(\mathcal{F},n)$:

$$R_{\mathrm{orc}}(\mathcal{F},n) \;\le\; \min_i \bar{R}_i(\mathcal{F},n) \;\le\; \bar{R}(\mathcal{F},n). \tag{2}$$

On our 28-task suite at n=35 the three terms are 0.039, 0.058 (TuRBO), and 0.090, and the per-task best tool is spread over four optimizers. But $i^*(f)$ depends on the landscape of f (Fig. 6), which a search space and a task description do not reveal. Without evaluating f, a rule can at best commit to one tool for every task ($\min_i \bar{R}_i$, if it knows which tool is best on average), and otherwise sits at $\bar{R}$; meta-learned priors (Wang et al., 2024) do better but need a corpus of related tasks. Replacing one tool by an LLM adds a row to the ladder; it does not choose among rows.

Reformulation 1: from static to dynamic tool selection. Each evaluation reveals more of the landscape, so the tool can be chosen *per round*. A selection rule s maps D_n to a tool index, and the round evaluates that tool's proposal:

$$x_n = g_{s(D_n)}(D_n), \qquad R_s(f,n) = f^* - \max_{t\le n} f(x_t). \tag{3}$$

Static choices are special cases ($s \equiv i$ gives R_i; uniform s gives $\bar{R}$ in expectation). Dynamic selection decides with D_n in hand, can switch tools along a run (a broad proposer early, a local one late), and pays for a wrong choice with one round rather than the whole budget; its ceiling is the per-round oracle, not the per-task oracle of Eq. equation 2.

Reformulation 2: from selecting tools to selecting their results. Dynamic selection still needs to know which tool suits the landscape. Since running a tool costs no evaluation of f, we instead run *all* K tools on the same D_n, collect

$$C_n = \bigcup_{i\in[K]} G_i(D_n), \tag{4}$$

and let a Judge h pick $x_n = h(C_n, D_n)$, comparing *concrete configurations* rather than abstract tools. With $r_c(f,n) = f^* - f(c)$ the per-round regret of evaluating $c \in C_n(f)$, a Judge that picks uniformly has expected regret equal to the task's *pool mean* $\bar{r}(f,n)$, and $\bar{r}(\mathcal{F},n)$ on average over the suite (Eq. equation 10, Appendix A). This elementary identity fixes the reference point: $\bar{r}(\mathcal{F},n)$ is the shared-history, per-round counterpart of $\bar{R}$ (not the same quantity, since $\bar{R}$ lets each tool run on its own history). Per round, integration alone changes nothing, and any gain over the pool average must come from the selector. (End-to-end, the pool's diversity does carry a portfolio gain: random selection over C_n beats every single tool, and the Judge lowers regret further, from 0.046 to 0.026; Table 1.) With $\bar{r}_{\mathrm{int}}(\mathcal{F},n)$ the per-round mean regret under a real Judge (Eq. equation 12), integration wins under a deliberately mild criterion:

$$\bar{r}_{\mathrm{int}}(\mathcal{F},n) \;<\; \bar{r}(\mathcal{F},n) \;\iff\; h \text{ outranks } \mathrm{U}(C_n(f)) \text{ on average over } \mathcal{F}, \tag{5}$$

i.e., the Judge need only beat uniform random selection over $C_n(f)$. This is a per-round statement; we do not prove that per-round wins compose into a trajectory-level win, but observe on replayed candidate pools that the Judge beats uniform selection at *every* round (Fig. 5) and that the committed trajectory reaches the lowest end-to-end regret of any method, below the per-task oracle single tool (0.026 vs. 0.039) and the same pool under random or GP-EI selection (0.046).

Why an LLM as the Judge. (i) *The bar is low*: Eq. equation 5 asks only for better-than-uniform ranking of a small, historically grounded set, a discriminative rather than generative task. (ii) *Selection is contextual*: unlike a bandit over tool identities, which must learn K arms from tens of rounds, the Judge conditions on each candidate, on D_n, and on textual side information (task metadata, hyperparameter semantics), so it can rank a proposal from a tool it has never picked. (iii) *It is informed from the start and improves in context*: pretraining encodes hyperparameter regularities (learning rates near 10^{-3} on a log scale, batch sizes as powers of two), giving above-random selection from the first round, where classical surrogates are data-starved, and its few-shot regression sharpens as D_n grows (Fig. 5). The price is that every round now runs K tools plus a Judge before T_{ML}; reducing that overhead is the role of Module 2 (Sec. 4.1).

3.2 How Integrating: The Integrated Loop

Tool pool and per-round procedure. ASAP builds C_n from $K = 8$ diverse optimizers: GP (Snoek et al., 2012), TPE (Bergstra et al., 2011), SKOPT (scikit optimize), TuRBO (Eriksson et al., 2020), Optuna (Akiba et al., 2019), SMAC (Lindauer et al., 2022), DNGO (Snoek et al., 2015), and the LLM-based LLAMBO (Liu et al., 2024). The pool is extensible: more diversity widens the gap between uniform selection and the per-task best, the headroom a better-than-random Judge converts into gains. In round n, **(1)** all tools propose **in parallel** from the shared D_n, so the propose latency is that of the slowest tool; **(2)** the Judge selects $x_n = h(C_n, D_n)$ from the problem context and D_n; **(3)** x_n is evaluated, yielding $y_n = f(x_n)$; **(4)** $D_{n+1} = D_n \cup \{(x_n, y_n)\}$.

Judge implementation: In-Context Learning Judge. We keep the Judge simple. Each candidate $c \in C_n$ is scored with a prompt that shares a common prefix (problem context and D_n) and differs only in the queried candidate; we draw S=20 predictions (two prompt templates $\times$ 10 samples, temperature 0.7), discard unparsable outputs, and use the mean μ_c and standard deviation σ_c (floored at 10^{-5}) as predictive mean and uncertainty. With objectives oriented so that higher is better and y^+ the best observed value, the Judge selects the candidate maximizing the Gaussian Expected Improvement $\text{EI}(c) = (\mu_c - y^+)\Phi(z_c) + \sigma_c\phi(z_c)$, $z_c = (\mu_c - y^+)/\sigma_c$. The sample spread is a practical rather than calibrated uncertainty, so decoding temperature acts as an explore/exploit knob, and framing selection as in-context regression plus EI keeps the Judge characterizable with a clean acquisition geometry. The prompt template and problem context are in Appendix B.3 (Table 3).

4 ASAP Module 2: Reducing the Wall-Clock Cost of Selection

4.1 Serial Workflow and Its Wall-Clock Cost

Module 1's selection quality comes at a price classical HPO never pays: the tool calls and LLM inference add to the wall-clock of every round. Module 2 lowers this latency. Parallel proposing and the cache-friendly prompt leave every decision unchanged; speculation commits a decision made on a predicted outcome only when the prediction passes the accept test and otherwise re-runs the serial decision on the true history, which in our experiments has little effect on regret (Sec. 5.3), an empirical observation rather than a guarantee. Fig. 2 contrasts the serial loop with the speculative one. Over N rounds the serial wall-clock is

$$T_{\text{wall}}^{\text{base}} = \sum_{n=1}^{N} \left(T_{\text{tool}}^{(n)} + T_{\text{judge}}^{(n)} + T_{\text{ML}}^{(n)}\right), \tag{6}$$

where a classical optimizer's proposal cost is negligible beside $T_{\text{ML}}^{(n)}$. Three mechanisms attack the extra $T_{\text{tool}}+T_{\text{judge}}$: parallel proposing collapses T_{tool} to the slowest tool (Sec. 3.2), a *KV-cache-aware prompt* shrinks T_{judge} (Sec. 4.2), and *cross-iteration speculation* hides the remaining cost under T_{ML} (Sec. 4.3), driving T_{wall} toward the lower bound $\sum_n T_{\text{ML}}^{(n)}$.

4.2 Reducing T_{judge}: KV-Cache-Aware Prompts

Across rounds the Judge prompt is almost entirely repeated: each prompt is a fixed task description, the (config, performance) demonstrations in D_n, and a short query for one candidate. Because the history is *append-only*, round n+1 differs from round n only by the new (x_n, y_n) and the query,

so the leading block is token-for-token identical, exactly the redundancy a KV cache exploits. We therefore split the prompt into an **immutable prefix** (task description and committed demonstrations), shared by all candidates scored in a round and extended append-only across rounds, and a **per-round suffix** holding only the candidate to be scored (example in Table 3). Per-round prefill then drops from the full prompt to the suffix, $T_{\text{prefill}}^{\text{cached}} \approx L_{\text{suffix}}\, t_{\text{token}} \ll L_{\text{full}}\, t_{\text{token}}$, and since the Judge decodes only a few tokens, prefill dominates T_{judge}. The layout is model-agnostic, and the saving is realized whenever the serving stack reuses prefix KV caches.

4.3 Hiding Overhead via Cross-Iteration Speculation When Evaluation and Overhead Are Comparable

Speculative execution. Each round's stages run serially because each consumes the previous one's output, so during T_{ML} the tools and Judge sit idle, waiting for a single scalar. That scalar need not be waited for: when the Judge selects x_n it has already predicted its performance $\hat{\mu}(x_n)$ for EI scoring. We take $\hat{y}_n \triangleq \hat{\mu}(x_n)$ as the speculated outcome, with no separate predictor, and run round $n{+}1$ on $\hat{D}_{n+1} = D_n \cup \{(x_n, \hat{y}_n)\}$ while round n's evaluation is in flight, hiding the cost of $g_i(\hat{D}_{n+1})$ and $h(\hat{C}_{n+1}, \hat{D}_{n+1})$ under $T_{\text{ML}}^{(n)}$. Speculation is one step deep, so no prediction is conditioned on another; it is *cross-iteration* because the speculative work belongs to round $n{+}1$ while the predicted y_n comes from round n.

Accept test. When y_n returns, we accept the speculative round only if $a_n = \mathbb{1}\big[\,|\hat{y}_n - y_n| / \max(|y_n|, \epsilon) < \tau\,\big]$, with $\epsilon{=}10^{-6}$ keeping the test defined when $y_n \approx 0$. On accept, the speculatively chosen configuration is evaluated immediately and the tool+Judge cost is fully hidden; on reject, only the tool+Judge stage re-runs on the true y_n, exactly the serial round, and the ML evaluation is never re-run. Speculation can therefore only save wall-clock. An accepted round costs

$$T_{\text{round}}^{(n)} = \max\big(T_{\text{ML}}^{(n)},\, T_{\text{tool}}^{(n+1)} + T_{\text{judge}}^{(n+1)}\big), \tag{7}$$

and a rejected round the serial sum, so each accepted round saves $\min\big(T_{\text{ML}}^{(n)},\, T_{\text{tool}}^{(n+1)} + T_{\text{judge}}^{(n+1)}\big)$, the most when the two stages are comparable. The serial warm-up and the final round are paid in full, so when most rounds are accepted and the overhead fits under T_{ML}, T_{wall} approaches $\sum_n T_{\text{ML}}^{(n)}$ plus these two terms. The guarantee is on latency only: relative to the serial loop, speculation never adds wall-clock, but it does not guarantee unchanged decisions.

Choosing τ. The gate does not require an exact prediction. HPO already operates on noisy evaluations, and an accepted misprediction affects at most one decision, because the true y_n enters the history as soon as it returns, whereas a rejection costs a full serial round. We therefore treat the gate as a safeguard against failed predictions rather than a precision filter, and fix $\tau = 0.30$ for all tasks without per-task tuning. Because the Judge predicts a single value by in-context regression, its errors are small, so the gate rarely binds and the outcome is insensitive to τ (Sec. 5.3). Domains with less reliable predictions call for a tighter τ, which is always safe: as $\tau \to 0$ ASAP reduces to its serial loop and only the saving shrinks.

5 Experiments

We organize the evaluation around three research questions: **RQ1 (Effectiveness and Generalization)**: does ASAP reach lower regret per unit wall-clock? **RQ2 (Decomposing the gains)**: how do integration and speculation each contribute? **RQ3 (Interpretability)**: why does ASAP work?

5.1 Experiment Settings

Benchmarks. We evaluate on 28 (task, model) combinations with non-trivial evaluation cost: (1) **HPOBench** (Eggensperger et al., 2022), real training of XGBoost and RandomForest on six OpenML tabular tasks (9 combinations, ∼9–240 s per trial); (2) **PD1** (Wang et al., 2024), Google HyperBO's replay of deep-learning HPO for CNNs, Wide-ResNets, ResNet-50, and Transformers on vision, language, and protein data (19 combinations, ∼2 min–48 h per trial). All search spaces are four-dimensional. Details are in Appendix B.1 (Table 2).

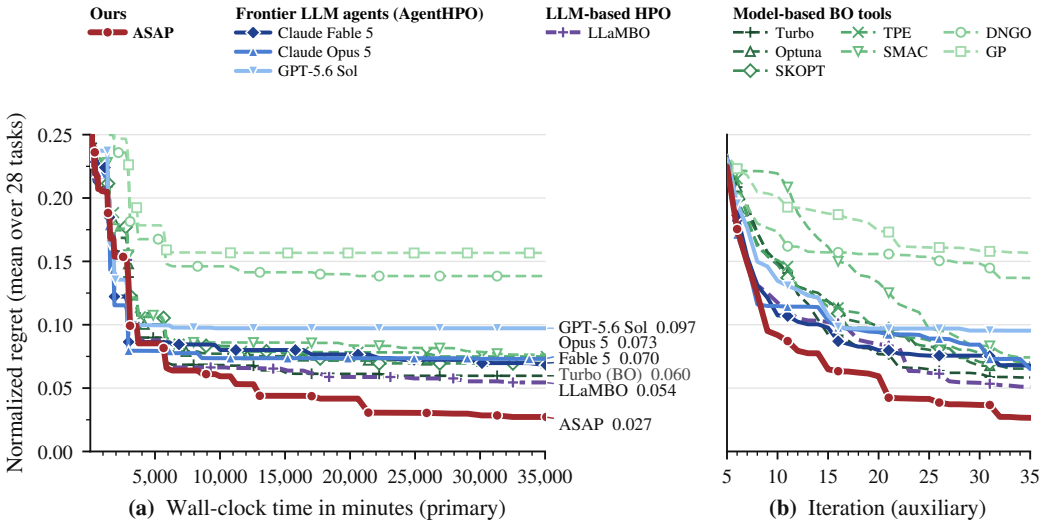


Figure 3: Normalized regret vs. (a) end-to-end wall-clock (primary) and (b) iteration (auxiliary), averaged over the 28 combinations; labels at the right edge of (a) give the regret at 35,000 min. **ASAP** (red) runs in its serial loop without speculation, paying the full propose+Judge overhead every round.

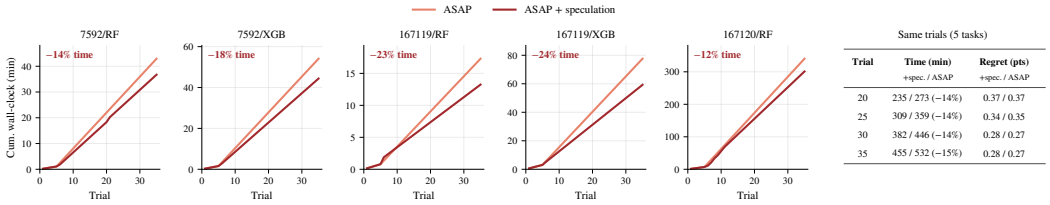


Figure 4: ASAP vs. ASAP + speculation on the five speculated HPOBench combos (same warm-up points): cumulative wall-clock to complete each trial, composed from logged per-round times. Right: totals and mean regret (validation error above the best error any method found, in accuracy points) at matching trials.

Baselines and our method. We compare against (1) seven statistical tools (GP, TPE, Optuna, SKOPT, TuRBO, SMAC, DNGO); (2) LLAMBO, an LLM-as-proposer method; and (3) frontier LLM agents following the AgentHPO protocol (Liu et al., 2025), in which one LLM (Claude Fable 5, Claude Opus 5, or GPT-5.6 Sol) proposes each configuration directly. We report **ASAP**, the integrated selection of Module 1 run in the serial loop, which applies to every task and pays the full propose+Judge overhead every round; it is the ASAP curve in Fig. 3 and in the landscape and case-study analyses. **ASAP + speculation** adds Module 2's speculation and is evaluated in RQ2 on the tasks where it applies.

Setup and Evaluation Metrics. All methods share initialization, seeds, and budgets (5 initial + 30 iterations; seed counts per method in Appendix B.4), and all LLM-based methods use GPT-3.5-turbo-0125 (Appendix B.4). We report normalized regret, $\text{Regret}_t = \big(s^*_{\max} - \max_{h\in\mathcal{H}_t} f(h)\big)/\big(s^*_{\max} - s^*_{\min}\big)$, where $\mathcal{H}_t$ is the set of configurations evaluated up to trial t and $s^*_{\max}, s^*_{\min}$ are the best and worst scores observed on the task across all methods, so regret values are relative to this method set. The **primary metric is regret vs. end-to-end wall-clock**; regret vs. iteration is auxiliary.

5.2 RQ1: Effectiveness and Generalization

ASAP dominates every baseline, even without speculation. **Result 1:** *ASAP reaches a far lower regret floor (0.026) than every baseline on both axes, including AgentHPO-style agents driven by frontier LLMs, even in its serial loop.* ASAP's final regret is about half that of the best baselines, LLAMBO (0.051) and TuRBO (0.058), and the gap holds on the wall-clock axis (0.027 vs. 0.054

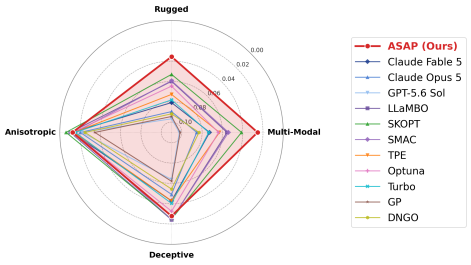


Figure 6: Descriptive analysis: mean normalized regret at iteration 35 on the tasks that most strongly express each landscape feature, computed post hoc from existing evaluations (outer = lower regret).

Table 1: Ablations at the 5 init + 30-iteration budget. (a) Final normalized regret, mean over all 28 tasks; the selector rows keep ASAP's candidate pool C_n and replace only the Judge (uniform selection is over the 17 candidates, so LLAMBO's batch of 10 carries more weight than each single-candidate tool). (b) Same warm-up points; regret is the validation error above the best error any method found, in accuracy points, at trial 35; wall-clock is the total over the five combos.

(a) Integration: all 28 tasks, serial loop

Variant	Regret ↓
ASAP (pool + LLM Judge)	**0.026**
Single LLM proposer, no pool (LLAMBO)	0.051
Pool + uniform-random selector	0.046
Pool + GP-EI selector	0.046

(b) Speculation: five $\rho \leq 1$ HPOBench combos

Variant	Regret (pts) ↓	Wall (min) ↓	Accepted
ASAP	0.27	532	–
ASAP + speculation	0.28	**455** (−15%)	97%

and 0.060 at 35,000 min; Fig. 3). A stronger stand-alone model does not close the gap: the frontier agents descend fastest in the first iterations but plateau at 0.065–0.095 by iteration 35, the signature of a strong prior that no surrogate refines. Since ASAP pays its full overhead here, the advantage comes from integration alone and holds from minutes-long HPOBench evaluations to multi-hour PD1 training.

The LLM Judge beats random selection from the first round. **Result 2:** *On identical candidate pools, the default GPT-3.5 Judge picks the pool's best candidate* 2.5× *as often as random selection, and GPT-4.1 and GPT-5.6 Judges* 4× *as often.* On these replayed pools, every Judge stays above random at every trial (Fig. 5), consistent with the per-round precondition Eq. equation 5 of Sec. 3.1; pick rates are a rank-based proxy for that expected-regret condition. The advantage is already about 2× at the first round, when there is almost no history to fit, consistent with the LLM's pretraining prior over hyperparameters (Sec. 3.2).

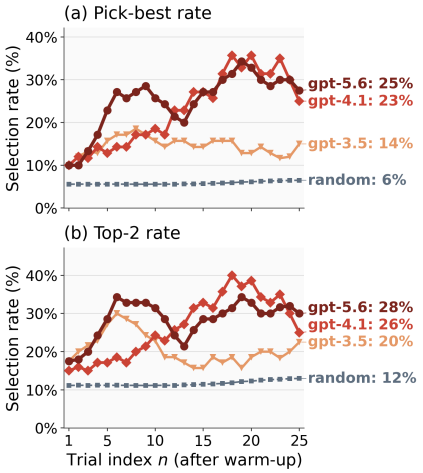


Figure 5: Offline replay: pick-best and top-2 rates on the same 250 frozen candidate pools (10 PD1 studies × 25 trials) for Judges driven by GPT-3.5, GPT-4.1, and GPT-5.6 vs. uniform-random selection (grey).

The best tool changes with the landscape; the Judge need not know which. **Result 3:** *Tool rankings reshuffle across landscape features, and no fixed tool wins more than half the tasks on any of them; ASAP, which never picks a tool, stays in the top group on all four.* In a post-hoc, descriptive analysis, we characterize tasks along four landscape features computed from existing evaluations (Appendix C.7). On Rugged and Multi-Modal tasks ASAP even beats the per-task oracle that picks the best BO tool in hindsight, while the frontier agents behave like one more fixed tool (Fig. 6).

5.3 RQ2: Decomposing the Gains

Ablation. **Result 4:** *Integration sets the regret floor and speculation saves wall-clock: removing integration nearly doubles regret, while removing speculation raises wall-clock at nearly unchanged regret. Within integration, the pool alone beats every single tool, and the Judge lowers regret further.* Keeping ASAP's pool but replacing the Judge by random or GP-EI selection gives 0.046 (Table 1a), already below the best single tool, so the pool's diversity carries a portfolio gain; the Judge lowers regret further to 0.026, mostly on the hardest tasks, where the best candidates are LLM proposals a GP surrogate does not rank highly. Over all but one study, commits are split between the LLM proposer (54%) and the seven BO tools (46%); since LLAMBO supplies 10 of the 17 candidates (59%), the Judge picks it slightly less often than its share of the pool, so it draws on the whole pool

rather than favoring the LLM proposer (Appendix C.1). Swapping GPT-3.5 for GPT-5.6 leaves the regret essentially unchanged (Appendix C.2).

Speculation behavior. **Result 5:** *Where it applies, speculation completes the same trials in 12–24% less time while keeping regret on par with the serial loop.* An accepted round saves the shorter of T_{ML} and the propose+judge overhead (Sec. 4.3), so we enable speculation where the overhead is at least as long as the evaluation, $\rho = T_{\text{ML}}/(T_{\text{tool}} + T_{\text{judge}}) \leq 1$. This holds for all HPOBench combos and no PD1 study, whose $\sim$10 s overhead caps the possible saving at about 6%; among HPOBench we leave out the four combos whose runs are too long for additional speculative experiments (7–12 h per serial run). On the other five, speculation saves 15% of wall-clock in total at matching regret, with 97% of speculative rounds accepted (Table 1b, Fig. 4), and the searched hyperparameter distributions of the two variants overlap by 89–94% (Appendix C.3). This low impact is empirical: the accept test bounds the prediction error, not the change in the next selection. Speculation is thus worthwhile only near $\rho \approx 1$, whereas integration applies across the whole non-trivial range, including multi-hour training (Fig. 7; Appendix C.4).

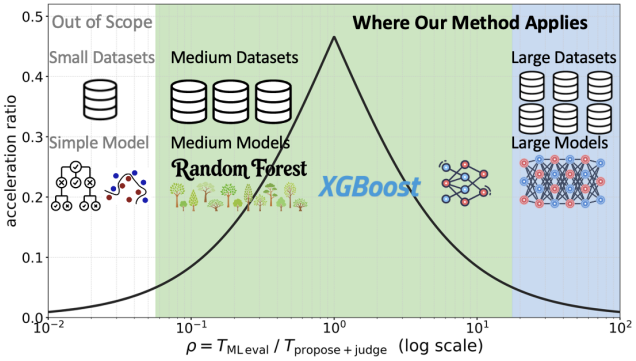


Figure 7: Where ASAP applies and where speculation helps, vs. $\rho = T_{\text{ML}}/(T_{\text{tool}} + T_{\text{judge}})$ (log scale). The curve is analytical: the per-round saving $p_{\text{acc}} \min(\rho, 1)/(1 + \rho)$ at an accept rate $p_{\text{acc}} = 0.93$ (97% is measured on the five speculated combos). Speculation pays off near $\rho \approx 1$; integration applies across the non-trivial range, and ASAP is out of scope only for trivial evaluations.

Sensitivity to the accept threshold. **Result 6:** *The speedup does not hinge on τ: the Judge's predictions are accurate enough that a much tighter gate keeps most of the saving.* On the five speculated combos, the Judge's relative prediction error has a median of 1.2%, and with the logged trajectories held fixed, $\tau = 0.10$ would still accept 90% of rounds and keep a 13.7% saving, against 15.3% at $\tau = 0.30$.

5.4 RQ3: Case Studies for Interpreting Optimization Behaviors

Every single proposer concentrates in a sub-region; the Judge integrates across them. **Result 7:** *Each single proposer concentrates in a biased sub-region, so none reliably reaches the optimum; by switching between proposers, only the Judge-integrated ASAP queries the optimum in all three case studies.* Fig. 11 (Appendix C.5) projects three searches (PD1/Transformer, PD1/Wide-ResNet, HPOBench/XGBoost) onto two hyperparameters: each proposer's queries cluster inside its own region, while the Judge commits different proposers' candidates in different rounds and reaches the optimum.

6 Conclusion

We presented **ASAP** for HPO: an LLM-as-Judge integrates diverse optimizers so complementary biases cover problems no single proposer handles. Integration alone attains the lowest regret per unit wall-clock across 28 tasks and stays in the top group on every landscape axis, and cross-iteration speculation with a latency-safe fallback adds a 12–24% wall-clock saving where evaluation and overhead are of similar size. These results suggest that an LLM contributes most to HPO not as one more proposer but as a selector over complementary ones, and that such agents should be judged by the regret they reach per unit wall-clock rather than per iteration. Future work includes a decision-level accept test for speculation, larger and more heterogeneous tool pools, and measuring the prompt-caching benefit on serving stacks that reuse prefix KV caches. Limitations are discussed in Appendix D.

DISCLOSURE OF THE USE OF GENERATIVE AI

Following the ICLR 2027 AI Policy for Authors, we disclose our use of generative AI tools in the preparation of this work. This section does not count toward the page limit.

Research ideation and execution (required disclosure). In this work, we used generative AI tools to help refine research hypotheses and to assist in implementing methods. Specifically, (i) during research ideation, we used LLM-based assistants as a brainstorming aid to stress-test and refine our framing of the two limitations of existing LLM-for-HPO methods (single-tool replacement and per-iteration evaluation); the core research questions and the design of ASAP's components (the tool-integrating Judge, the prefix-stable prompt, and speculation parallelism) were conceived, decided, and validated by the authors. (ii) During execution, we used AI coding assistants to help write and debug parts of the experiment and analysis code; all AI-generated code was reviewed, verified, and tested for correctness by the authors before use. We have not used generative AI tools to generate synthetic datasets, develop theoretical models or conceptual frameworks, formulate or prove mathematical claims, clean or reformat datasets, perform qualitative or thematic data analysis, or interpret experimental results; the remaining required-disclosure tasks (e.g., translation) are not applicable to this work.

Retrieval and discovery (recommended disclosure). We used generative AI tools to search for and identify relevant literature and to summarize existing work when compiling the related-work discussion. Every reference cited in this paper was retrieved, read, and verified against its original source by the authors.

Writing assistance (recommended disclosure). We used generative AI tools to polish the writing, i.e., to improve the grammar, clarity, and readability of author-written text and to help format references. All scientific content, claims, and conclusions were written and are fully owned by the authors, and all AI-assisted edits were reviewed by the authors.

LLMs as the object of study. Separately from the research workflow above, an LLM (GPT-3.5-turbo-0125) is a *component of the system under study*: it serves as the Judge inside ASAP and as the proposer in the LLAMBO baseline, exactly as described in the main text; Claude Fable 5, Claude Opus 5, and GPT-5.6 Sol are likewise used only as the proposer inside the frontier-LLM-agent baselines. This usage is part of the proposed method and its evaluation, not an authoring aid.

We have reviewed all AI-assisted work, and the authors take full responsibility for the entire content of this paper.

REFERENCES

Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework, 2019. URL https://arxiv.org/abs/1907.10902.

James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(10):281–305, 2012. URL http://jmlr.org/papers/v13/bergstra12a.html.

James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K. Weinberger (eds.), *Advances in Neural Information Processing Systems*, volume 24. Curran Associates, Inc., 2011. URL https://proceedings.neurips.cc/paper_files/paper/2011/file/86e8f7ab32cfd12577bc2619bc635690-Paper.pdf.

Eric Brochu, Matthew W. Hoffman, and Nando de Freitas. Portfolio allocation for bayesian optimization, 2011. URL https://arxiv.org/abs/1009.5419.

Yizhou Chi, Yizhang Lin, Sirui Hong, Duyi Pan, Yaying Fei, Guanghao Mei, Bangbang Liu, Tianqi Pang, Jacky Kwok, Ceyao Zhang, Bang Liu, and Chenglin Wu. Sela: Tree-search enhanced llm agents for automated machine learning, 2024. URL https://arxiv.org/abs/2410.17238.

Nan Ding, Youhan Fang, Ryan Babbush, Changyou Chen, Robert D. Skeel, and Hartmut Neven. Bayesian sampling using stochastic gradient thermostats. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Weinberger (eds.), *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014. URL https://proceedings.neurips.cc/paper_files/paper/2014/file/b610047c85e73cb7ec04fd36ec503f93-Paper.pdf.

Katharina Eggensperger, Philipp Müller, Neeratyoy Mallik, Matthias Feurer, René Sass, Aaron Klein, Noor Awad, Marius Lindauer, and Frank Hutter. Hpobench: A collection of reproducible multi-fidelity benchmark problems for hpo, 2022. URL https://arxiv.org/abs/2109.06716.

David Eriksson, Michael Pearce, Jacob R Gardner, Ryan Turner, and Matthias Poloczek. Scalable global optimization via local bayesian optimization, 2020. URL https://arxiv.org/abs/1910.01739.

David Ginsbourger, Rodolphe Le Riche, and Laurent Carraro. *Kriging Is Well-Suited to Parallelize Optimization*, pp. 131–162. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010. ISBN 978-3-642-10701-6. doi: 10.1007/978-3-642-10701-6_6. URL https://doi.org/10.1007/978-3-642-10701-6_6.

Yilin Guan, Qingfeng Lan, Sun Fei, Dujian Ding, Devang Acharya, Chi Wang, William Yang Wang, and Wenyue Hua. Dynamic speculative agent planning, 2025. URL https://arxiv.org/abs/2509.01920.

Siyuan Guo, Cheng Deng, Ying Wen, Hechang Chen, Yi Chang, and Jun Wang. Ds-agent: Automated data science by empowering large language models with case-based reasoning, 2024. URL https://arxiv.org/abs/2402.17453.

Taicheng Guo, Hai Wang, ChaoChun Liu, Mohsen Golalikhani, Xin Chen, Xiangliang Zhang, and Chandan K Reddy. Mtsql-r1: Towards long-horizon multi-turn text-to-sql via agentic training. *arXiv preprint arXiv:2510.12831*, 2025.

Taicheng Guo, Nitesh V Chawla, Olaf Wiest, and Xiangliang Zhang. Autollmresearch: Training research agents for automating llm experiment configuration–learning from cheap, optimizing expensive. *arXiv preprint arXiv:2605.11518*, 2026.

Sirui Hong, Yizhang Lin, Bang Liu, Bangbang Liu, Binhao Wu, Ceyao Zhang, Chenxing Wei, Danyang Li, Jiaqi Chen, Jiayi Zhang, Jinlin Wang, Li Zhang, Lingyao Zhang, Min Yang, Mingchen Zhuge, Taicheng Guo, Tuo Zhou, Wei Tao, Xiangru Tang, Xiangtao Lu, Xiawu Zheng, Xinbing Liang, Yaying Fei, Yuheng Cheng, Zhibin Gou, Zongze Xu, and Chenglin Wu. Data interpreter: An llm agent for data science, 2024. URL https://arxiv.org/abs/2402.18679.

Kirthevasan Kandasamy, Akshay Krishnamurthy, Jeff Schneider, and Barnabas Poczos. Parallelised bayesian optimisation via thompson sampling. In Amos Storkey and Fernando Perez-Cruz (eds.), *Proceedings of the Twenty-First International Conference on Artificial Intelligence and Statistics*, volume 84 of *Proceedings of Machine Learning Research*, pp. 133–142. PMLR, 09–11 Apr 2018. URL https://proceedings.mlr.press/v84/kandasamy18a.html.

Pascal Kerschke. Comprehensive feature-based landscape analysis of continuous and constrained optimization problems using the r-package flacco, 2017. URL https://arxiv.org/abs/1708.05258.

Eric Hans Lee, Valerio Perrone, Cedric Archambeau, and Matthias Seeger. Cost-aware bayesian optimization, 2020. URL https://arxiv.org/abs/2003.10870.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding, 2023. URL https://arxiv.org/abs/2211.17192.

Yucen Lily Li, Tim G. J. Rudner, and Andrew Gordon Wilson. A study of bayesian neural network surrogates for bayesian optimization, 2024. URL https://arxiv.org/abs/2305.20028.

Marius Lindauer, Katharina Eggensperger, Matthias Feurer, André Biedenkapp, Difan Deng, Carolin Benjamins, Tim Ruhopf, René Sass, and Frank Hutter. Smac3: A versatile bayesian optimization package for hyperparameter optimization, 2022. URL https://arxiv.org/abs/2109.09831.

Siyi Liu, Chen Gao, and Yong Li. Large language model agent for hyper-parameter optimization, 2025. URL https://arxiv.org/abs/2402.01881.

Tennison Liu, Nicolás Astorga, Nabeel Seedat, and Mihaela van der Schaar. Large language models to enhance bayesian optimization, 2024. URL https://arxiv.org/abs/2402.03921.

Yeonju Ro, Haoran Qiu, Íñigo Goiri, Rodrigo Fonseca, Ricardo Bianchini, Aditya Akella, Zhangyang Wang, Mattan Erez, and Esha Choukse. Sherlock: Reliable and efficient agentic workflow execution, 2025. URL https://arxiv.org/abs/2511.00330.

scikit optimize. https://scikit-optimize.github.io/stable/.

Bobak Shahriari, Ziyu Wang, Matthew W. Hoffman, Alexandre Bouchard-Côté, and Nando de Freitas. An entropy search portfolio for bayesian optimization, 2015. URL https://arxiv.org/abs/1406.4625.

Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. Practical bayesian optimization of machine learning algorithms, 2012. URL https://arxiv.org/abs/1206.2944.

Jasper Snoek, Oren Rippel, Kevin Swersky, Ryan Kiros, Nadathur Satish, Narayanan Sundaram, Md. Mostofa Ali Patwary, Prabhat, and Ryan P. Adams. Scalable bayesian optimization using deep neural networks, 2015. URL https://arxiv.org/abs/1502.05700.

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, 2024. URL https://arxiv.org/abs/2306.08107.

Zi Wang, George E. Dahl, Kevin Swersky, Chansoo Lee, Zachary Nado, Justin Gilmer, Jasper Snoek, and Zoubin Ghahramani. Pre-trained gaussian processes for bayesian optimization, 2024. URL https://arxiv.org/abs/2109.08215.

Shuhei Watanabe. Tree-structured parzen estimator: Understanding its algorithm components and their roles for better empirical performance, 2026. URL https://arxiv.org/abs/2304.11127.

Christopher Williams and Carl Rasmussen. Gaussian processes for regression. In D. Touretzky, M.C. Mozer, and M. Hasselmo (eds.), *Advances in Neural Information Processing Systems*, volume 8. MIT Press, 1995. URL https://proceedings.neurips.cc/paper_files/paper/1995/file/7cce53cf90577442771720a370c3c723-Paper.pdf.

David H Wolpert and William G Macready. No free lunch theorems for optimization. *IEEE transactions on evolutionary computation*, 1(1):67–82, 2002.

Naimeng Ye, Arnav Ahuja, Georgios Liargkovas, Yunan Lu, Kostis Kaffes, and Tianyi Peng. Speculative actions: A lossless framework for faster agentic systems, 2026. URL https://arxiv.org/abs/2510.04371.

Michael R. Zhang, Nishkrit Desai, Juhan Bae, Jonathan Lorraine, and Jimmy Ba. Using large language models for hyperparameter optimization, 2024. URL https://arxiv.org/abs/2312.04528.

APPENDIX

APPENDIX CONTENTS

A Why Integrate Rather Than Replace: Full Analysis 13
B Implementation Details 15
B.1 Benchmark Details 15
B.2 Hyperparameter Search Spaces 15
B.3 LLM-as-Judge Prompt Design 16
B.4 Experimental Protocol Details 17
C Additional Experimental Results 18
C.1 Per-Tool Selection Fractions 18
C.2 Sensitivity to the LLM Behind ASAP 19
C.3 Speculation Does Not Change the Search Distribution 19
C.4 Where Speculation Applies 20
C.5 Case-Study Visualizations 20
C.6 Performance on Trivial-Eval ML Experiments (< 5 s) 21
C.7 Optimization-Landscape Features and Per-Task Classification 22
D Limitations 23

A WHY INTEGRATE RATHER THAN REPLACE: FULL ANALYSIS

This section contains the complete analysis summarized in Sec. 3.1 of the main text, using the setup and notation defined there.

Single-tool regret and the grand-mean floor. As in the main text, tool i is a proposer $g_i(D_t)$ and running it alone for n rounds on f incurs the simple regret $R_i(f, n) = f^* - \max_{t \le n} f(x_t)$ with $x_t = g_i(D_t)$ (Eq. equation 1), where $f^* = \max_{x \in \mathcal{X}} f(x)$ and D_t is the history available at round t. Each tool is designed around a family of landscapes and carries the corresponding bias: a GP with a stationary kernel excels on smooth objectives yet squanders its budget on rugged ones where TPE or SMAC thrive, and vice versa. Because $\mathcal{F}$ spans heterogeneous task structures, no single tool minimizes $R_i(f, n)$ uniformly over $\mathcal{F}$.[1]

Aggregating over the suite, define $\bar{R}_i(\mathcal{F}, n) = \frac{1}{M} \sum_{f \in \mathcal{F}} R_i(f, n)$ as tool i's mean regret on $\mathcal{F}$. A practitioner who does not know in advance which tool fits which task must fall back on a choice that does not depend on the task; the canonical such choice is to pick a tool uniformly at random, whose expected mean regret is the *grand-mean regret*

$$\bar{R}(\mathcal{F}, n) \triangleq \frac{1}{K} \sum_{i=1}^{K} \bar{R}_i(\mathcal{F}, n), \tag{8}$$

where K is the number of available tools and $M = |\mathcal{F}|$ is the number of tasks. The limitation is epistemic rather than one of raw capability: the tool that fits f can of course beat $\bar{R}(\mathcal{F}, n)$ on that task, but a rule with no information that distinguishes the tasks has expected mean regret exactly $\bar{R}(\mathcal{F}, n)$; this uninformed-selection baseline is the reference point of Eq. equation 2 in the main text. Replacing a tool by an LLM proposer does not change this: the LLM is one more biased proposer whose mean regret $\bar{R}_{\mathrm{LLM}}(\mathcal{F}, n)$ is just another row of the ladder. Dynamic selection

[1] This echoes the classical No-Free-Lunch results for optimization (Wolpert & Macready, 2002), although those require the function class to be closed under permutation, a condition that the structured objectives considered here do not satisfy.

(Eq. equation 3) contains both the static and the random rule as special cases. The analysis below does not bound any class of selection rules; it fixes the per-round, shared-history analogue of this reference point and states the (mild) condition under which a Judge that selects among the tools' *results* beats it (Eq. equation 5). Sec. 5 then shows empirically that clearing this per-round condition coincides with lower trajectory-level regret.

Per-task quantities. All per-round quantities below are indexed by the task. At round n of a run on task f, the committed history is $D_n(f)$; given it, each tool $i \in [K]$ proposes a batch $G_i(D_n(f)) \subset \mathcal{X}$, and the candidate set is their union $C_n(f) = \bigcup_{i\in[K]} G_i(D_n(f))$ (Eq. equation 4), where $[K] = \{1, \dots, K\}$ indexes the active pool. Both $D_n(f)$ and $C_n(f)$ depend on f and on the trajectory that produced $D_n(f)$; in particular $|C_n(f)|$ may differ across tasks and rounds, and no candidate set is shared across the suite. We treat $C_n(f)$ as an indexed family, so coincident proposals are counted with multiplicity (in our pool $|C_n(f)| = 17$: the seven statistical tools emit one candidate each and the LLM proposer emits a batch of ten, Sec. 3.2). A Judge h selects $x_n(f) = h(C_n(f), D_n(f))$; as a reference, the *naive Judge* draws $c \sim \mathrm{Unif}(C_n(f))$.

The pool mean as the per-round reference point. Let $r_c(f, n) = f^* - f(c)$ denote the per-round regret of evaluating $c \in C_n(f)$ on task f, and define the task's *pool mean* and its average over the suite,

$$\bar{r}(f, n) \triangleq \frac{1}{|C_n(f)|} \sum_{c\in C_n(f)} r_c(f, n), \qquad \bar{r}(\mathcal{F}, n) \triangleq \frac{1}{M} \sum_{f\in\mathcal{F}} \bar{r}(f, n). \tag{9}$$

The naive Judge attains exactly this quantity, task by task and hence on average:

$$\mathbb{E}_{c\sim\mathrm{Unif}(C_n(f))}\big[f^* - f(c) \,\big|\, D_n(f)\big] = \frac{1}{|C_n(f)|} \sum_{c\in C_n(f)} r_c(f, n) = \bar{r}(f, n), \tag{10}$$

$$\frac{1}{M} \sum_{f\in\mathcal{F}} \mathbb{E}_{c\sim\mathrm{Unif}(C_n(f))}\big[f^* - f(c) \,\big|\, D_n(f)\big] = \bar{r}(\mathcal{F}, n). \tag{11}$$

Eq. equation 10 holds for every $f \in \mathcal{F}$ and is an elementary identity (the expectation of a uniform draw is the pool mean), stated conditionally on each task's own history and only then averaged in Eq. equation 11; it is not a statement about any class of selection rules. We record it because it fixes the correct *reference point* for the integrated agent: $\bar{r}(\mathcal{F}, n)$ is the shared-history, per-round counterpart of the uninformed baseline $\bar{R}(\mathcal{F}, n)$. Two differences from Eq. equation 8 must be kept in mind. First, $\bar{R}$ is a trajectory-level quantity in which each tool runs on its own history, whereas $\bar{r}$ is evaluated on the single committed history the integrated agent has built; the two are different numbers, and Eq. equation 10 does not relate them. Second, $\bar{r}(f, n)$ averages over *candidates*, so a tool that emits a batch is weighted by the number of candidates it emits; it is the pool average, not the average over tool identities (which would draw a tool uniformly and evaluate that tool's top candidate). In words: pooling the tools and then selecting uniformly at random recovers the pool average, so **integration alone is not enough**; whatever the integrated agent gains over the pool average in a round must come from the Judge.

From random to informed: the condition for integration to win. Let

$$\bar{r}_{\mathrm{int}}(\mathcal{F}, n) = \frac{1}{M} \sum_{f\in\mathcal{F}} \big[f^* - f(x_n(f))\big], \qquad x_n(f) = h(C_n(f), D_n(f)), \tag{12}$$

denote the integrated agent's per-round mean regret with a real (non-random) Judge h, each task evaluated on its own history. Comparing Eqs. equation 12 and equation 11, both per-round, shared-history quantities, gives the criterion in Eq. equation 5 of the main text: $\bar{r}_{\mathrm{int}}(\mathcal{F}, n) < \bar{r}(\mathcal{F}, n)$ holds exactly when, averaged over the suite, h selects from $C_n(f)$ better than a uniform draw does. **What the criterion does and does not claim.** We stress that Eq. equation 5 is a *per-round, shared-history* statement: both $\bar{r}_{\mathrm{int}}(\mathcal{F}, n)$ and $\bar{r}(\mathcal{F}, n)$ are computed, task by task, on the *same* committed history $D_n(f)$ that the integrated agent has already built, and compare the Judge against uniform selection over the *same* candidate set $C_n(f)$. Under this criterion the Judge need not be optimal: clearing this single, deliberately mild bar suffices for the integrated agent to beat uniform selection over $C_n(f)$ at round n. This is weaker than, and should not be conflated with, a claim about the trajectory-level grand mean $\bar{R}(\mathcal{F}, n)$ in Eq. equation 8, where each tool runs an entire trajectory on its own

history; we do not prove that a per-round win composes into a trajectory-level win, since the histories themselves diverge once selection begins, nor do we claim a lower bound for static or task-agnostic selection rules. What we *do* claim, and verify empirically (Sec. 5.2, Fig. 5), is that the per-round precondition Eq. equation 5 holds at *every* round, and that the resulting committed trajectory attains lower end-to-end regret than every baseline (RQ1). The per-round criterion is thus the *mechanism* we can characterize cleanly; its compounding into the global regret gain is an empirical result, not a theorem.

Two facts make this criterion informative rather than vacuous. First, Eq. equation 10 shows that integration *alone*, pooling tools and selecting at random, merely recovers the pool average *per round*, so any per-round gain over that average must come from the Judge, not from having more tools. (Across a whole run the pool's diversity does yield a portfolio gain: random selection over $C_n(f)$ already beats every single tool, and the Judge lowers that regret further; Table 1.) Second, the Judge's task is *discriminative*, not generative: it only ranks the small, historically grounded set $C_n(f)$, whereas a proposer must generate good points over all of $\mathcal{X}$. ASAP thereby converts a hard generative inductive bias into a far milder discriminative one, which is why the criterion in Eq. equation 5 is routinely satisfiable. Moreover, the more heterogeneous $\mathcal{F}$ becomes, the larger the gap between the per-round pool average and the per-task best, so the problem grows harder for any task-agnostic strategy while the Judge's task (ranking the same $|C_n(f)|$ candidates) does not.

B Implementation Details

This section collects everything needed to reproduce our experiments: the benchmark tasks we evaluate on (Sec. B.1), the full hyperparameter search spaces (Sec. B.2), the prompt design used by the LLM-as-Judge inside Module 1 (Sec. B.3), and the experimental protocol details (Sec. B.4).

B.1 Benchmark Details

Table 2 lists the 28 (task, model) combinations used in our evaluation. "ML Eval Time" is the median measured wall-clock cost of a single configuration evaluation: real training time for HPOBench, and pre-computed GPU training wall (the cost a practitioner actually pays per trial in deep-learning HPO) for PD1, which the benchmark surfaces from ∼50k recorded training runs. "Objective" is the metric the benchmark returns (validation accuracy for HPOBench classification; validation error rate from the pre-computed training run for PD1). All tasks are pure HPO with a 4-D search space; full ranges appear in Sec. B.2.

The pool deliberately covers (i) classical tabular learners under real training cost (HPOBench / XGBoost, RandomForest), (ii) modern CNN architectures of practical interest (Wide-ResNet, ResNet-50, Simple-CNN, Max-Pooling-CNN) on standard vision benchmarks at the batch sizes used by their authors, and (iii) modern Transformer-style training on the kind of workloads that dominate today's HPO cost: language modeling (LM1B), machine translation (WMT En–De), and protein modeling (Uniref50). Per-trial cost spans four orders of magnitude, from ∼9 s for the cheapest HPOBench task to ∼48 h for the most expensive (WMT En–De translation) and ∼25 h for ImageNet / ResNet-50, so every operating regime relevant to ASAP's walltime-centered design appears in the evaluation.

B.2 Hyperparameter Search Spaces

HPOBench (XGBoost / RandomForest on OpenML tabular tasks). All HPOBench combos in Table 2 share a 4-D search space per model:

- **XGBoost**: {colsample_bytree: [linear, 0.1, 1.0], eta: [log, 2^{-10}, 1.0], max_depth: [log, 1, 50], reg_lambda: [log, 2^{-10}, 1024]}
- **RandomForest**: {max_depth: [log, 1, 50], max_features: [linear, 0.0, 1.0], min_samples_leaf: [linear, 1, 20], min_samples_split: [log, 2, 128]}

PD1 (deep-learning HPO replay). All PD1 architectures (Wide-ResNet, ResNet-50, Simple-CNN, Max-Pooling-CNN, Transformer / xFormer-Translate) share the same 4-D optimizer search space exposed by the PD1 study schema:

Benchmark	Domain	Dataset	Model	ML Eval Time (s)	Objective
HPOBench (raw mode: real training of sklearn-style tabular models on OpenML)					
HPOBench	Tabular	jasmine (OpenML 167119, 27k×6, 3-class)	RandomForest	9.5	Val. accuracy
HPOBench	Tabular	jasmine	XGBoost	37	Val. accuracy
HPOBench	Tabular	adult (OpenML 7592, 29k×105, binary)	RandomForest	13	Val. accuracy
HPOBench	Tabular	adult	XGBoost	19	Val. accuracy
HPOBench	Tabular	high-dim (OpenML 168908, 3.3k×1646)	RandomForest	29	Val. accuracy
HPOBench	Tabular	high-dim	XGBoost	238	Val. accuracy
HPOBench	Tabular	madeline (OpenML 167120, 58k×21, binary)	XGBoost	30	Val. accuracy
HPOBench	Tabular	madeline	RandomForest	85	Val. accuracy
HPOBench	Tabular	volkert (OpenML 168329, 39k×27, 100-class)	RandomForest	148	Val. accuracy
PD1 (replay of ∼50k pre-computed deep-learning HPO runs; "Eval Time" = pre-computed GPU training wall per trial)					
PD1	Vision	Fashion-MNIST (60k×28^2, 10-class)	Max-Pooling-CNN (bs=2048)	116	Val. error
PD1	Vision	MNIST (60k×28^2, 10-class)	Max-Pooling-CNN (bs=2048)	120	Val. error
PD1	Vision	Fashion-MNIST	Simple-CNN (bs=2048)	142	Val. error
PD1	Vision	MNIST	Simple-CNN (bs=2048)	166	Val. error
PD1	Vision	Fashion-MNIST	Max-Pooling-CNN (bs=256)	519	Val. error
PD1	Vision	MNIST	Max-Pooling-CNN (bs=256)	547	Val. error
PD1	Vision	Fashion-MNIST	Simple-CNN (bs=256)	592	Val. error
PD1	Vision	MNIST	Simple-CNN (bs=256)	718	Val. error
PD1	Vision	CIFAR-100 (50k×32^2, 100-class)	Wide-ResNet (bs=2048)	1,560	Val. error
PD1	Vision	CIFAR-10 (50k×32^2, 10-class)	Wide-ResNet (bs=2048)	2,338	Val. error
PD1	Vision	CIFAR-100	Wide-ResNet (bs=256)	2,484	Val. error
PD1	Vision	SVHN (73k×32^2, 10-class)	Wide-ResNet (bs=1024)	2,724	Val. error
PD1	Vision	CIFAR-10	Wide-ResNet (bs=256)	3,934	Val. error
PD1	Vision	SVHN	Wide-ResNet (bs=256)	4,157	Val. error
PD1	Vision	ImageNet (1.3M×224^2, 1000-class)	ResNet-50 (bs=512)	78,900	Val. error
PD1	Protein	Uniref50 (protein sequences)	Transformer (bs=128)	84,417	Val. error
PD1	Vision	ImageNet	ResNet-50 (bs=256)	90,517	Val. error
PD1	Language	LM1B (billion-word language modeling)	Transformer (bs=2048)	110,293	Val. error
PD1	Language	WMT En–De (machine translation)	xFormer-Translate (bs=64)	172,699	Val. error

Table 2: The 28 (task, model) combinations used in our evaluation. "ML Eval Time" is the median measured wall-clock cost of a single configuration evaluation: real model training for HPOBench, and the pre-computed GPU training wall replayed by PD1 (the actual cost a practitioner pays per trial in deep-learning HPO). All tasks are pure HPO with a 4-D search space (full ranges in Sec. B.2). The pool spans classical tabular learners (RandomForest, XGBoost), modern CNNs (Wide-ResNet, ResNet-50, Simple/Max-Pooling-CNN), and modern Transformers (language modeling, machine translation, protein), with per-trial cost ranging over four orders of magnitude (∼9 s to ∼48 h), entirely within the non-trivial-eval regime that ASAP's walltime-centered design targets.

- {lr_initial_value: [log, 10^{-5}, 10.0], lr_power: [linear, 0.1, 2.0], lr_decay_steps_factor: [linear, 0.01, 1.0], opt_momentum: [logit, 10^{-4}, 0.9999]}

PD1 trains the model with these optimizer hyperparameters; the dataset and architecture identify the study, and each lookup returns the recorded best validation error of the closest precomputed trial in HP space.

B.3 LLM-AS-JUDGE PROMPT DESIGN

The ICL Judge in Sec. 3.2 conditions on a textual problem context built from the *meta-features* of the target HPO task. The context includes the ML model name and task type (classification or regression) together with key dataset attributes: number of samples n, number of features d (and the ratio n/d), feature types, inter-feature correlations, and feature-label correlations. Dataset names are deliberately omitted to mitigate memorization risk in the LLM's pretraining corpus. The prompt further encodes the search-space semantics of each hyperparameter (name, range, scale) and the optimization state in D_n (round index, best-so-far, recent improvement), which supplies explore/exploit context; grounding the pretraining prior in the concrete problem steers, e.g., an XGBoost run on a small, high-dimensional dataset toward shallower, more regularized trees than one on a large dataset. On top of this context, the Judge prompt frames the prediction task in an *inductive* few-shot regression style: given a list of previously evaluated (HP configuration, performance) pairs and a candidate configuration, the model predicts the candidate's performance, which the outer loop turns into an Expected Improvement score from S sampled predictions per candidate (Sec. 3.2). Table 3 shows a concrete instantiated example, factored into the immutable prefix and the per-round suffix that the KV-cache-aware design of Sec. 4.2 exploits. The complete template, including system message, few-shot block, candidate list, and response format, is included with the released code.

Immutable prefix *(task description + committed demonstrations through round $n-1$; prefilled once; KV cache reused across rounds and candidates)*

```
The following are hyperparameter configurations for a XGBoost and
the corresponding performance measured in accuracy.  The model is
evaluated on a tabular classification task and the label contains 3
classes.  The tabular dataset contains 27025 samples and 6 features
(0 categorical, 6 numerical).  Your response should only contain the
predicted accuracy in the format ## performance ##.
-- observed history (D_n):  config → true accuracy --
Hyperparameter configuration:  colsample_bytree is 0.6, eta is
0.0500823341, max_depth is 4, reg_lambda is 0.5445010154  Performance:
## 0.867788 ##
Hyperparameter configuration:  colsample_bytree is 0.6, eta
is 0.2360640703, max_depth is 46, reg_lambda is 0.0382369760
Performance:  ## 0.930664 ##
...  (remaining committed demonstrations in D_n)
Hyperparameter configuration:  colsample_bytree is 0.5, eta is
0.0015978643, max_depth is 1, reg_lambda is 0.0012671438  Performance:
## 0.703050 ##
```

Round n suffix *(the query candidate; only this is prefilled at round n)*

```
Hyperparameter configuration:  colsample_bytree is 0.4, eta
is 0.5069423632, max_depth is 20, reg_lambda is 2.0054749362
Performance:  ## → predict ##
```

Table 3: KV-cache-aware Judge prompt. A long immutable prefix (task description + committed (config, performance) demonstrations up through round $n-1$) is prefilled once and its KV cache is reused; round n only appends a short suffix (the new candidate query), so only the suffix is prefilled. After round n commits, (x_n, y_n) extends the prefix for round $n+1$ without recomputing earlier tokens.

B.4 Experimental Protocol Details

We disable warm-starting and initialize all methods with the same five randomly sampled points per run. Each method runs two independent seeds, with 5 initial + 30 iteration trials per task per seed, so the full evaluation grid comprises 35 evaluations × 28 task-models × 10 methods × 2 seeds; reported curves average the two seeds and shaded bands denote ± 1 standard deviation. The three frontier LLM-agent baselines (AgentHPO protocol with Claude Fable 5, Claude Opus 5, and GPT-5.6 Sol; the Claude models served through DeepInfra's OpenAI-compatible endpoint, GPT-5.6 Sol through the OpenAI API, all at default reasoning settings) use the same initialization, budget, and search spaces but a single seed, since their per-run API cost is 3–5× that of the GPT-3.5 methods (about $7, $35, and $22 for GPT-5.6 Sol, Claude Fable 5, and Claude Opus 5 across the 28 tasks); out-of-range proposals from these models are clipped to the search-space bounds before evaluation. Hyperparameter search spaces appear in Sec. B.2. GPT-3.5-turbo-0125 serves as the proposer in LLAMBO and as the Judge in ASAP, matching the LLAMBO baseline so the comparison is on the integration mechanism rather than the underlying language model; a stronger LLM would likely improve all LLM-based methods further. All experiments run on identical CPU-only nodes (dual-socket AMD EPYC 7543 32-core processors, 64 cores per node, ∼250 GB RAM) so that the wall-clock comparison across methods is unaffected by hardware variation; LLM inference is served by the OpenAI API, so the same compute environment carries every method's full propose-evaluate loop. For PD1 we add the precomputed per-trial GPU training time to every method's measured wall-clock so the comparison reflects what an end-to-end deployment would actually pay. ASAP's speculative runs on the five HPOBench combos of Sec. 5.3 use one seed, with the same warm-up points as the serial run they are compared against. For the two ASAP variants on HPOBench, we compose wall-clock from each task's median evaluation time and mean propose+Judge time measured in the logs, so that system-load noise does not enter the comparison and the two variants differ only in whether the overhead overlaps the evaluation.

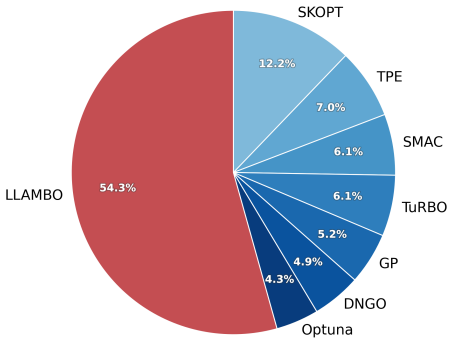


Figure 8: Fraction of Judge selections per tool, excluding one outlier study (PD1 CIFAR-10/Wide-ResNet, batch 2048), where the LLM proposer accounts for 83% of picks.

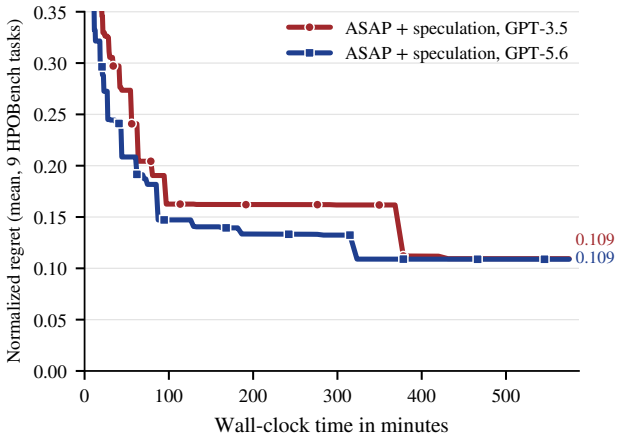


Figure 9: Swapping the LLM does not move ASAP's regret. ASAP + speculation with GPT-3.5 (red) vs. GPT-5.6 (blue) as both proposer and Judge, normalized regret vs. end-to-end wall-clock, same protocol and normalization as Fig. 3 restricted to the 9 HPOBench tasks; the labels at the right edge give the regret at the end of the range.

C Additional Experimental Results

This section reports auxiliary experiments and analyses that complement the main results in Sec. 5: the per-tool selection fractions (Sec. C.1), the sensitivity of ASAP to the LLM behind it (Sec. C.2), the per-hyperparameter agreement analysis showing speculation does not change the search distribution (Sec. C.3), the case-study visualizations (Sec. C.5), ASAP's behavior on the *trivial-eval* regime that the main scope excludes (Sec. C.6), and a breakdown of regret across qualitatively different optimization landscapes (Sec. C.7).

C.1 Per-Tool Selection Fractions

Fig. 8 shows the fraction of committed picks per tool from exact per-round attribution, supporting the pool-usage analysis in Sec. 5.3: the LLM proposer accounts for 54.3% of commits, the seven BO tools for the remaining 45.8%, with the BO contribution spread across the pool rather than concentrated on one tool.

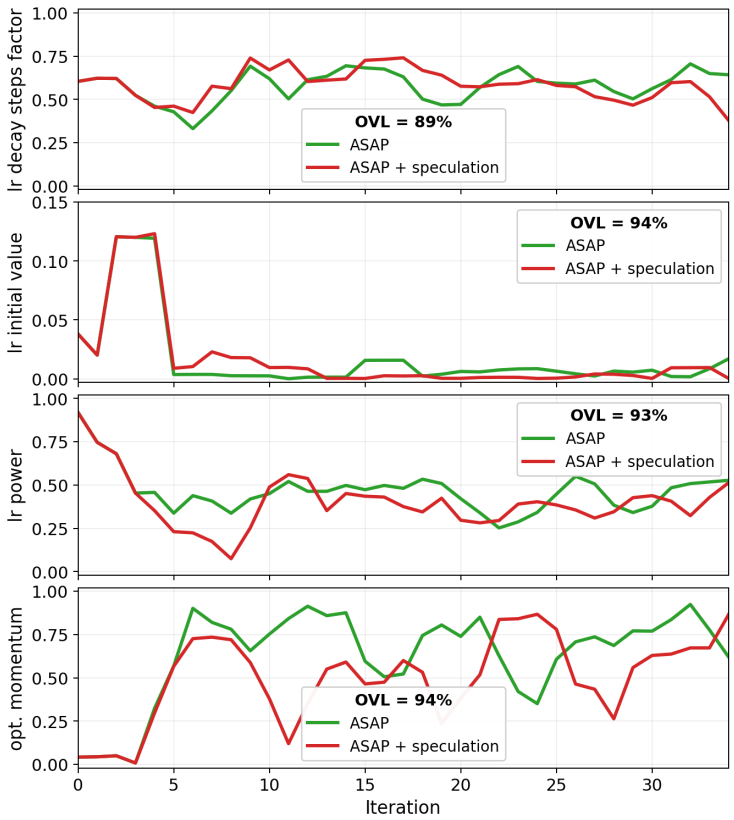


Figure 10: Speculation does not change which hyperparameters are explored. Per-round median of each evaluated hyperparameter (normalized to $[0, 1]$) for ASAP + speculation (red) vs. ASAP (green), pooled over the 19 PD1 tasks (one run each, same warm-up points). The medians follow the same trend with round-to-round fluctuations, and the pooled per-hyperparameter distributions overlap by 89–94% (OVL, per legend): speculation reorders only *when* the tool+Judge stage runs, not *where* the agent searches.

C.2 Sensitivity to the LLM Behind ASAP

Sec. 5.3 attributes ASAP's regret floor to integration, and Sec. 5.2 shows that the Judge's pick quality saturates beyond GPT-4.1 (Fig. 5). Here we ask the complementary question end-to-end: does a stronger LLM on *both* sides of the integration, proposer and Judge, lower the floor? We rerun ASAP + speculation on the 9 HPOBench tasks with GPT-5.6 in place of GPT-3.5 everywhere (one seed), keeping the tool catalog, initial configurations, budget $(5 + 30)$, and accept threshold of Fig. 3 (the GPT-5.6 run speculates on all nine tasks, the GPT-3.5 curve on the five of Sec. 5.3). Fig. 9 shows that it does not. The two curves track each other over the whole wall-clock range (mean absolute gap 0.018): GPT-5.6 leads through the middle of the run, and both end at the same regret (0.109). Per task, GPT-5.6 reaches a better final score on five tasks and a worse one on four, and no raw difference in best validation accuracy exceeds 0.6 points. End-to-end wall-clock is unchanged, since HPOBench rounds are bound by the BO tools rather than the LLM, while the API cost rises from about \$0.9 to \$7 per run. A frontier LLM therefore neither helps nor is needed: the regret floor is set by integrating the pool, not by the model that proposes or judges within it. This mirrors Fig. 3, where the frontier stand-alone agents plateau above ASAP, from the opposite direction: a stronger model does not substitute for integration, and integration does not need a stronger model.

C.3 Speculation Does Not Change the Search Distribution

Complementing the speculation-behavior discussion in Sec. 5.3, agreement between ASAP + speculation and ASAP holds on which configurations are explored: Fig. 10 pools the per-hyperparameter distribution of evaluated configs. For this check we run speculation on PD1 (the main comparison

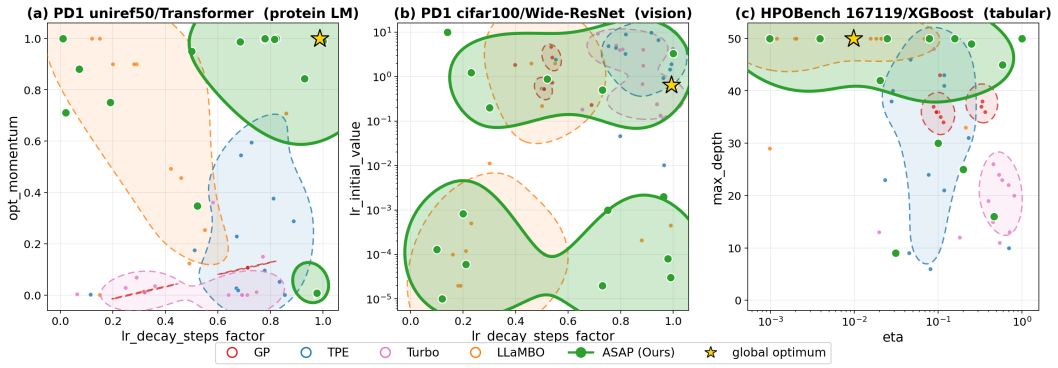


Figure 11: Case studies (RQ3) on PD1/Transformer, PD1/Wide-ResNet, and HPOBench/XGBoost, spanning single-proposer failure modes: no single proposer reaches the optimum (a, b), or one grazes it without converging (c). Dashed blobs are each proposer's KDE contour with a sparse query sample; green dots are ASAP's queries. Only ASAP reaches the gold-star optimum, driven by the Judge switching between proposers.

reports the serial loop there; Sec. 5.2), because its four tuned hyperparameters are shared across all 19 tasks, so evaluated configurations can be pooled in one space; each task contributes one ASAP + speculation run and the ASAP run with the same warm-up points. The median trajectories follow the same trend on all four hyperparameters, with round-to-round fluctuations expected from single runs, and the pooled distributions overlap by 89–94% (overlapping coefficient, OVL); the best score reached differs by 0.002 (median over tasks). Speculation thus reorders only *when* the tool+Judge stage runs (hiding it under T_{ML}), not *where* the agent searches, which is why the hidden latency carries no regret cost.

C.4 Where Speculation Applies

Fig. 7 shows how the benefit of speculation depends on ρ. At $\rho \approx 1$ (RandomForest, XGBoost), hiding one of two roughly equal stages saves the most per trial. At $\rho \gg 1$ (PD1, where T_{ML} reaches hours), the overhead is still hidden but is a small fraction of T_{ML}, so the relative speedup shrinks to a few percent, while integration keeps ASAP at the lowest regret (Fig. 3). At $\rho \ll 1$ (the sub-second trivial-eval regime of Sec. C.6), T_{ML} is too short to absorb the overhead. ASAP therefore targets the broad non-trivial-eval band, the operating point of modern model-training HPO: integration makes it the best method across this band, including multi-hour training where speculation adds little, and speculation adds a further saving where evaluation and overhead are comparable.

C.5 Case-Study Visualizations

Fig. 11 visualizes the three case studies analyzed in Sec. 5.4 of the main text; we detail each panel below.

(a) PD1/uniref50/Transformer. BO tools sit at low `opt_momentum`, well below the optimum at ≈1.0; LLAMBO reaches the highest momentum but its `lr_decay` support stops near 0.55, short of the optimum at 0.99. Only ASAP's queries span from LLAMBO's high-momentum band to the upper-right corner holding the star.

(b) PD1/cifar100/Wide-ResNet. BO tools concentrate at `lr_decay`≈0.7–0.95, just short of the optimum at 0.99; LLAMBO is broader in `lr_initial` but drifts to lower `lr_decay`≈0.2–0.55. ASAP queries past the BO cluster to the star on both seeds, also covering the LLAMBO band.

(c) HPOBench/167119/XGBoost. On a different benchmark the pattern reappears: BO tools concentrate at moderate `eta` (≈0.05–0.7); LLAMBO's region grazes the star (high `max_depth`=50, low `eta`) without converging to it. ASAP covers both regions and reaches the star on both seeds.

Across all three panels the same mechanism holds: the Judge integrates across the pool, committing one proposer's proposal one round and another's the next, so ASAP reaches the star even when no single proposer's region does (a, b) or when one grazes it imprecisely (c).

C.6 Performance on Trivial-Eval ML Experiments (< 5 s)

The main paper restricts to HPO tasks where per-evaluation cost is non-trivial ($\geq$ 5 s); under that regime ASAP's walltime advantage compounds because speculation hides per-round LLM/tool overhead under the ML evaluation. For completeness, we report results on the *trivial-eval* regime where each ML evaluation is sub-second (classical sklearn HPO on small tabular data, Bayesmark). Here speculation cannot meaningfully hide the agent's overhead (per-round wall is dominated by the LLM/tool stack rather than the model fit), so the design assumption that motivates ASAP no longer holds, and we expect cheap model-free BO baselines (e.g., TPE) to be highly competitive on the walltime axis. This experiment quantifies that boundary and grounds the limitations discussion.

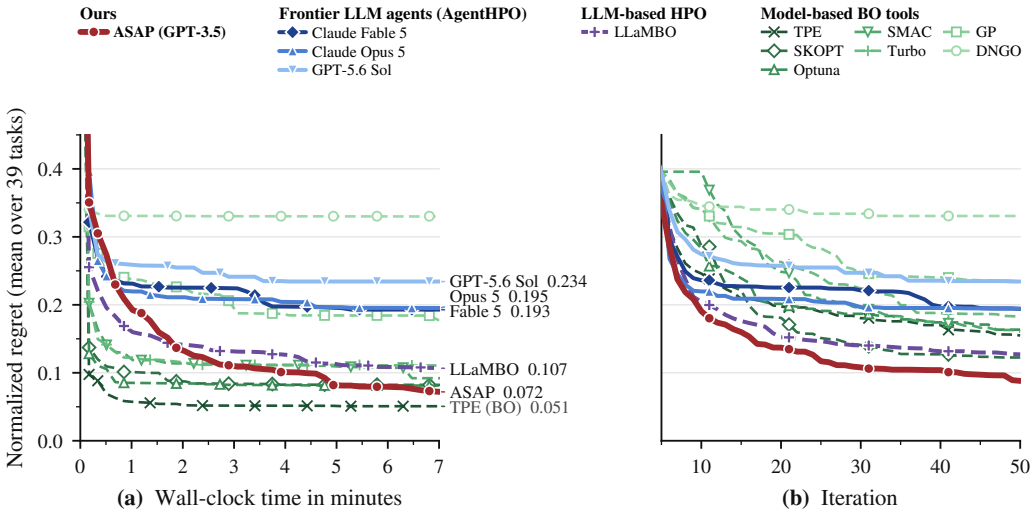
Setup. We use the full original Bayesmark grid: 4 CLF datasets (*breast*, *digits*, *iris*, *wine*) and 4 REG datasets (*diabetes*, *Griewank*, *KTablet*, *Rosenbrock*), each paired with 5 sklearn models (*ada*, *DT*, *MLP-sgd*, *RF*, *SVM*), giving $8 \times 5 = 40$ (task, model) combos. Regret is normalized per task by the cross-method best/worst window observed across all baselines and averaged over tasks. For the iteration panel (Fig. 12b), all methods run the same 5 init + 50 trial budget, and we average three seeds per method (two for ASAP on 34 of the 40 combos), plotting the first 50 trials. For the wall-clock panel (Fig. 12a), to give the BO baselines a fair wall-time budget comparable to the LLM methods' $\sim$7 min, we re-ran every BO tool with a much larger 5 init + 500 trials budget; the LLM methods keep their original 80-trial runs since they already consume the full 7 min window.

Hyperparameter Search Spaces The Bayesmark search space used in classical sklearn HPO:

- **SVM**: {C: [log, 1, 10^3], γ: [log, 10^{-4}, 10^{-3}], tolerance: [log, 10^{-5}, 10^{-1}] }
- **DecisionTree**: {max depth: [linear, 1, 15], min samples split: [logit, 0.01, 0.99], min samples leaf: [logit, 0.01, 0.49], min weight fraction leaf: [logit, 0.01, 0.49], max features: [logit, 0.01, 0.99], min impurity decrease: [linear, 0.0, 0.5] }
- **RandomForest**: {max depth: [linear, 1, 15], min samples split: [logit, 0.01, 0.99], min samples leaf: [logit, 0.01, 0.49], min weight fraction leaf: [logit, 0.01, 0.49], max features: [logit, 0.01, 0.99], min impurity decrease: [linear, 0.0, 0.5] }
- **MLP_SGD**: {hidden layer sizes: [linear, 50, 200], alpha: [log, 10^{-5}, 10^{1}], batch size: [linear, 10, 250], learning rate init: [log, 10^{-5}, 10^{-1}], power t: [logit, 0.1, 0.9], momentum: [logit, 0.001, 0.999] }
- **AdaBoost**: {n estimators: [linear, 10, 100], learning rate: [log, 10^{-4}, 10^{1}] }

Findings. On the iteration axis (Fig. 12b), ASAP retains its decision-quality advantage: at trial 50 ASAP's normalized regret is ≈ 0.09, ahead of LLAMBO (≈ 0.13) and the best BO (SKOPT ≈ 0.12); the gap to all other BOs is even larger. The three frontier LLM agents (AgentHPO protocol with Claude Fable 5, Claude Opus 5, and GPT-5.6 Sol; one seed, 5 init + 50 trials) drop quickly in the first ten trials and then stall at ≈ 0.19, ≈ 0.19, and ≈ 0.23, behind LLAMBO and every BO tool except GP and DNGO: on these small sklearn spaces the models' priors are exhausted within a handful of proposals and, with no surrogate to refine them, later proposals barely improve. ASAP's pool-plus-Judge selection remains the most sample-efficient on a per-iteration basis.

On the wall-clock axis with a fair 7 min budget (Fig. 12a), the picture changes. Trivial-eval lets model-free BOs blast through hundreds of trials per minute, and the strongest of them (TPE) actually *beats* ASAP: TPE plateaus at ≈ 0.05 regret within $\sim$1 min and stays there, while ASAP needs $\sim$5 min before crossing under the BO plateau and ends at ≈ 0.075. Optuna and SKOPT (≈ 0.085) lead ASAP for most of the 7 min window, though ASAP edges under them by the end. ASAP remains ahead of LLAMBO (≈ 0.11), TuRBO (≈ 0.11), SMAC (≈ 0.09), GP (≈ 0.18), and DNGO (≈ 0.33), and well ahead of the frontier LLM agents, which plateau at ≈ 0.19–0.23 within the first minute and never recover. This is exactly the boundary result we expect: when per-trial wall is dominated by the LLM stack and the ML fit is essentially free, a model-free BO that can issue $5\times$ more trials in the same wall window wins. The ranking inversion confirms that ASAP's design assumption, a non-trivial T_{ML} to hide overhead under, is load-bearing. In the main paper's non-trivial-eval regime ($T_{\mathrm{ML}} \geq 5$ s) the same TPE that wins here is among the worst baselines, illustrating that "best HPO algorithm" is regime-dependent and that the trivial-eval regime is not the one practitioners actually pay for in real model training.



(a) Wall-clock time in minutes **(b)** Iteration

Figure 12: Trivial-eval (sub-second sklearn HPO) on the full Bayesmark grid, in the same style as Fig. 3: (a) normalized regret vs. wall-clock on the fair 7 min budget (BO: 5 init + 500 trials, 3 seeds; LLM methods: 5 init + 50 trials, LLAMBO 3 seeds, ASAP 2–3 seeds; frontier LLM agents 1 seed), with each method's regret at 7 min labelled at the right edge, and (b) regret vs. iteration (50 trials); in this regime the two axes are reported on equal footing. Curves average 39 of the 40 combos (Griewank/SVM excluded, see text). **ASAP (red, ours)** retains its per-iteration advantage but loses the wall-clock axis to model-free BOs that fit hundreds of trials inside its 7 min LLM-bound window: TPE reaches 0.051 while ASAP ends at 0.072, still ahead of LLAMBO (0.107) and most BOs. The three frontier LLM agents (blues) pay the same LLM latency without any surrogate and plateau at 0.19–0.23 on both axes, the weakest LLM-based methods here. This is the boundary at which the speculation-driven wall-clock story breaks down, exactly outside the regime the main paper targets.

C.7 Optimization-Landscape Features and Per-Task Classification

This section documents the flacco-style (Kerschke, 2017) landscape features that drive Fig. 6 in the main paper. We compute every feature *from existing logs* (no extra evaluations) by pooling the (config, value) pairs that every method already produced on each task. Concretely, for a single (task, model) combo T we collect all $35 \times 2 \times 9 = 630$ evaluated points across the 7 BO baselines, LLAMBO, and ASAP (Ours) (2 seeds each), normalize each configuration $x \in \mathbb{R}^4$ to $[0,1]^4$ using the search-space bounds in Sec. B.2 (with the correct linear / log / logit warping per hyperparameter), and treat the resulting (X, y) as a sample of the optimization landscape on which we evaluate the seven scalar features below. Aliases: $N_{\text{pts}} = |y|$, $\bar{x}_i$ standardized x_i.

Smooth or Rugged. **(1) Dispersion ratio** `disp_ratio_05` is

$$\texttt{disp_ratio_05} = \text{disp}(X_{\text{top }5\%}) \,/\, \text{disp}(X),$$

where $\text{disp}(\cdot)$ is the mean pairwise Euclidean distance among the listed points and $X_{\text{top }5\%}$ are the top 5% by y. Smaller values indicate the high-fitness region is concentrated (smooth), larger values indicate the top points are spread over multiple basins (rugged). **(2) Information content** `ic_hmax` is the Shannon entropy of bigrams of $\text{sign}(\Delta y_t)$ along a greedy nearest-neighbor walk through X, normalized to $[0,1]$. Larger $\Rightarrow$ value sign flips more often along the walk $\Rightarrow$ rugged.

Multi-Modal. **(3) Number of peaks** `n_peaks` is the number of local maxima of a smoothed kernel-density estimate of the value histogram (proximity-pruned with prominence $\geq 0.05\,\hat{f}_{\max}$). More peaks $\Rightarrow$ more modes. **(4) Kurtosis** of y (Pearson, not excess). High kurtosis indicates that the value distribution has heavy outlier modes alongside a dominant basin.

Deceptive. **(5) Fitness-distance correlation** `fdc` is

$$\texttt{fdc} = |\rho_{\text{Spearman}}(d_i, -y_i)|,$$

where $d_i = \|x_i - x^*\|_2$ and $x^* = \arg\max y$. Smaller values mean the value gives no signal about distance to the best observed point (deceptive); larger values mean the response monotonically guides search toward the optimum.

Anisotropic. **(6)** `coef_max_by_min` is

$$\texttt{coef_max_by_min} = \max_j |\beta_j| \,/\, \min_j |\beta_j|$$

from an OLS fit of $\bar{y}$ on $\bar{X}$. Larger $\Rightarrow$ the response depends much more strongly on some hyperparameters than others (anisotropic sensitivity). **(7)** `quad_cond` is the condition number of the 4×4 Hessian H of the simple quadratic fit

$$\bar{y} = a + b^\top \bar{x} + \tfrac{1}{2}\, \bar{x}^\top H \bar{x}$$

(Hessian symmetrized; absolute eigenvalues; condition number $|\lambda|_{\max}/|\lambda|_{\min}$). Larger $\Rightarrow$ axis-aligned curvatures differ more strongly $\Rightarrow$ basin is more elongated.

Classification into the four landscape axes. For each feature we z-score across the 28 tasks (after a $\log_{10}$ transform for the two heavy-tailed anisotropy features). The four landscape axes used in Fig. 6 are then the simple combinations

$$\begin{aligned} \text{Rugged} &= \tfrac{1}{2}(z_{\text{ic_hmax}} + z_{\text{disp_ratio_05}}), \\ \text{Multi-Modal} &= \tfrac{1}{2}(z_{\text{n_peaks}} + z_{\text{kurtosis}}), \\ \text{Deceptive} &= -z_{\text{fdc}}, \\ \text{Anisotropic} &= \tfrac{1}{2}(z_{\text{coef_max_by_min}} + z_{\text{quad_cond}}). \end{aligned} \tag{13}$$

For each axis we keep the 14 tasks (top 50%) whose axis score is largest (the "strongly expressing" subset), and report each method's mean normalized regret at iteration 35 averaged over those tasks (and 2 seeds each); the single high-dimensional outlier (OpenML 168908 / XGBoost, 1646 features), on which every method shows extreme variance, is kept in the classification but dropped from the averages, leaving 13 tasks on the Rugged, Multi-Modal, and Deceptive axes. The same per-task regrets give the tool-selection view reported in Sec. 5.2: per axis, the per-task best BO tool, the ordering of fixed tools by axis mean, and the per-task oracle that picks the best BO tool with hindsight (all written to `exp_landscape_radar.json`). In Fig. 6 the radial axis is inverted so that 0 regret sits at the outer ring and larger regret pulls lines toward the centre ("outer = better"). Membership is non-exclusive: tasks can express multiple axes.

Per-task feature values and membership. Table 4 lists the seven feature values for every task in our paper scope and marks which of the four landscape axes that task belongs to. The right-hand four columns make the classification reproducible end-to-end from this table alone.

D LIMITATIONS

ASAP has the following limitations. First, the speculation gain depends on the ratio of evaluation cost to overhead: it is largest when the two are comparable, and small at either extreme, when T_{ML} is too cheap (below roughly 5 s) to hide most of the propose+Judge overhead, or so long (multi-hour training) that the hidden overhead is a negligible share; integration still delivers the regret gain in both cases. We also fix the accept threshold τ rather than adapting it; an adaptive gate would need to know whether an accepted misprediction changed the decision, which is not observable online without re-running the tools and Judge, so a decision-level gate is left to future work. Second, the KV-cache-aware prompt assumes the serving engine reuses the KV cache of token-identical prefixes; where such reuse is unsupported, this benefit degrades. Third, our wall-clock figures are controlled estimates rather than repeated end-to-end measurements of the asynchronous system: PD1 adds replayed training time to measured overhead and the comparison of ASAP with and without speculation composes per-round times from the logs, so realistic queueing and API-latency variance are not captured.

Task / Model	disp_05	ic_hmax	n_pk	kurt	fdc	$\texttt{coef}_{\max/\min}$	quad.cond	R	M	D	A
adult/RF	0.71	0.96	1	7.37	0.57	34.0	7.80	✓	✓		
adult/xgboost	1.03	0.95	1	20.64	0.07	11.0	33.7	✓	✓	✓	✓
high-dim/RF	0.51	0.88	1	9.55	0.55	11.6	21.2	✓	✓		✓
high-dim/xgboost	0.79	0.87	1	5.14	0.34	6.71	18.4	✓	✓	✓	
jasmine/RF	0.66	0.98	1	12.12	0.31	138	465	✓	✓	✓	✓
jasmine/xgboost	0.79	0.97	2	7.88	0.46	16.5	10.3	✓	✓		
madeline/RF	0.87	0.88	1	3.12	0.17	4.03	13.9	✓		✓	
madeline/xgboost	1.05	0.88	2	2.83	0.22	2.62	5.80	✓	✓	✓	
volkert/RF	0.26	0.91	2	7.07	0.82	21.1	10.9	✓	✓		
cifar10/wide_resnet/2048	0.33	0.67	2	4.20	0.31	34.5	51.8		✓	✓	✓
cifar10/wide_resnet/256	0.43	0.49	0	7.43	0.31	130	12.3			✓	✓
cifar100/wide_resnet/2048	0.41	0.68	1	2.69	0.37	22.3	10.7			✓	
cifar100/wide_resnet/256	0.38	0.61	0	3.92	0.74	18.2	15.6				✓
fashion_mnist/max_pooling_cnn/2048	0.30	0.65	1	2.62	0.41	70.8	167				✓
fashion_mnist/max_pooling_cnn/256	0.25	0.58	0	4.20	0.35	9.23	24.2			✓	
fashion_mnist/simple_cnn/2048	1.01	0.61	2	3.48	0.33	29.3	116	✓	✓	✓	✓
fashion_mnist/simple_cnn/256	1.12	0.54	0	4.75	0.20	126	34.9	✓		✓	✓
imagenet/resnet/256	0.28	0.67	0	3.41	0.46	11.0	7.37				
imagenet/resnet/512	0.56	0.58	0	2.90	0.44	27.2	9.68				✓
lm1b/transformer/2048	0.63	0.64	1	5.36	0.63	6.97	8.60		✓		
mnist/max_pooling_cnn/2048	0.62	0.57	1	3.29	0.09	2.01	56.2			✓	
mnist/max_pooling_cnn/256	0.71	0.39	0	4.67	0.13	48.5	241			✓	✓
mnist/simple_cnn/2048	0.56	0.56	0	3.08	0.42	672	54.3				✓
mnist/simple_cnn/256	0.99	0.42	0	4.66	0.14	32.2	96.3	✓		✓	✓
svhn_no_extra/wide_resnet/1024	0.20	0.49	1	4.89	0.68	4.97	21.4		✓		
svhn_no_extra/wide_resnet/256	0.15	0.40	0	7.03	0.70	71.4	51.5				✓
translate_wmt/xformer_translate/64	0.79	0.62	2	3.48	0.71	22.8	5.94	✓	✓		
uniref50/transformer/128	0.79	0.66	2	4.78	0.56	7.47	10.7	✓	✓		

Table 4: Per-task flacco-style landscape features and the resulting membership in the four landscape axes used by Fig. 6. Higher `disp_05`, `ic_hmax` indicate a more rugged landscape; higher `n_pk`, `kurt` indicate multi-modality; lower `fdc` indicates a more deceptive landscape; larger $\texttt{coef}_{\max/\min}$ and `quad.cond` indicate stronger anisotropy. The last four columns (**R**ugged, **M**ulti-Modal, **D**eceptive, **A**nisotropic) mark whether the task lies in the top 50% of all 28 tasks by that combined landscape axis; membership is non-exclusive.