# Improving LLM-based Global Optimization with Search Space Partitioning

**Andrej Schwanke**[*1] **Lyubomir Ivanov**[*1] **David Salinas**[1,2]
**Fabio Ferreira**[1] **Aaron Klein**[4] **Frank Hutter**[3,2,1] **Arber Zela**[*1]
[1]University of Freiburg, [2]ELLIS Institute Tübingen, [3]Prior Labs, [4]ScaDS.AI, University of Leipzig

## Abstract

Large Language Models (LLMs) have recently emerged as effective surrogate models and candidate generators within global optimization frameworks for expensive blackbox functions. Despite promising results, LLM-based methods often struggle in high-dimensional search spaces or when lacking domain-specific priors, leading to sparse or uninformative suggestions. To overcome these limitations, we propose HOLLM, a novel global optimization algorithm that enhances LLM-driven sampling by partitioning the search space into promising subregions. Each subregion acts as a "meta-arm" selected via a bandit-inspired scoring mechanism that effectively balances exploration and exploitation. Within each selected subregion, an LLM then proposes high-quality candidate points, without any explicit domain knowledge. Empirical evaluation on standard optimization benchmarks shows that HOLLM consistently matches or surpasses leading Bayesian optimization and trust-region methods, while substantially outperforming global LLM-based sampling strategies.

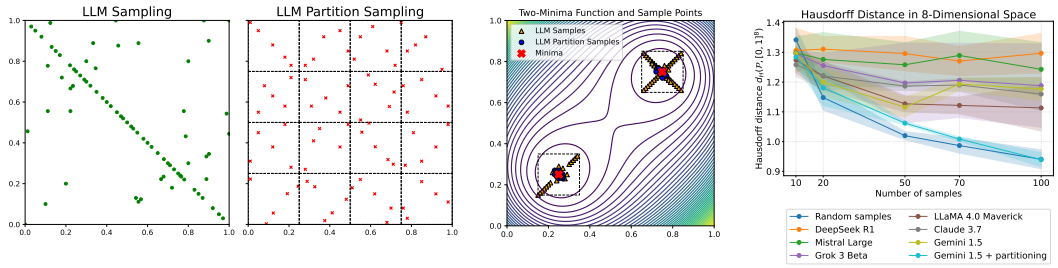
## 1 Introduction and Motivation

Global optimization [26, 44] (also known as gradient-free or zeroth-order optimization) of blackbox functions, where the only information provided to the optimizer is the function value, is a fundamental challenge across numerous domains including hyperparameter tuning [50, 54], policy search [12], molecular design and chemical engineering [33, 25], just to name a few. Methods such as Bayesian optimization [47, 21] and evolutionary algorithms [24] have been a standard and effective choice across various applications. However, they typically require assumptions regarding the underlying objective function's nature, which consecutively affect algorithmic design choices.

At the same time, recent advances in Large Language Models (LLMs) have demonstrated remarkable capabilities in generative modelling and reasoning [9, 39, 53], suggesting their potential usage for optimization tasks as well [51]. Efforts in integrating LLMs within blackbox optimization algorithms as surrogate models or as candidate samplers have already shown encouraging results [60, 34, 64, 2, 1, 32]. Yet these methods typically rely on carefully engineered, domain-specific prompts, and in higher dimensions and complex search spaces the LLM's suggestions tend to scatter sparsely, covering only a fraction of the domain [31].

As a motivating example, we investigated the capabilities of LLMs to simulate uniform sampling from a unit hypercube. In Figure 1a we show 80 samples drawn from the unit square $[0, 1]^2$, comparing uniform sampling (blue) with Gemini-1.5's [43] attempt at simulating uniform sampling using the prompt provided in Listing 1 in Appendix D (green points), and Gemini-1.5 performing uniform sampling with 5 samples per smaller subregion, using the same prompt (red points). We can clearly notice that even in 2D the LLM demonstrates high bias when sampling, therefore failing to appropriately fill the space as it was tasked to, whilst partitioning the space and prompting the LLM

[*]Equal contribution. Email to: {schwankea, ivanovl, zelaa}@cs.uni-freiburg.de




(a) (b) (c)

Figure 1: (*a*) 80 samples in $[0, 1]^2$: Gemini-1.5 simulating uniform sampling (green), and with region-wise partitioning (red) using the prompt in Listing 1. (*b*) Gemini-1.5 prompted (see Listing 2) to generate 80 samples around the 2 minima (red crosses) globally (triangles) and withing the two bounding boxes (circles). (*c*) Hausdorff distance $d_H(\mathcal{P}, [0, 1]^8)$ for uniform vs. LLM-simulated sampling in the 8-D hypercube.

16 times yields a more faithful simulation. Another illustrative example is shown in Figure 1b, where we prompt Gemini-1.5 (using the prompt shown in 2) to sample close to the two global minima (red stars) of a quadratic function, given the input space boundaries. We can clearly notice the higher sampling bias when the input space is $[0, 1]^2$ instead of the smaller regions denoted via the dashed bounding boxes. Finally, in Figure 1c, we compute the Hausdorff distance, $d_H(\mathcal{P}, [0, 1]^8)$, between the set of $N \in \{10, 20, 50, 70, 100\}$ sampled points $\mathcal{P}$ and the 8-dimensional unit hypercube $[0, 1]^8$. The blue curve indicates the values for standard uniform sampling and the other ones performed by various non-agentic LLMs. Similarly as in the 2D case, partitioning the hypercube into 32 regions and sampling within each (Gemini 1.5 + partitioning) notably improves the spatial coverage, enabling the LLM to more closely approximate uniform sampling.

In this paper, we introduce *Hierarchical Optimization with Large Language Models (HOLLM)*, a novel blackbox optimization method that leverages adaptive spatial partitioning to guide LLM-based sampling. HOLLM iteratively builds a KD-tree on existing evaluation data, creating adaptive local partitions whose granularity evolves with sampling density. Each subregion is assigned a bandit-inspired utility score, balancing exploitation (regions with promising observed values) and exploration (geometrically large or statistically uncertain regions). Subregions are selected stochastically according to these scores, and LLMs then generate localized candidate proposals within the chosen regions. As LLMs trained on optimization literature and scientific data encode a valuable *meta-prior* about typical function behavior (e.g., local unimodality), we effectively harness this prior without assuming a fixed parametric surrogate (e.g., Gaussian Process). Furthermore, restricting candidate generation to smaller, lower-dimensional subregions significantly reduces LLM sampling difficulty compared to global high-dimensional sampling. We note that spatial partitioning heuristics have already proven effective in continuum-armed bandits [38, 10, 55, 23], Trust Region Bayesian optimization [18, 15], and Monte Carlo Tree Search [27, 56, 61]. The key contribution of this work is the integration of these partitioning ideas to substantially improve LLM-driven global optimization performance.

Empirical evaluations on continuous and discrete benchmark functions, including hyperparameter optimization and neural architecture search, demonstrate that HOLLM effectively balances exploration and exploitation, matching or outperforming state-of-the-art methods, including established Bayesian optimization variants and Trust Region algorithms, particularly in scenarios requiring efficient navigation of complex landscapes. Furthermore, compared to approaches that prompt the LLM to propose candidates globally, HOLLM achieves considerable gains by focusing LLM suggestions locally. We provide the implementation of our algorithm in the following repository: `https://github.com/automl/hollm`.

## 2 Background and Related Work

We consider the problem of maximizing a blackbox function $f : \mathcal{X} \to \mathbb{R}$ where $\mathcal{X}$ is a compact domain. The objective is to find $x^* = \arg\max_{x \in \mathcal{X}} f(x)$ through a sequence of function evaluations. In this blackbox setting, we do not have access to gradients or other properties of $f$, and can only observe function values at queried points. The performance of optimization algorithms in this context

can also be measured using *simple regret* or *cumulative regret*. For a sequence of evaluated points $x_1, x_2, \ldots, x_t$, the simple regret after $t$ iterations is defined as: $r_t = f(x^*) - \max_{i \in \{1,\ldots,t\}} f(x_i)$, while the cumulative regret is: $R_t = \sum_{i=1}^{t} (f(x^*) - f(x_i))$.

**Bayesian Optimization.** Bayesian Optimization (BO) [21, 20, 47] is a well-established framework for optimizing expensive blackbox functions by maintaining a probabilistic surrogate (typically a Gaussian Process [41]) to guide evaluations and optimizing an acquisition function (e.g. Expected Improvement [63]) in order to balance exploration and exploitation and efficiently search the space. Extensions like TuRBO [18, 15] address high-dimensional settings by maintaining multiple trust regions, which are dynamically resized based on optimization progress, enabling scalable and focused exploration around promising evaluations via local GPs.

**Multi-Armed Bandits and Hierarchical Optimization Algorithms.** Multi-Armed Bandits (MABs) [49] deal with the problem of sequential decision-making under the exploration-exploitation dilemma. In the basic setting, a MAB algorithm repeatedly selects among a fixed (also infinite) number of arms or actions, each with an unknown reward distribution, aiming to minimize the cumulative regret. In the global optimization setting, the arms are the points that lie in the input space $\mathcal{X}$ and at each iteration $t$, an arm $x_t \in \mathcal{X}$ is pulled and the regret is computed by evaluating the function $f(x_t)$ [23]. Several MAB algorithms leverage hierarchical space partitioning [29]. Most notably, HOO [10] constructs a hierarchical partitioning of the search space using $n$-ary trees and at each step, an unexplored region (tree leaves) is selected based on upper confidence bounds (UCB) [5, 30] and $f$ is evaluated at a point uniformly sampled inside the selected region. Building on HOO, extensions include parallel versions [23], optimization without explicit smoothness knowledge [38] or under noisy observations [55], and adaptive trees [11] including Monte Carlo bandits [56, 61]. Most of these methods come with theoretical guarantees on regret bounds that depend on the dimensionality and smoothness properties of the objective function.

**Large Language Models for Blackbox Optimization.** Recent work has increasingly explored integrating LLMs into blackbox optimization workflows. Some approaches prompt LLMs directly to generate candidate solutions in natural language [34, 60, 1, 64], use them to estimate uncertainty [40], extract features [32], or even design novel acquisition functions [2]. Others replace traditional surrogate models with LLMs to predict function values and guide search in applications such as hyperparameter tuning [34] and molecular design [40]. However, these methods often rely on carefully engineered prompts containing domain-specific information (e.g., dataset statistics or problem descriptions), raising concerns about their robustness in domains where this information is not available. Recent work by [31] shows that, in simple MAB settings, LLMs struggle to explore effectively without significant prompt intervention, highlighting their limitations in decision-making.

Our algorithm builds upon these foundations in order to improve LLM-based blackbox optimization by integrating tree-based space partitioning, a UCB-inspired score function for balancing exploration and exploitation, and LLM-based candidate generation within locally promising regions.

## 3 HOLLM: Hierarchical Optimization with LLMs

In this section, we present the HOLLM algorithm for optimizing potentially noisy blackbox functions $f : \mathcal{X} \to \mathbb{R}$, which consists of 5 main steps: `Partition`, `Score`, `Select`, `Sample` and `Evaluate`. Given an initial set of $n_0$ evaluations $\mathcal{D}_{n_0} = \{(x_i, f(x_i))\}_{i=1}^{n_0}$, the algorithm iteratively calls each of these steps. It starts by adaptively partitioning the search space in a data-driven way, scores each of these regions to balance exploration-exploitation, selects the $M$ most promising regions based on their score, leverages LLMs to sample candidates within these regions, and finally evaluates the best candidates according to their predicted function value from the LLM. We provide an illustrative depiction of these steps in Figure 2. This approach allows the LLM to focus on promising smaller regions of the space while benefiting from the global partitioning strategy. We provide the algorithm pseudocode in Algorithm 1 and a more detailed version in the Appendix A. In the following, we explain each step in detail.

### 3.1 `Partition`: Adaptive Discretization

Based on the motivating examples we presented in Section 1, we hypothesize that firstly identifying promising smaller regions in the input space $\mathcal{X}$ makes the LLM-based sampling more reliable

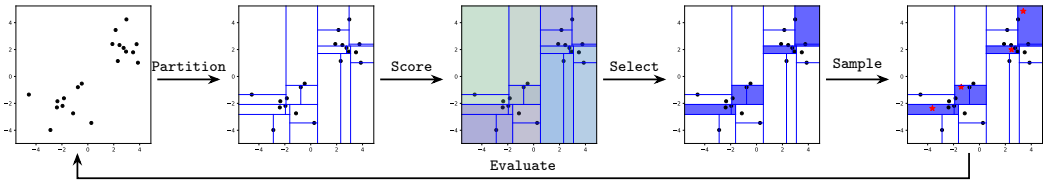



Figure 2: Overview of the HOLLM algorithm: starting from initial data $\mathcal{D}$, it iteratively performs `Partition`, `Score`, `Select`, `Sample` (via LLM), and `Evaluate` steps to balance exploration and exploitation. For the partitioning here, we utilized a KD-Tree where each axis is split based on the mean values. Each rectangle represents a partition defined by the tree leaves. The red stars represent the new sampled points from the LLM.

compared to prompting the LLM to sample globally. To this end, we propose using an adaptive input space partitioning method based on the evaluated data at each iteration of the algorithm. In order to obtain disjoint space partitions that cover the entire space, we use k-dimensional trees (KD-trees), a space-partitioning data structure that recursively divides the space into half-spaces, so that we can efficiently compute the partitions in high dimensions ($\mathcal{O}(t\log(t))$ for a balanced tree where $t$ is the number of iterations), whereas for other methods, such as a Delaunay triangulation [22] and Voronoi diagram [27, 59], this would become quickly impractical as the dimension $d$ increases. Each non-leaf node in a KD-tree represents a splitting hyperplane perpendicular to one of the coordinate axes, dividing the space into two parts. Points on the "left" side of this hyperplane are represented by the left subtree, and points on the "right" side are represented by the right subtree. Starting from the root node $\mathcal{X}_\varnothing = \mathcal{X}$, every internal node chooses a split dimension $s$ (the one with the largest variance among points in the node) and a split value $\delta$ (the mean across the selected dimension). This produces two child nodes

$$\mathcal{X}_{\text{left}} = \{x \in \mathcal{X} : x_s \leq \delta\}, \quad \mathcal{X}_{\text{right}} = \{x \in \mathcal{X} : x_s > \delta\},$$

whose union equals their parent and whose interiors are disjoint. After inserting $n$ sample points, the $K$ leaves $\{\mathcal{X}_l\}_{l=1}^{K}$ form a partition of $\mathcal{X}$ into axis-aligned hyperrectangles and contain information about the points evaluated within it, including their coordinates and function values. We denote the set of indices each leaf $\mathcal{X}_\ell$ holds as: $I_\ell = \{\, i \leq t : x_i \in \mathcal{X}_\ell\}$, with sample size $n_\ell = |I_\ell| \leq m_t$. $m_t$ is the maximum number of points a leaf in the KD-tree can keep before splitting, parameterized by the number of iterations. At the start of round $t$, we optionally set $m_t = m_0 + \lceil \lambda \log(1+t) \rceil$, where $m_0$ ($\lceil d/2 \rceil$ by default) is the initial leaf size and $\lambda$ (0 by default) is the growth parameter. The logarithmic growth of $m_t$ ensures that the partitions do not become too fine-grained quickly.

**An infinite-armed bandit view.** Conceptually, the KD-tree can be interpreted as a *data-driven discretizer* in infinite-armed bandits [57, 13]: its leaves form a coarse partition at the beginning of learning and refine only where information accumulates, mirroring the "zooming" phenomenon in continuum–armed bandits [29, 57, 38, 10, 48, 55, 11, 13, 23]. A key distinction, however, is that the entire tree is *re-fitted* at every round, adapting the partitions' boundaries based on the current data to avoid potential early convergence to local minima. This strategy is similar to [56, 61], where they observed that recomputing the partitioning every few iterations resulted in better empirical performance. Although partition boundaries may merge or shift, one can frame the procedure as operating on a *fixed, infinite* KD-tree whose internal nodes are activated and deactivated on the fly, as abstracted in adaptive-treed bandits [11, 48]. Following this paradigm, every axis-aligned hyper-rectangular partition, $\mathcal{X}_l \subset \mathcal{X}$, can be seen as a "meta-arm" [48]. The reservoir of all such boxes is uncountable, hence discovering arms, rather than merely pulling them, becomes part of the learning problem. In this language, our algorithm may be viewed as an infinite-armed bandit strategy that *(i)* repeatedly draws a batch of candidate active input space partitions by re-fitting a KD-tree to the ever-growing set of evaluations, and *(ii)* allocates pulls among those boxes according to a score function (as described below).

### 3.2 `Score`: Synthesis of Exploitation, Geometry and Uncertainty

At every iteration $t$ the KD-tree yields a finite collection of leaves (partitions) $\{\mathcal{X}_{\ell,t}\}_{\ell=1}^{K_t}$. In order to decide where to spend our limited evaluation budget, we need to rank these leaves based on a scoring function (also called utility or acquisition function) that balances exploitation of good leaves and exploration of large regions that may hold good points and that are under-sampled.

**(i) Exploitation via the empirical *maximum*.** The exploitation term should *optimistically* reflect the best empirical evidence available for each region. Classical HOO algorithms use the sample mean as a low-variance proxy for the local reward [29, 57, 10, 55]. In global optimization, however, the objective may be highly *heteroscedastic*, where one exceptionally good point inside an otherwise mediocre box can be more informative than the entire distribution. We therefore let our exploitation statistic be the largest improvement ever observed in a region $\mathcal{X}_{\ell,t}$:

$$f_{\min}(t) = \min_{i \le t} f(x_i), \quad Y_i = f(x_i) - f_{\min}(t) + \varepsilon, \quad \mu_{\ell,t} = \max_{i \in I_{\ell,t}} Y_i. \tag{1}$$

We subtract the current empirical minimum $f_{\min}(t)$ (since we are maximizing $f$) so the values become strictly non-negative and comparable across rounds [1]. Choosing a *max* rather than an average emphasizes regions that contain a good function value, a behavior also found in acquisition functions in Bayesian optimization [21] and MCTS [56, 61].

**(ii) Geometric exploration through *hypervolume*.** Let $[l_{\ell_1}, u_{\ell_1}] \times \cdots \times [l_{\ell_d}, u_{\ell_d}]$ be the axis-aligned hyperrectangle corresponding to leaf $\mathcal{X}_{\ell,t}$, where $l_{\ell_d}$ and $u_{\ell_d}$ are the low and upper axis values across dimension $d$ determined by the points in $\mathcal{X}_{\ell,t} = \{x_i \in \mathcal{X} : i \in I_{\ell,t}\}$. In order to assign a high exploration score to regions in the input space that are underexplored, we use the $d$-th root of the leaves' Euclidean volume $\mathrm{Vol}(\mathcal{X}_{\ell,t})$: $V_{\ell,t} = \left(\prod_{j=1}^{d}(u_{\ell_j} - l_{\ell_j})\right)^{1/d}$, which is equivalent to the geometric mean of the side lengths of the hyperrectangle and is less sensitive to side lengths across single dimensions compared to the cell diameter. The $d$-th root scales $\mathrm{Vol}(\mathcal{X}_{\ell,t})$ so it has the same units as a *length*. Because axis-aligned boxes shrink anisotropically as the KD-tree refines, the $d$-th root removes the strong dependence on dimension and yields comparable numbers across $d$.

**(iii) Statistical exploration via a UCB–V term.** Even a tiny region may deserve further sampling if it contains a few samples with high variance. Let $\sigma^2_{\ell,t}$ be the empirical unbiased variance of the observed function values $\{Y_i\}_{i \in I_{\ell,t}}$ within region $\mathcal{X}_{\ell,t}$ at iteration $t$, and let $n_{\ell,t} = |I_{\ell,t}|$ be the number of samples in that cell. We adopt an exploration factor inspired by UCB-V (Upper Confidence Bound with Variance estimates) type algorithms [4, 3, 57, 37], and apply it to our dynamic KD-tree partitioning, reminiscing UCB-AIR for infinite-armed bandits where the number of arms increases at each iteration [57]. More specifically, we score the region $\mathcal{X}_{\ell,t}$ with:

$$\mathcal{E}_{\ell,t} = \sqrt{\frac{2\,\sigma^2_{\ell,t}\,\max\big(0, \ln(t/(K_t n_{\ell,t}))\big)}{n_{\ell,t}}} + \frac{c \cdot \max\big(0, \ln(t/(K_t n_{\ell,t}))\big)}{n_{\ell,t}}. \tag{2}$$

Here, $K_t = |\{\mathcal{X}_{\ell,t}\}|$ is the current number of active leaves (partitions) in the KD-tree at iteration $t$, and $c$ is a positive constant [2]. The argument of the logarithm, $t/(K_t n_{\ell,t})$, compares the average number of samples per region $(t/K_t)$ to the samples $n_{\ell,t}$ in the specific region $\mathcal{X}_\ell$. This is a concentration term that focuses exploration on regions sampled less frequently than the current average. The $\max(0, \ln(\cdot))$ ensures the logarithmic term contributes non-negatively, effectively diminishing direct exploration incentive from this term for regions sampled more than average relative to $K_t$. Since the effective noise or function variability can vary significantly across regions, we scale this concentration term inside the first summand with the empirical variance $\sigma^2_{\ell,t}$ of the corresponding region. The second summand is a correction term characteristic of Bernstein-style concentration bounds [4, 36]. It helps to ensure that the exploration bonus is sufficiently large, particularly when $n_{\ell,t}$ is small or when the empirical variance $\sigma^2_{\ell,t}$ happens to be small or zero [3]. This makes the exploration strategy more robust for leaves with limited observations.

**Final composite score.** All components must live on a shared numeric scale; otherwise, whichever component happens to have the largest dynamic range would dominate the others and nullify the intended trade–off. After each rebuild, we normalize the scores to $[0, 1]$, preserving the intended relative weights even when the set of leaves changes drastically. The total score of each partition determined by the KD-tree partitioning is:

$$B_{\ell,t} \;=\; \bar{\mu}_{\ell,t} + \alpha_t \left(\beta_1 \bar{V}_{\ell,t} + \beta_2 \,\overline{\mathcal{E}}_{\ell,t}\right), \tag{3}$$

[1] The additive constant $\varepsilon$ prevents zero scores during the startup phase.

[2] $c$ is often related to the range of function values or is a tuning parameter. We set it to 1 since in the total score we weight the total exploration factor.

[3] When $n_{\ell,t} < 2$, the empirical variance $\sigma^2_{\ell,t}$ is undefined or zero. To prevent a misleadingly small exploration bonus in such highly uncertain cases, $\sigma^2_{\ell,t}$ might be initialized to a small positive default value.

**Algorithm 1:** HIERARCHICAL OPTIMIZATION WITH LLMS (HOLLM)

**Data:** Initial data $\mathcal{D}$, budget $T$, batch size $b$, regions to sample from $M$, proposals per region $k$

**while** $t \leq T$ **do**
  Update temperature $\alpha_t$ (and optionally maximum leaf size $m_t$)
  Partition space by building KD-tree on $\mathcal{D}$, obtaining $K_t$ leaves $\{\mathcal{X}_{\ell,t}\}_{\ell=1}^{K_t}$ `// Partition`
  **for** *each leaf* $\mathcal{X}_{\ell,t}$ **do**
    Compute $\mu_{\ell,t}$ (Eq. 1), $V_{\ell,t} = \mathrm{Vol}(\mathcal{X}_{\ell,t})^{1/d}$ and $\mathcal{E}_{\ell,t}$ (Eq. 2)
    Normalize and compute total score $B_{\ell,t} = \bar{\mu}_{\ell,t} + \alpha_t \left(\beta_1 \bar{V}_{\ell,t} + \beta_2 \overline{\mathcal{E}}_{\ell,t}\right)$ `// Score`
  **end**
  Select $M$ leaves by sampling with probabilities $p_{\ell,t} \propto B_{\ell,t}$ `// Select`
  Generate $k$ candidates for each chosen leaf via LLM_GENERATE$(\mathcal{D}, \mathcal{X}_{\ell,t}, k)$ `// Sample`
  Pick the top $b$ proposals by their LLM predicted scores
  Evaluate $f$ on them, add to $\mathcal{D}$, and set $t \leftarrow t + b$ `// Evaluate`
**end**
**return** *best* $(x, y) \in \mathcal{D}$

where $\bar{\mu}_{\ell,t}$, $\bar{V}_{\ell,t}, \overline{\mathcal{E}}_{\ell,t}$ are the min-max normalized scores, and $\beta_1$, $\beta_2$ are hyperparameters ($\beta_1+\beta_2 = 1$ by default) weighting the geometric versus statistical exploration. The $\alpha_t$ multiplier is a total exploration weight following an annealing schedule (cosine in our experiments). In the early phase ($\alpha_t \approx \alpha_{\max}$) the $B_{\ell,t}$ reduces to a near-uniform mixture of exploitation and the two exploratory terms. Assuming $\beta_1 = \beta_2$ and non-drastically changing regions, as $t$ grows, the influence of $\bar{V}_{\ell,t}$ decays *faster* than that of $\overline{\mathcal{E}_{\ell,t}}$ because the latter itself shrinks with $n_\ell$. Hence, geometric exploration is front-loaded, while statistical calibration persists more throughout the optimization. When $t$ is close to $T$ the rule essentially becomes a greedy maximizer of $\bar{\mu}_{\ell,t}$, which is optimal once an $\varepsilon$-accurate maximizer has already been isolated. Thus, this composite score represents the classical trade-off: *"go where I have seen something good, go where I have not looked at all, and go where my estimate is still uncertain"*.

### 3.3 `Select`: Stochastic Selection of Partitions

Once the score $B_{\ell,t}$ (3) has been computed for every leaf, the algorithm must decide *where* to spend the next evaluation budget of size $b$. The `Select` step stochastically selects partitions by sampling from a categorical distribution over leaves. At round $t$, we draw *without replacement* a batch of $M$ distinct leaves, denoted as $\mathcal{B}_t$, from this categorical distribution where the sampling probability is: $p_{\ell,t} = B_{\ell,t} / \sum_{r=1}^{K_t} B_{r,t}$, where $\ell$ is the leaf index and $1 \leq \ell \leq K_t$. Sampling stochastically instead of selecting the top-$M$ leaves means that sub-optimal leaves are sampled infinitely often [4], potentially helping to mitigate premature convergence especially in highly non-convex and multimodal functions. Each leaf has always a positive probability due to the small constant $\epsilon > 0$ we add to the exploitation term in Equation 1 and the min-max normalization in Equation 3. As $t$ grows, those exploratory components shrink and $B_{\ell,t}$ become increasingly peaked around the empirical best leaves, pushing $p_{\ell,t}$ toward a near-greedy regime. Moreover, a smooth annealing of $\alpha_t$ in Equation 3 avoids an abrupt "switch-to-greedy" policy, which may ignore late-appearing, high-value regions if they happen to be discovered just after the switch. Finally, sampling $M$ leaves without replacement diversifies evaluations by always sampling on distinct regions.

### 3.4 `Sample`: LLM-Guided Candidate Generation

After the `Select` step has identified a batch of leaves $\mathcal{B}_t = \{\mathcal{X}_{1,t}, \ldots, \mathcal{X}_{b,t}\}$, which also contain their corresponding hyperrectangular partition boundaries, HOLLM suggests new candidate points inside each chosen partition by prompting an LLM with the following logic: *"Given the history of evaluations $\mathcal{D}_t$, propose $k$ new points that are likely to reveal high values of $f$ inside $\mathcal{X}_{i,t}$."* We construct a structured prompt (see Appendix D) containing: (i) points in $\mathcal{D}_t$ as in-context examples, (ii) the numeric partition bounds $(l_{i_s}, u_{i_s})_{s=1}^{d}$ for cell $\mathcal{X}_{i,t}$, and (iii) task instructions to return new proposals and their estimated function values. Feeding this prompt to the LLM yields

$$(\hat{\mathbf{x}}_i, \hat{\mathbf{f}}_i) \;=\; \text{LLM\_GENERATE}\big(\mathcal{D}_t, (l_{i_s}, u_{i_s})_{s=1}^{d}, k\big),$$

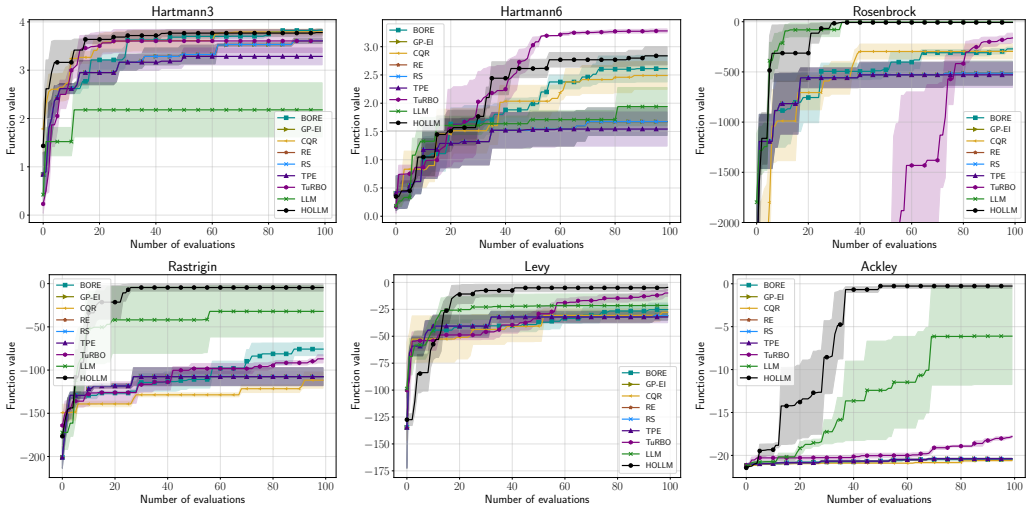



Figure 3: Best function value across 100 iterations on the synthetic problems. HOLLM outperforms or matches the performance of baselines, especially on higher dimensional problems (e.g., Ackley).

where $\hat{\mathbf{x}}_i = \{\hat{x}_{i,1}, \ldots, \hat{x}_{i,k}\} \subset \mathcal{X}_{i,t}$ and $\hat{\mathbf{f}}_i = (\hat{f}_{i,1}, \ldots, \hat{f}_{i,k}) \in \mathbb{R}^k$ are, respectively, the LLM's candidate locations and their predicted function values [4]. Across the $M$ selected leaves we thus obtain $k \cdot M$ suggestions. The parameter $k$ trades off the breadth of local exploration against prompt complexity and LLM inference cost. Finally, HOLLM keeps the globally best $b$ according to $\hat{f}$ and evaluates them on true function.

## 4 Empirical Evaluation

In this section, we evaluate HOLLM on a variety of search spaces and tasks. These span continuous synthetic functions and discrete search spaces for neural architecture search (NAS) [17, 58] and hyperparameter optimization [19, 62].

**Baselines.** On these benchmarks, we compare against different state-of-the-art algorithms from Bayesian optimization (BO), such as multi-fidelity methods (CQR [45]), Gaussian Process BO with Expected Improvement (GP-EI [50, 6]), density estimator methods (TPE [7] & BORE [52]), trust region BO (TuRBO [18]), evolutionary strategies (RE [42]) and random search (RS [8]). In all benchmarks, we also compare to the global LLM-based optimizer baseline (see Algorithm 2 in appendix) that uses the exact same prompt structure as HOLLM (we provide the prompt templates in Appendix D), with the only difference being the region boundaries.

**Setup.** Starting from $n_0 = 5$ initial random evaluations, we run each method 3 times for a total of $T = 100$ iterations with different random seeds and report their mean and standard error. We use their implementation in SyneTune [46], except TuRBO, for which we use the official BoTorch code from the authors [6]. If not stated otherwise, for HOLLM we always decay the $\alpha_t$ exploration coefficient from 1.0 to 0.01 using a cosine annealing schedule [35], a batch size $b = 4$, $M = 5$ selected partitions, $k = 5$ proposals per selected partition, and a fixed maximum leaf size $m_t = m_0 = \lceil d/2 \rceil$. In Appendix C.1, we provide ablations on these hyperparameter choices in our algorithm. We use Gemini-1.5-Flash as the LLM in `Sample` due to its fast inference speed, low cost, and large context window. Importantly, the LLM is provided with only minimal task information: the input dimensionality, variable names whenever applicable (e.g., hyperparameter names), partition boundaries, and in-context examples. No task-specific descriptions or dataset statistics are included. While prior work [34] shows that performance can improve by enriching prompts with such information, we avoid this to prevent potential contamination and reported performance on overly engineered prompts. We provide the full experimental details in Appendix B.

[4] We prompt the LLM to generate candidates and predict their performance with a single prompt or with two prompts, one for generation and one for prediction.

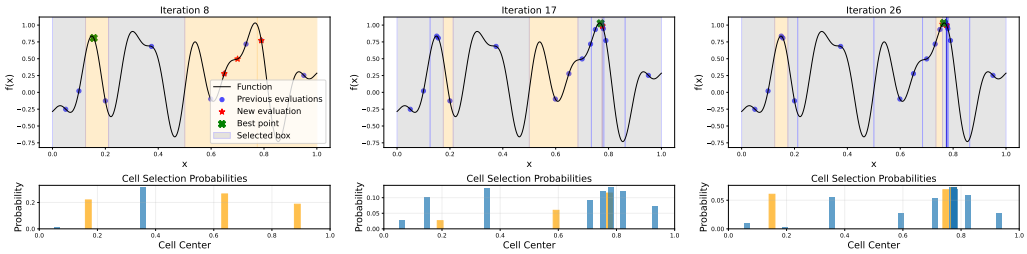



Figure 4: Illustrative example of HOLLM optimizing a 1D multimodal problem. The rectangles represent the space partitions (top figure) and are highlighted in orange whenever they are selected based on their respective probabilities (bottom figure). We used a batch size of 3. All new points (red stars) are LLM suggestions. Notice how partitions become fine-grained around the global maximum.

### 4.1 Synthetic Functions

We benchmark on six synthetic functions of varying nature and dimensionality: *Hartmann-3D* and *Hartmann-6D* (smooth but sharply multimodal), *Rosenbrock-8D* (unimodal with a narrow valley and ill-conditioning), *Rastrigin-10D* (regular multimodality with $10^{10}$ local minima), *Lévy-10D* (plateaus, cliffs, and funnels), and *Ackley-20D* (flat regions and a single sharp global minimum at the origin). These functions pose challenges ranging from fine local search to broad exploration. See Table 1 in Appendix B.2 for more details on these functions. Results presented in Figure 3 show that HOLLM consistently outperforms the global LLM baseline, especially on the multimodal functions, also exhibiting less variance between runs. It also matches or surpasses all other baselines. Most notably, on Ackley-20D with input range $[-32.768, 32.768]^{20}$, HOLLM locates the global maximum in just 50 iterations, while baselines struggle to improve beyond random search.

**Visualizing the Optimization Process.** In Figure 4, we show a visualization of HOLLM's mechanics on a 1D multimodal function. The rectangles represent the KD-tree space partitions and they are highlighted in orange whenever they get selected. We can see that during the first iterations the partitions are larger and HOLLM is more exploratory, also confirmed by the regions' respective probabilities (bottom bar plot). Later on, as the regions become smaller, high modes are identified and by the end the score probability mass concentrates more around the global maximum. We provide similar visualizations for Lévy-1D and Rosenbrock-1D in Appendix C.

### 4.2 Hyperparameter Optimization

We assess the effectiveness of HOLLM on hyperparameter optimization by optimizing the 9D categorical space from FCNet [28], where the task is to minimize the validation MSE of a fully connected network on 4 distinct datasets: `PROTEIN`, `NAVAL`, `PARKINSONS` and `SLICE`. See Appendix B.3 for more details on this search space. Results shown in Figure 5 demonstrate that our method outperforms or is on par with methods such as BORE and CQR, which typically are the off-the-shelf best choices on these benchmarks. Compared to the global LLM baseline, we can clearly see improvements on all datasets except on Parkinson, where the LLM seems to benefit more by sampling globally and reaches a low MSE after only 20 iterations. This may be due to potential outliers in the data that may impact HOLLM's performance.

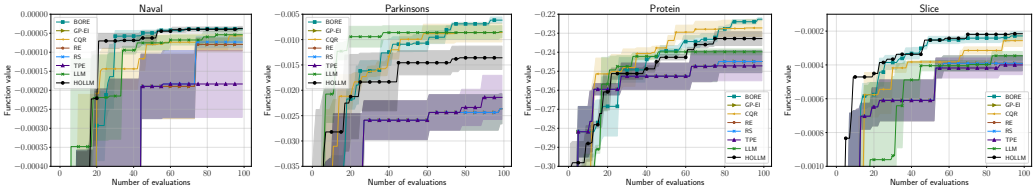



Figure 5: Hyperparameter optimization on 4 datasets from the FCNet search space. All baselines from Synetune are evaluated asynchronously using 4 workers.

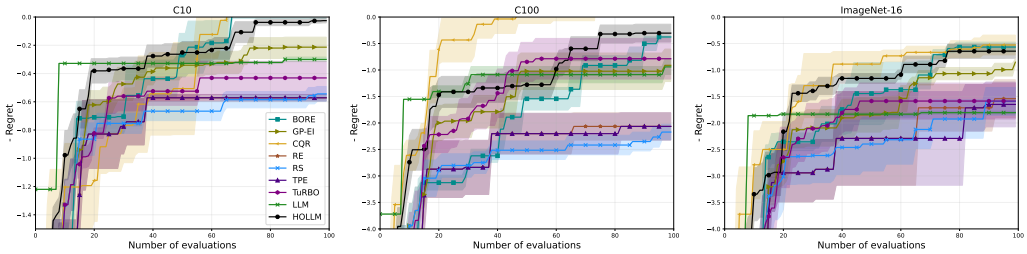



Figure 6: Results on the NAS-Bench-201 6 dimensional discrete function. We plot the negative regret vs. the number of iterations. We run each method 6 times and report the mean and standard error.

### 4.3 Neural Architecture Search

Neural Architecture Search (NAS) [58], like hyperparameter optimization, aims to identify the best-performing neural network architecture for a given dataset by maximizing validation accuracy. We use the NAS-Bench-201 benchmark [16], which provides precomputed validation accuracies for all architectures on CIFAR-10, CIFAR-100, and Downsampled ImageNet $16 \times 16$ [14]. The search space is 6D, with each dimension representing a discrete choice among 5 possible layer operations. See Appendix B.4 for full details. We use a continuous representation $[0, 1]^6$ of the input space and discretize it to evaluate the true function. As seen in Figure 6, HOLLM always outperforms the LLM baseline that samples globally and is on par with BORE and CQR. The global LLM seems to get stuck in local minima, therefore leading to stagnating performance from early on.

**Ablations.** To assess the impact of key design choices in HOLLM, we perform the following ablations: (1) we modify the score function in Equation 3 by isolating either the exploitation or exploration term; (2) we replace the variance-aware UCB-V bonus in Equation 2 with the simpler UCB1 [5]; (3) we substitute the categorical distribution used in line 8 of Algorithm 1 with a uniform distribution; and (4) we replace the LLM-based sampler in `Sample` with non-LLM baselines such as uniform sampling and samples from a local Gaussian Process fitted to each partition. Results in Figure 7 show that the choice of candidate sampler in `Sample` has the most significant effect on regret.

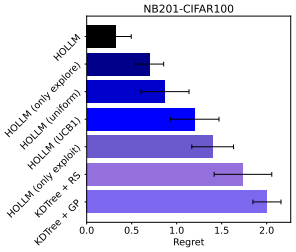



Figure 7: Ablations on HOLLM's components.

## 5 Conclusion, Limitations and Societal Impact

We propose HOLLM, a novel LLM-based global optimization method for expensive blackbox functions, that combines adaptive KD-tree partitioning, a bandit-inspired score function, and LLM capabilities for generating new candidate points on locally promising regions. HOLLM excels especially on multimodal functions with many local optima that pose a risk for premature convergence, hyperparameter optimization and neural architecture search, consistently outperforming LLMs with a global sampling policy and other non-LLM state-of-the-art methods.

**Limitations.** While HOLLM combines nonparametric bandit methods with LLM-based sampling and shows strong empirical performance, it has several limitations. First, the approach currently lacks formal theoretical guarantees, particularly regarding dynamic partitioning and regret bounds, which we leave for future work. Second, its effectiveness depends heavily on the quality of LLM-generated proposals; biased or miscalibrated models can misguide the search or waste evaluations. Third, the inference and monetary cost of LLMs, especially proprietary ones, can limit scalability in high-dimensional settings. Finally, although default parameter values perform well in our experiments, real-world deployment may require tuning them to avoid premature convergence or excessive exploration.

**Impact.** The use of LLMs in global optimization has societal implications. HOLLM has the potential to accelerate progress in areas such as drug discovery, materials design, and energy systems by reducing experimental costs and enabling personalized solutions. On the other hand, reliance on LLMs trained on biased data risks perpetuating social injustices when guiding sensitive decisions (e.g., hiring). Additionally, repeated LLM queries incur considerable energy costs, and the opacity of LLM-driven decisions may limit transparency and reproducibility. Therefore, responsible deployment requires bias assessment, usage controls, and transparency in both computational and ethical impacts.

## Acknowledgments

Robert Bosch GmbH is acknowledged for financial support. Fabio Ferreira and Frank Hutter acknowledge funding by the European Union (via ERC Consolidator Grant DeepLearning 2.0, grant no. 101045765). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Research Council. Neither the European Union nor the granting authority can be held responsible for them. Aaron Klein acknowledges the financial support by the Federal Ministry of Education and Research of Germany and by Sächsische Staatsministerium für Wissenschaft, Kultur und Tourismus in the programme Center of Excellence for AI-research „Center for Scalable Data Analytics and Artificial Intelligence Dresden/Leipzig", project identification number: ScaDS.AI. Frank Hutter acknowledges the financial support of the Hector Foundation. We also thank Google Cloud for their free trial program, that enabled us to use the Google Gemini models throughout this project.


## References

[1] Dhruv Agarwal, Manoj Ghuhan Arivazhagan, Rajarshi Das, Sandesh Swamy, Sopan Khosla, and Rashmi Gangadharaiah. Searching for optimal solutions with LLMs via bayesian optimization. In *The Thirteenth International Conference on Learning Representations*, 2025.

[2] Virginia Aglietti, Ira Ktena, Jessica Schrouff, Eleni Sgouritsa, Francisco J. R. Ruiz, Alan Malek, Alexis Bellot, and Silvia Chiappa. FunBO: Discovering acquisition functions forbayesian optimization with funsearch, 2025.

[3] Jean-Yves Audibert and Sébastien Bubeck. Minimax policies for adversarial and stochastic bandits. In *Proceedings of the 22th annual conference on learning theory*, pages 217–226, Montreal, Canada, June 2009.

[4] Jean-Yves Audibert, Rémi Munos, and Csaba Szepesvári. Exploration-exploitation tradeoff using variance estimates in multi-armed bandits. *Theor. Comput. Sci.*, 410(19):1876–1902, April 2009.

[5] Peter Auer, Nicolò Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47(2):235–256, May 2002.

[6] 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 33*, 2020.

[7] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In *Advances in Neural Information Processing Systems*, volume 24. Curran Associates, Inc., 2011.

[8] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(10):281–305, 2012.

[9] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NeurIPS '20, Red Hook, NY, USA, 2020. Curran Associates Inc.

[10] Sébastien Bubeck, Rémi Munos, Gilles Stoltz, and Csaba Szepesvári. X-armed bandits. *J. Mach. Learn. Res.*, 12(null):1655–1695, July 2011.

[11] Adam Bull. Adaptive-treed bandits. *Bernoulli*, 21, 02 2013.

[12] Roberto Calandra, André Seyfarth, Jan Peters, and Marc Peter Deisenroth. Bayesian optimization for learning gaits under uncertainty. *Annals of Mathematics and Artificial Intelligence*, 76(1–2):5–23, February 2016.

[13] Alexandra Carpentier and Michal Valko. Simple regret for infinitely many armed bandits. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, page 1133–1141. JMLR.org, 2015.

[14] Patryk Chrabaszcz, Ilya Loshchilov, and Frank Hutter. A downsampled variant of imagenet as an alternative to the cifar datasets. *CoRR*, abs/1707.08819, 2017.

[15] Samuel Daulton, David Eriksson, Maximilian Balandat, and Eytan Bakshy. Multi-objective bayesian optimization over high-dimensional search spaces. In James Cussens and Kun Zhang, editors, *Proceedings of the Thirty-Eighth Conference on Uncertainty in Artificial Intelligence*, volume 180 of *Proceedings of Machine Learning Research*, pages 507–517. PMLR, 01–05 Aug 2022.

[16] Xuanyi Dong and Yi Yang. Nas-bench-201: Extending the scope of reproducible neural architecture search. In *International Conference on Learning Representations*, 2020.

[17] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. *Journal of Machine Learning Research*, 20(55):1–21, 2019.

[18] David Eriksson, Michael Pearce, Jacob Gardner, Ryan D Turner, and Matthias Poloczek. Scalable global optimization via local bayesian optimization. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[19] Matthias Feurer and Frank Hutter. *Hyperparameter Optimization*, pages 3–33. Springer International Publishing, Cham, 2019.

[20] P. Frazier. A tutorial on bayesian optimization. *ArXiv*, abs/1807.02811, 2018.

[21] Roman Garnett. *Bayesian Optimization*. Cambridge University Press, 2023.

[22] Robert B Gramacy, Annie Sauer, and Nathan Wycoff. Triangulation candidates for bayesian optimization. In *Advances in Neural Information Processing Systems*, volume 35, pages 35933–35945. Curran Associates, Inc., 2022.

[23] Jean-Bastien Grill, Michal Valko, Remi Munos, and Remi Munos. Black-box optimization of noisy functions with unknown smoothness. In *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015.

[24] Nikolaus Hansen. The cma evolution strategy: A tutorial, 2016.

[25] José Miguel Hernández-Lobato, James Requeima, Edward O. Pyzer-Knapp, and Alán Aspuru-Guzik. Parallel and distributed thompson sampling for large-scale accelerated exploration of chemical space. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, page 1470–1479. JMLR.org, 2017.

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

[27] Beomjoon Kim, Kyungjae Lee, Sungbin Lim, Leslie Kaelbling, and Tomas Lozano-Perez. Monte carlo tree search in continuous spaces using voronoi optimistic optimization with regret bounds. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(06):9916–9924, Apr. 2020.

[28] Aaron Klein and Frank Hutter. Tabular benchmarks for joint architecture and hyperparameter optimization, 2019.

[29] Robert Kleinberg, Aleksandrs Slivkins, and Eli Upfal. Multi-armed bandits in metric spaces. In *Proceedings of the Fortieth Annual ACM Symposium on Theory of Computing*, STOC '08, page 681–690, New York, NY, USA, 2008. Association for Computing Machinery.

[30] Levente Kocsis and Csaba Szepesvari. Bandit based Monte-Carlo planning. In *European Conference on Machine Learning*, pages 282–203. Springer, 2006.

[31] Akshay Krishnamurthy, Keegan Harris, Dylan J. Foster, Cyril Zhang, and Aleksandrs Slivkins. Can large language models explore in-context? In *Advances in Neural Information Processing Systems*, volume 37, pages 120124–120158. Curran Associates, Inc., 2024.

[32] Agustinus Kristiadi, Felix Strieth-Kalthoff, Marta Skreta, Pascal Poupart, Alán Aspuru-Guzik, and Geoff Pleiss. A sober look at llms for material discovery: are they actually good for bayesian optimization over molecules? In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR, 2024.

[33] Robert Langer and David A. Tirrell. Designing materials for biology and medicine. *Nature*, 428(6982):487–492, April 2004.

[34] Tennison Liu, Nicolás Astorga, Nabeel Seedat, and Mihaela van der Schaar. Large language models to enhance bayesian optimization. In *The Twelfth International Conference on Learning Representations*, 2024.

[35] Ilya Loshchilov and Frank Hutter. SGDR: Stochastic gradient descent with warm restarts. In *International Conference on Learning Representations*, 2017.

[36] Andreas Maurer and Massimiliano Pontil. Empirical bernstein bounds and sample-variance penalization., 2009.

[37] Subhojyoti Mukherjee, K. P. Naveen, Nandan Sudarsanam, and Balaraman Ravindran. Efficient-ucbv: An almost optimal algorithm using variance estimates. *Proceedings of the AAAI Conference on Artificial Intelligence*, 32(1), Apr. 2018.

[38] Rémi Munos. Optimistic optimization of a deterministic function without the knowledge of its smoothness. In *Advances in Neural Information Processing Systems*, volume 24. Curran Associates, Inc., 2011.

[39] OpenAI. GPT-4 technical report, 2023.

[40] Mayk Ramos, Shane Michtavy, Marc Porosoff, and Andrew White. Bayesian optimization of catalysts with in-context learning, 04 2023.

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

[42] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V. Le. Regularized evolution for image classifier architecture search. In *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence and Thirty-First Innovative Applications of Artificial Intelligence Conference and Ninth AAAI Symposium on Educational Advances in Artificial Intelligence*, AAAI'19/IAAI'19/EAAI'19. AAAI Press, 2019.

[43] Machel Reid, Nikolay Savinov, Denis Teplyashin, Dmitry Lepikhin, Timothy P. Lillicrap, Jean-Baptiste Alayrac, Radu Soricut, Angeliki Lazaridou, Orhan Firat, Julian Schrittwieser, Ioannis Antonoglou, Rohan Anil, Sebastian Borgeaud, Andrew M. Dai, Katie Millican, Ethan Dyer, Mia Glaese, Thibault Sottiaux, Benjamin Lee, Fabio Viola, Malcolm Reynolds, Yuanzhong Xu, James Molloy, Jilin Chen, Michael Isard, Paul Barham, Tom Hennigan, Ross McIlroy, Melvin Johnson, Johan Schalkwyk, Eli Collins, Eliza Rutherford, Erica Moreira, Kareem Ayoub, Megha Goel, Clemens Meyer, Gregory Thornton, Zhen Yang, Henryk Michalewski, Zaheer Abbas, Nathan Schucher, Ankesh Anand, Richard Ives, James Keeling, Karel Lenc, Salem Haykal, Siamak Shakeri, Pranav Shyam, Aakanksha Chowdhery, Roman Ring, Stephen Spencer, Eren Sezener, and et al. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. *CoRR*, abs/2403.05530, 2024.

[44] Luis Miguel Rios and Nikolaos V. Sahinidis. Derivative-free optimization: a review of algorithms and comparison of software implementations. *J. Glob. Optim.*, 56(3):1247–1293, 2013.

[45] David Salinas, Jacek Golebiowski, Aaron Klein, Matthias Seeger, and Cedric Archambeau. Optimizing hyperparameters with conformal quantile regression. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett, editors, *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 29876–29893. PMLR, 23–29 Jul 2023.

[46] David Salinas, Matthias Seeger, Aaron Klein, Valerio Perrone, Martin Wistuba, and Cedric Archambeau. Syne tune: A library for large scale hyperparameter tuning and reproducible research. In Isabelle Guyon, Marius Lindauer, Mihaela van der Schaar, Frank Hutter, and Roman Garnett, editors, *Proceedings of the First International Conference on Automated Machine Learning*, volume 188 of *Proceedings of Machine Learning Research*, pages 16/1–23. PMLR, 25–27 Jul 2022.

[47] Bobak Shahriari, Kevin Swersky, Ziyun Wang, Ryan P. Adams, and Nando de Freitas. Taking the human out of the loop: A review of bayesian optimization. *Proceedings of the IEEE*, 104:148–175, 2016.

[48] Aleksandrs Slivkins. Multi-armed bandits on implicit metric spaces. In *Advances in Neural Information Processing Systems*, volume 24. Curran Associates, Inc., 2011.

[49] Aleksandrs Slivkins. Introduction to multi-armed bandits. *Found. Trends Mach. Learn.*, 12(1–2):1–286, November 2019.

[50] Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. Practical bayesian optimization of machine learning algorithms. In *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*, NeurIPS'12, page 2951–2959, Red Hook, NY, USA, 2012. Curran Associates Inc.

[51] Xingyou Song, Yingtao Tian, Robert Tjarko Lange, Chansoo Lee, Yujin Tang, and Yutian Chen. Position: leverage foundational models for black-box optimization. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org, 2024.

[52] Louis Tiao, Aaron Klein, Cédric Archambeau, Edwin V Bonilla, Matthias Seeger, and Fabio Ramos. Bayesian Optimization by Density-Ratio Estimation. In *Proceedings of the 38th International Conference on Machine Learning (ICML2021)*, Virtual (Online), July 2021.

[53] Hugo Touvron, Louis Martin, Kevin R. Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, D. Bikel, Lukas Blecher, Cristian Cantón Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, A. Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel M. Kloumann, A. Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, R. Subramanian, Xia Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zhengxu Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. Llama 2: Open foundation and fine-tuned chat models, 7 2023.

[54] Ryan Turner, David Eriksson, Michael McCourt, Juha Kiili, Eero Laaksonen, Zhen Xu, and Isabelle Guyon. Bayesian optimization is superior to random search for machine learning hyperparameter tuning: Analysis of the black-box optimization challenge 2020. In *Proceedings of the NeurIPS 2020 Competition and Demonstration Track*, volume 133 of *Proceedings of Machine Learning Research*, pages 3–26. PMLR, 06–12 Dec 2021.

[55] Michal Valko, Alexandra Carpentier, and Rémi Munos. Stochastic simultaneous optimistic optimization. In Sanjoy Dasgupta and David McAllester, editors, *Proceedings of the 30th International Conference on Machine Learning*, Proceedings of Machine Learning Research, pages 19–27, Atlanta, Georgia, USA, 17–19 Jun 2013. PMLR.

[56] Linnan Wang, Rodrigo Fonseca, and Yuandong Tian. Learning search space partition for black-box optimization using monte carlo tree search. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NeurIPS '20, Red Hook, NY, USA, 2020. Curran Associates Inc.

[57] Yizao Wang, Jean-Yves Audibert, and Rémi Munos. Algorithms for infinitely many-armed bandits. In *Proceedings of the 22nd International Conference on Neural Information Processing Systems*, NeurIPS'08, page 1729–1736, Red Hook, NY, USA, 2008. Curran Associates Inc.

[58] Colin White, Mahmoud Safari, Rhea Sanjay Sukthanker, Binxin Ru, Thomas Elsken, Arber Zela, Debadeepta Dey, and Frank Hutter. Neural architecture search: Insights from 1000 papers. *ArXiv*, abs/2301.08727, 2023.

[59] Nathan Wycoff, John W. Smith, Annie S. Booth, and Robert B. Gramacy. Voronoi candidates for bayesian optimization. *ArXiv*, abs/2402.04922, 2024.

[60] Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V Le, Denny Zhou, and Xinyun Chen. Large language models as optimizers. In *The Twelfth International Conference on Learning Representations*, 2024.

[61] Kevin Yang, Tianjun Zhang, Chris Cummins, Brandon Cui, Benoit Steiner, Linnan Wang, Joseph E Gonzalez, Dan Klein, and Yuandong Tian. Learning space partitions for path planning. In *Advances in Neural Information Processing Systems*, volume 34, pages 378–391. Curran Associates, Inc., 2021.

[62] Tong Yu and Hong Zhu. Hyper-parameter optimization: A review of algorithms and applications. *arXiv preprint arXiv:2003.05689*, 2020.

[63] Dawei Zhan and Huanlai Xing. Expected improvement for expensive optimization: a review. *J. of Global Optimization*, 78(3):507–544, November 2020.

[64] Michael Zhang, Nishkrit Desai, Juhan Bae, Jonathan Lorraine, and Jimmy Ba. Using large language models for hyperparameter optimization. In *NeurIPS 2023 Foundation Models for Decision Making Workshop*, 2023.

# A Algorithm Pseudocodes

In this section, we present detailed pseudocode for the HOLLM algorithm in Algorithm 3, which complements Algorithm 1. In the algorithm, we omit the subscript $t$ for easier readability. Additionally, Algorithm 2 describes the LLM-based global optimization baseline method used throughout our experiments. To ensure fair comparison, we configure the baseline to propose $k \cdot b$ points per iteration (line 2 of Algorithm 2), matching the total number of proposals generated by HOLLM across all subregions, i.e., $k$ proposals in each of the $M$ subregions.

**Algorithm 2:** GLOBAL-LLM baseline

**Data:** Initialize $\mathcal{D}$ with $n_0$ points, budget $T$, batch size $b$, proposals $k \cdot M$

1. **for** $t = n_0, \ldots, T-1$ **do**
2. Propose $k \cdot M$ points with LLM_GENERATE$(\mathcal{X}, \mathcal{D}, k \cdot M)$
3. Evaluate the top $b$ and add to $\mathcal{D}$
4. **return** *best* $(x, y) \in \mathcal{D}$

# B Details on Tasks, Baselines and Experimental Setup

## B.1 Baselines

We compare HOLLM to the following baselines:

- **Random Search (RS) [8]** serves as a simple baseline that uniformly samples configurations from the search space without any learning or adaptation.
- **Regularized Evolution (RE) [42]** is an evolutionary algorithm that maintains a population of candidate solutions and evolves them through mutation operations. The method regularizes the population by removing the oldest individuals, preventing stagnation and maintaining diversity.
- **Conformalized Quantile Regression (CQR) [45]** uses gradient boosted trees to predict performance quantiles and provides prediction intervals with statistical guarantees through conformal prediction techniques.
- **Tree-structured Parzen Estimator (TPE) [7]** is a sequential model-based optimization method that models the distribution of good and bad configurations separately. It builds two probability densities: $\ell(x)$ for configurations with performance better than a threshold, and $g(x)$ for those below the threshold. New candidates are selected by maximizing the ratio $\ell(x)/g(x)$.
- **Bayesian Optimization by Density-Ratio Estimation (BORE) [52]** reformulates Bayesian optimization as a binary classification problem. It trains a probabilistic classifier to distinguish between high-performing and low-performing configurations, then uses the predicted class probabilities to construct an acquisition function equivalent to expected improvement.
- **Gaussian Process BO with Expected Improvement (GP-EI) [50, 6]** employs a Gaussian Process as a surrogate model to capture the objective function's behavior and uncertainty. It uses the Expected Improvement acquisition function, implemented via BoTorch, to balance exploration and exploitation when selecting new evaluation points.
- **Trust Region Bayesian Optimization (TuRBO) [18]** addresses the curse of dimensionality in high-dimensional optimization by maintaining multiple local trust regions. Each region uses an independent Gaussian Process and adapts its size based on optimization progress, allowing the method to scale effectively to high-dimensional problems.

## B.2 Synthetic Benchmarks

In Section 4, we initially evaluated HOLLM and the baselines on 6 synthetic deterministic function with varying dimensionality and nature. In Table 1, we provide details on each of them.

**Algorithm 3:** HIERARCHICAL OPTIMIZATION WITH LLMS (HOLLM) – DETAILED

**Data:** Initial data $\mathcal{D} = \{(x_i, f(x_i))\}_{i=1}^{n_0}$, batch size $b$, regions to sample from $M$, proposal count per leaf $k$, dimension $d$, initial leaf size $m_0$, adaptive growth rate $\lambda$, total evaluations $T$, exploration weights $\beta_1$, $\beta_2$, annealing $\alpha_{min}$, $\alpha_{max}$

$t \leftarrow n_0$ `// global evaluation counter`
**while** $t < T$ **do**
  (optional) $m_{\text{leaf}} \leftarrow m_0 + \lceil \lambda \log(1+t) \rceil$ `// adaptive leaf size`
  Build KD-tree on $\mathcal{D}$ with leafsize $m_{\text{leaf}}$, obtaining $K_t$ leaves $\{\mathcal{X}_\ell\}_{\ell=1}^{K_t}$
  $\alpha \leftarrow \alpha_{min} + \frac{1}{2}(\alpha_{max} - \alpha_{min})(1 + \cos(\pi t/T))$ `// cosine annealing schedule`
  $f_{\min} \leftarrow \min_{i=1}^{t} f(x_i)$
  $Y_+ \leftarrow \{f(x_i) - f_{\min} + \epsilon\}_{i=1}^{t}$ `// positive transformed values`
  **for** $l = 1$ **to** $K_t$ **do**
    $n_\ell \leftarrow |\mathcal{X}_\ell|$ `// number of points in this leaf`
    $(l_\ell, u_\ell) \leftarrow$ bounds of cell $\mathcal{X}_\ell$
    $V_\ell \leftarrow \prod_{j=1}^{d} (u_{\ell j} - l_{\ell j})^{1/d}$ `// normalized volume`
    **if** $n_\ell > 0$ **then**
      $\mu_\ell \leftarrow \max_{i \in I_\ell} Y_+[i]$ `// best value in cell`
      **if** $n_\ell > 1$ **then**
        $\sigma_\ell^2 \leftarrow \frac{1}{n_\ell - 1} \sum_{i \in I_\ell} (Y_+[i] - \bar{Y}_\ell)^2$ `// variance in cell`
      **else**
        $\sigma_\ell^2 \leftarrow 0.01$ `// default variance for single-point cells`
      $\log\_term \leftarrow \max(0, \log(t/(K_t \cdot n_\ell)))$
      $\mathcal{E}_\ell \leftarrow (\sqrt{2\sigma_\ell^2 \cdot \log\_term / n_\ell} + \log\_term / n_\ell)$
    **else**
      $\mu_\ell \leftarrow 0$
      $expl_\ell \leftarrow 1$ `// high exploration for empty cells`
    $\bar{\mu}, \bar{V}_\ell, \bar{\mathcal{E}}_\ell \leftarrow$ min-max normalize $\mu_\ell$, $V_\ell$, $\mathcal{E}_\ell$ across all cells
    $B_\ell \leftarrow \bar{\mu} + \alpha \cdot (\beta_1 \cdot \bar{V}_\ell + \beta_2 \cdot \bar{\mathcal{E}}_\ell)$ `// composite score`
  $p_\ell = B_\ell / \sum_{r=1}^{K_t} B_r$ `// normalize score across cells`
  Sample $M$ cells $\{\mathcal{X}_{i_j}\}_{j=1}^{M} \sim \text{Categorical}(\{p_\ell\})$
  $\hat{X} \leftarrow \emptyset,\ \hat{F} \leftarrow \emptyset$
  **for** $j = 1$ **to** $M$ **do**
    $(\hat{\mathbf{x}}_j, \hat{\mathbf{f}}_j) \leftarrow \text{LLM\_GENERATE}(\mathcal{D}, (l_{i_j}, u_{i_j}), k)$
    Append $\hat{\mathbf{x}}_j$ to $\hat{X}$; append $\hat{\mathbf{f}}_j$ to $\hat{F}$
  $\pi \leftarrow \text{argsort}(\hat{F})$ `// indices of sorted values (descending)`
  $X^{\text{new}} \leftarrow$ top $b$ points from $\hat{X}$ using indices $\pi$
  **for** *each* $x \in X^{new}$ **do**
    Evaluate $y = f(x)$
    $\mathcal{D} \leftarrow \mathcal{D} \cup \{(x, y)\}$
    $t \leftarrow t + 1$
**return** *best point* $(x^*, f(x^*))$ *where* $x^* = \arg\max_{x \in \mathcal{D}} f(x)$

Table 1: List of synthetic optimization functions and their main characteristics.

| Function (dim.) | Landscape & key traits | Global boundary | Global optimum |
|---|---|---|---|
| Hartmann 3D | Smooth, strongly multimodal surface generated by four weighted Gaussians inside the unit cube; narrow, steep basins punish local search. | $(x_1, x_2, x_3) \in [0, 1]^3$ | $f_{\min} \approx -3.86278$ |
| Hartmann 6D | Six Gaussians in $[0, 1]^6$ create an even denser constellation of deceptive wells; still smooth but mildly ill-conditioned, and the search space grows exponentially. | $(x_1, x_2, \ldots, x_6) \in [0, 1]^6$ | $f_{\min} \approx -3.32237$ |
| Rosenbrock 8D | Classic curved "banana" valley stretched to eight variables; unimodal yet highly ill-conditioned, requiring precise valley-tracking; non-separable. | $(x_1, x_2, \ldots, x_8) \in [-2.048, 2.048]^8$ | $f_{\min} = 0$ |
| Rastrigin 10D | Quadratic core overlaid with cosine ripples forms a perfectly regular grid of $10^{10}$ local minima; separable but brutally multimodal, exposing algorithms prone to premature convergence. | $(x_1, x_2, \ldots, x_{10}) \in [-5.12, 5.12]^{10}$ | $f_{\min} = 0$ |
| Lévy 10D | Sine perturbations on a quadratic backbone yield wide plateaus, sudden cliffs, and deep funnels—rugged and non-separable, stressing step-size control. | $(x_1, x_2, \ldots, x_{10}) \in [-10, 10]^{10}$ | $f_{\min} = 0$ |
| Ackley 20D | Exponential of radius plus averaged cosines: vast flat outer region, encircling ridge, and a single sharp basin at the origin; tests exploration versus exploitation in very high dimension. | $(x_1, x_2, \ldots, x_{20}) \in [-32.768, 32.768]^{20}$ | $f_{\min} = 0$ |

### B.3 Hyperparameter Optimization Benchmarks

For our hyperparameter optimization experiments, we evaluate on four tasks from the FCNet benchmark [28]: `PROTEIN`, `NAVAL`, `PARKINSONS`, and `SLICE`. The FCNet benchmark provides a tabulated hyperparameter optimization setting where fully connected neural networks are trained on each dataset with different hyperparameter configurations. The search space consists of 9 categorical hyperparameters (network architecture and training parameters), yielding 62,208 possible configurations with pre-computed validation accuracies. To enable KD-tree partitioning on the categorical search space, we apply ordinal encoding to convert categorical variables into numerical split indices. Below we describe the four regression datasets used as the underlying machine learning tasks:

- `PROTEIN` is a regression dataset containing physicochemical properties of protein tertiary structures. The task involves predicting protein properties from 9 molecular descriptors across 45,730 protein samples.

- `PARKINSONS` contains biomedical voice measurements from 42 individuals with early-stage Parkinson's disease participating in a six-month telemonitoring trial. The regression target is the progression of Parkinson's symptoms, with 5,875 samples and 19 acoustic features.
- `NAVAL` consists of simulated sensor data from a naval frigate's propulsion system, including gas turbine, propeller, gearbox, and control systems. The regression task predicts component degradation levels using 11,934 samples with 16 operational features.
- `SLICE` involves predicting the relative axial location of CT scan slices within the human body. The dataset contains 384 features extracted from 53,500 CT images, describing bone structures, air inclusions, and anatomical positioning.

Table 2: Search space of the FCNet benchmark. The left column lists the hyperparameter names of the neural network that need to be tuned, whilst the right column the possible categorical choices for each hyperparameter.

| **Hyperparameter** | **Categorical Configuration Space** |
|---|---|
| Initial LR | {0.0005, 0.001, 0.005, 0.01, 0.05, 0.1} |
| Batch Size | {8, 16, 32, 64} |
| LR Schedule | {cosine, fix} |
| Activation (Layer 1) | {relu, tanh} |
| Activation (Layer 2) | {relu, tanh} |
| Layer 1 Size | {16, 32, 64, 128, 256, 512} |
| Layer 2 Size | {16, 32, 64, 128, 256, 512} |
| Dropout (Layer 1) | {0.0, 0.3, 0.6} |
| Dropout (Layer 2) | {0.0, 0.3, 0.6} |

### B.4 Neural Architecture Search Benchmarks

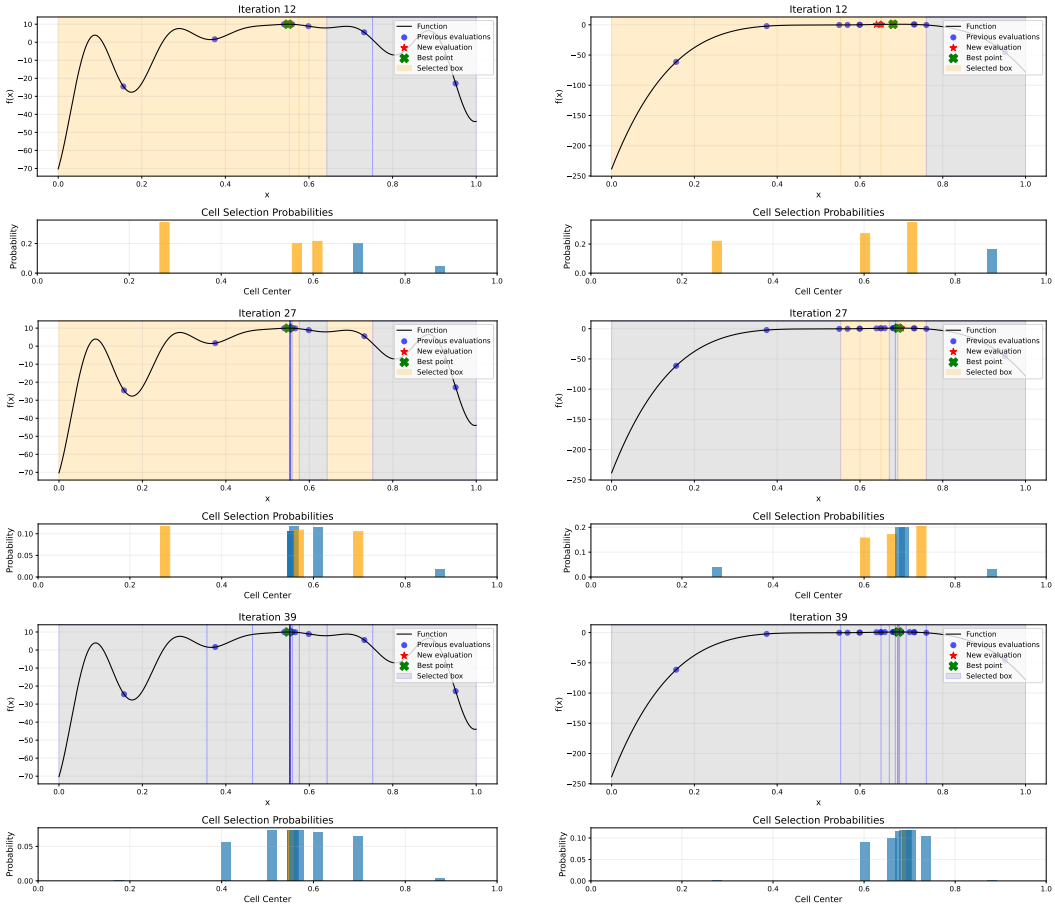
For neural architecture search (NAS), we utilize the NAS-Bench-201 [16] tabular benchmark, which provides a comprehensive evaluation suite for architecture optimization. The search space consists of selecting optimal CNN operations for each of the 6 edges in a predefined cell-based computational graph. Each edge can be assigned one of 5 categorical operations: `avg_pool_3x3` (average pooling), `nor_conv_3x3` (normal 3×3 convolution), `skip_connect` (identity connection), `nor_conv_1x1` (normal 1×1 convolution), and `none` (no operation). This yields a total search space of $5^6 = 15{,}625$ possible architectures. NAS-Bench-201 provides precomputed validation accuracies for all architectures across three image classification datasets: CIFAR-10, CIFAR-100, and ImageNet16-120 (a 16×16 downsampled version of ImageNet with 120 classes). This tabulated format enables efficient benchmarking by eliminating the computational overhead of training each architecture from scratch.

## C Additional Experiments

In this section, we provide additional experiments and ablations, complementing the ones conducted throughout Section 4 of the main paper.

### C.1 Ablations

To assess the robustness of our method and understand the influence of key hyperparameters on performance, we conducted a comprehensive ablation study. We employ the 10D Levy test function and examine 3 hyperparameters that directly impact the exploration-exploitation balance and efficacy of our approach: (i) maximum leaf capacity $m_{\text{leaf}} = m_0 + \lceil \lambda \log(1+t) \rceil$ ($\lambda = 0$), which controls the granularity of space partitioning; (ii) candidate sampling rate $k$ (proposals generated per selected region), which determines the diversity of proposals within each selected region; and (iii) region selection parameter $M$ (partitions selected per iteration), which governs the number of promising subregions explored simultaneously per iteration. The default hyperparameter configuration also used throughout the experiments in the main paper is: exploration parameter bounds $\alpha_{\text{max}} = 1.0$ and $\alpha_{\text{min}} = 0.01$, initial random sampling phase of $n_0 = 5$ evaluations, batch size $b = 4$ (points evaluated per iteration), $k = 5$, $M = 5$, and maximum leaf capacity $m_0 = d/2$, where $d$ denotes




(a) 1D Levy problem.

(b) 1D Rosenbrock problem.

Figure 8: Illustrative examples of optimizing 1D functions. The rectangles represent the input space partitions (top row). They are highlighted in orange whenever selected based on their respective probabilities (bottom row). Both experiments used 5 initial points and a batch size of 4. All new points (red stars) are suggestions from the LLM.

problem dimensionality. We run each setting with 5 independent random seeds and report the mean performance $\pm$ standard error in Figure 9.

**Impact of Leaf Size ($m_0$).** The leaf size parameter $m_0$ defines the maximum number of data points within a single leaf of the partitioning tree, directly controlling the granularity of search space decomposition. Our analysis across different values of $m_0$ as a factor of problem dimensionality $d$ reveals a clear trade-off between partition resolution and statistical reliability (Figure 9, *left*). Coarse partitioning with $m_0 = d$ yields suboptimal performance due to overly broad regions that group diverse areas of the search space, diminishing the method's ability to precisely isolate promising subregions. Conversely, extreme fine partitioning with $m_0 = 1$ also degrades performance because singleton regions provide insufficient statistical information and the variance component becomes a small constant across all regions, eliminating valuable uncertainty estimates necessary to guide exploration. We observe the best performance at $m_0 = d/4$, which strikes an effective balance by enabling detailed space partitioning while maintaining sufficient data density within each region to compute meaningful variance estimates for the exploration term.

**Impact of Number of Candidates per Region ($k$).** We investigated the effect of varying the number of candidate points $k$ sampled from each selected region, testing values $k \in \{1, 3, 5, 7, 10\}$. Results shown in Figure 9, *middle*) reveal a clear trade-off between under- and over-sampling within regions. Setting $k = 1$ leads to significant performance degradation as the method fails to adequately exploit promising regions by drawing only a single sample per region. Conversely, $k = 10$ results in

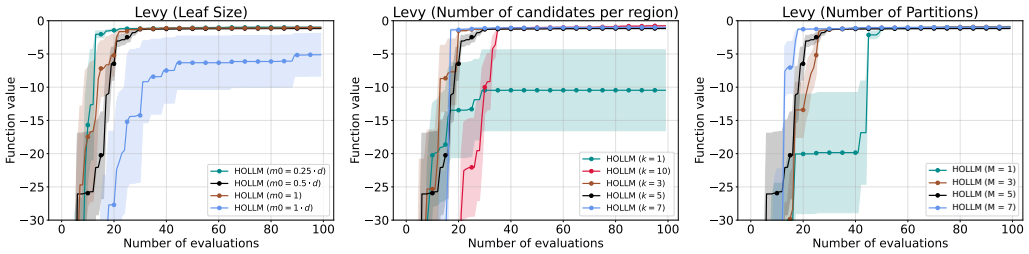



Figure 9: Impact of key hyperparameters on optimization performance for the 10D Levy function. **Left:** Ablation on leaf size ($m_0$) demonstrates that coarse partitioning ($m_0 = 1d$) yields the poorest performance, while finer partitioning enables superior exploitation of promising regions through more granular space decomposition. **Middle:** Varying the number of candidates per selected region ($k$) reveals optimal performance at intermediate values, where undersampling ($k = 1$) significantly degrades performance by limiting exploitation of high-potential regions, and oversampling ($k = 10$) slows convergence due to inefficient allocation of evaluation budget. **Right:** The number of partitions selected per trial ($M$) governs exploration breadth, where single-region focus ($M = 1$) impedes convergence through insufficient exploration, while moderate values ($M \in \{3, 5, 7\}$) accelerate optimization by enabling simultaneous exploration of multiple promising regions.

worse performance during initial iterations compared to intermediate $k$ values, which we attribute to increased risk of oversampling sub-optimal regions in the beginning. While the method can recover from this scenario as oversampled sub-optimal regions receive lower scores in subsequent iterations, the initial performance penalty demonstrates that excessive sampling can be counterproductive. These findings showcase the importance of balanced exploitation within selected regions: sufficient sampling to capitalize on promising areas without overcommitting computational budget to potentially sub-optimal regions.

**Impact of Number of Partitions Selected per Trial ($M$).** We examined how the number of partitions $M \in \{1, 3, 5, 7\}$ selected per trial affects optimization performance. As seen in Figure 9, *right*, setting $M = 1$ notably hinders performance, particularly during initial iterations, as HOLLM severely restricts exploration breadth by focusing all sampling efforts on a single region per iteration. When the initially chosen region lacks global promise, progress becomes slow, though the method can eventually exploit good regions once identified, leading to convergence that is typically slower than broader exploration strategies. Conversely, increasing $M$ to moderate or high values (3, 5, or 7) generally improves initial exploration by enabling simultaneous consideration of multiple diverse regions. This expansion of the candidate selection pool allows the algorithm to benefit from a larger, more diverse set of proposals per trial, improving performance up to a saturation point. The results demonstrate that balanced multi-region exploration through appropriate $M$ values provides superior performance compared to overly focused single-region strategies, highlighting the importance of maintaining exploration breadth while preserving the ability to exploit promising areas effectively.

### C.1.1 Impact of exploration parameter $\alpha_{max}$

We evaluated the effect of different exploration settings on the FCNet benchmarks across four tasks: `PROTEIN`, `NAVAL`, `PARKINSONS`, and `SLICE`. Results in Figure 10 show that the impact of the exploration parameter $\alpha_{\text{max}}$ exhibits task-dependent variation, with optimal settings determined by the underlying problem structure. Higher values of $\alpha_{\text{max}}$ bias the search toward less-explored regions, proving beneficial for highly multimodal or non-convex landscapes where diverse exploration is crucial for escaping local optima. Conversely, lower $\alpha_{\text{max}}$ values reduce exploration of new regions and concentrate search efforts on exploitation, which may be more appropriate for smoother or convex solution spaces where intensive local search around promising areas yields better returns. These findings suggest that prior knowledge of the task landscape characteristics, such as modality, convexity, and noise structure, can effectively guide the selection of $\alpha_{\text{max}}$ to match the exploration-exploitation balance to the problem's inherent difficulty and structure.

### C.1.2 Effect of Hyperparameters on Computational Cost

The computational complexity of our method scales directly with the total number of candidate points generated per iteration, calculated as $k \cdot M$. When employing computationally expensive models

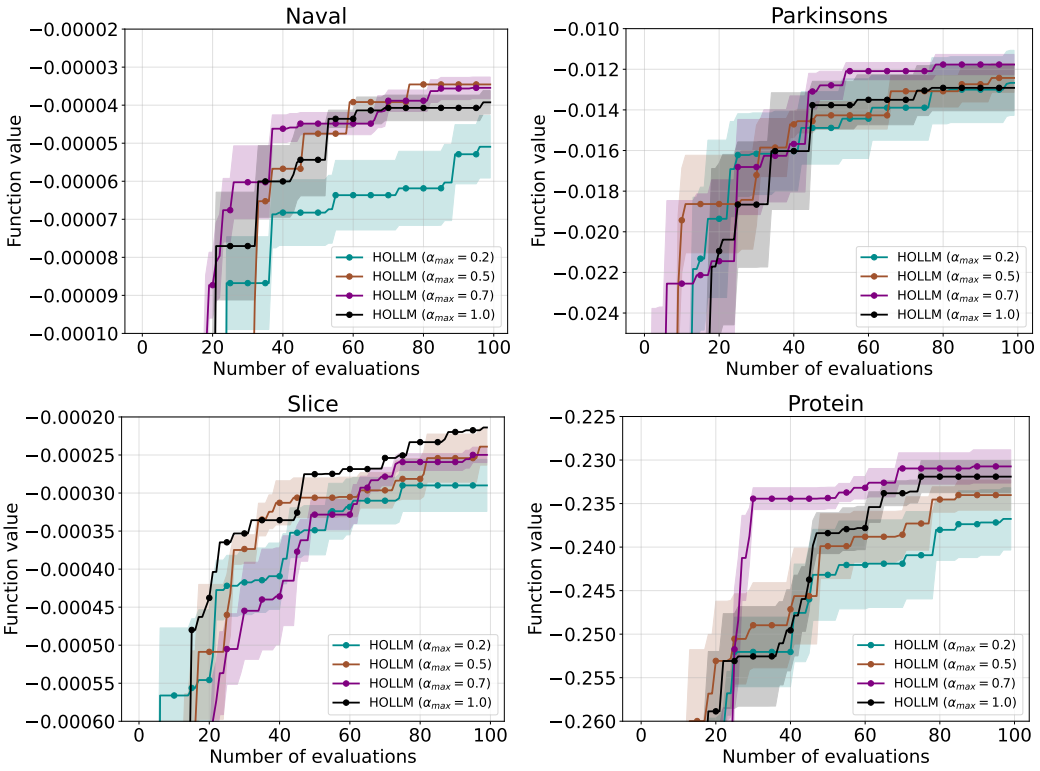



Figure 10: Performance comparison of exploration parameter settings ($\alpha_{\max} \in \{0.2, 0.5, 0.7, 1.0\}$) across the four FCNet benchmark tasks. Each curve represents the mean objective function value over 5 independent runs, with shaded regions denoting standard error. The results illustrate that optimal $\alpha_{\max}$ selection exhibits task-dependent behavior, reflecting the varying landscape characteristics and exploration requirements across different hyperparameter optimization problems.

such as large language models accessed via API calls or hosted locally, increases in either $M$ (which amplifies the number of inference calls per iteration) or $k$ (which extends the number of output tokens generated per call) result in proportionally higher inference times and associated costs per optimization step. This creates a fundamental trade-off between optimization performance and computational efficiency that practitioners must carefully consider based on their specific resource constraints and performance requirements. Our default configuration of $M = 5$ and $k = 5$, yielding 25 candidate evaluations per iteration, represents a calibrated compromise that balances exploration capability with computational practicality across the diverse benchmarks in our experimental evaluation.

# D Prompts

We design structured prompts to LLMs for both candidate generation and evaluation prediction within our optimization framework. Our prompting strategy consists of two complementary components: a candidate generation prompt (Listing 3) that produces new solutions based on historical observations, and an evaluation prediction prompt (Listing 4) that estimates the quality of proposed candidates, or both in a single prompt. Each prompt follows a systematic structure:

1. **Task specification:** We provide a description of the optimization objective and establish the problem context, including the nature of the search space and optimization goals.
2. **Dynamic constraints:** We define the feasible region constraints (boundaries) derived from the current KD tree partitioning. These constraints are automatically computed based on the selected leaf nodes and translated into natural language descriptions that specify valid input ranges alongside example-evaluation pairs.
3. **In-context examples:** We supply the model with historical observations consisting of previously evaluated points and their corresponding objective function values. These examples serve as

demonstrations to guide the model's understanding of the optimization landscape and desired output format.

4. **Task-specific instructions:** We provide explicit directives tailored to each prompt's purpose. The candidate generation prompt instructs the model to *propose new configurations*, while the evaluation prediction prompt directs it to *estimate performance* for given candidates. Both prompts enforce structured JSON output formatting for automated parsing of model responses.

Throughout our prompts, we employ placeholder variables denoted with *$* symbols to represent task-specific information that is dynamically populated during execution. Comprehensive examples and descriptions for each placeholder are provided in Table 3. For the NAS-Bench-201 experiments, we adopt a streamlined approach using a unified prompt structure (Listing 5) that simultaneously elicits both candidate proposals and their predicted evaluations in a single model query, reducing the computational overhead of separate generation and prediction phases.

Table 3: Description of placeholders for candidate proposal and prediction prompts.

| **Placeholder** | **Description** | **Example of Replaced Text** |
|---|---|---|
| `$metrics` | The performance metrics for the specific task. | `F1 (lower is better)` |
| `$region_constraints` | The allowable ranges or discrete values for the parameters in the configuration search space. | `{ lr: range(float([0.0, 0.9])), activation: choice(["relu", "tanh"]), num_layer: range(int([1, 20])) ... }` |
| `$Region_ICL_examples` | Examples of previously evaluated configurations and their performance metrics. These are the in-context learning examples. | `{ {lr: 0,4, activation: "relu", num_layer: 8,...} F1: 5.65 {lr: 0.03, activation: "tanh", num_layer: 8,...} F1: 3.23 ... }` |
| `$target_number_of_candidates` | The number of new configurations that the candidate sampler should generate. | `15` |
| `$candidate_sampler_response_format` | The required JSON structure for each new candidate configuration proposed by the sampler. | `{ lr: ?, activation: ?, num_layer: ? ... }` |
| `$target_architectures` | The set of new configurations for which the surrogate model predicts the performance metrics. | `{ 1: {lr: 0,4, activation: "relu", num_layer: 8,...} 2: {lr: 0.03, activation: "tanh", num_layer: 8,...} ... }` |
| `$surrogate_model_response_format` | The required JSON structure for the performance prediction output. | `{F1: ? }` |

```
Suggest 100 random samples for 8 dimensions within the specified
    bounding box with a maximum of 3 decimal places.

 Bounding Box:
   x1_min: 0, x1_max: 1
   x2_min: 0, x2_max: 1
   x3_min: 0, x3_max: 1
   x4_min: 0, x4_max: 1
   x5_min: 0, x5_max: 1
   x6_min: 0, x6_max: 1
   x7_min: 0, x7_max: 1
   x8_min: 0, x8_max: 1

Return the suggestions in the following JSON format exactly, without
    any additional text:
[{"x1": float, "x2": float, "x3": float, "x4": float, "x5": float, "x6
    ": float, "x7": float, "x8": float}]
```

Listing 1: Prompt for LLMs simulating 100 8-D uniform random samples.

```
Suggest 80 sample points in 2 dimensions within the specified bounding
     box.

Bounding Box:
  x1_min: 0,  x1_max: 1
  x2_min: 0,  x2_max: 1

such that they are clustered around the points given below.
Points:
  Point 1: x1: 0.25, x2: 0.25
  Point 2: x1: 0.75, x2: 0.75

Return the suggestions in the following JSON format exactly, without
    any additional text:
[{"x1": float, "x2": float}]
```

Listing 2: Prompt for LLMs sampling 80 points around the minima.

```
# Optimization task

## Problem Description
You are tasked with solving a optimization problem that requires
    finding optimal solutions.

- **Evaluation**: Configurations are measured by $metrics

## Constraints
The allowable ranges for the hyperparameters are:
$region_constraints

## Previously Evaluated Architectures
Below are examples of architectures that have been evaluated, showing
    their operations and performance metrics:

$Region_ICL_examples

## Your Task
Generate $target_number_of_candidates new architecture configurations
    that:
1. Are likely to achieve lower $metrics than the examples
2. Are different from all previously evaluated architectures
3. Satisfy all the specified constraints: $region_constraints
```

```
## Output Format
Each configuration has to follow this format:

$candidate_sampler_response_format

Provide your response in a JSON list containing each proposed
    configuration.
Return only the required JSON list output without additional text.
```

Listing 3: Prompt template used for candidate points generation in the LLM_Generate function.

```
# Configuration Performance Prediction

## Problem Description
You are tasked with predicting the performance of configurations.

- **Evaluation Metric**: $metrics (to be predicted)
- **Constraint**: The allowable ranges for the hyperparameter are:
    $Region_ICL_examples

## Reference configurations with Known Performance
Below are examples of configurations that have been evaluated, showing
     their operations and performance metrics:

## Candidate configurations to Evaluate
You must predict performance for these new configurations:

$target_architectures

## Your Task
1. Predict the $metrics value for each candidate configuration
2. Base your predictions on patterns in the reference examples

## Output Format
Each evaluation has to follow this format:

$surrogate_model_response_format

Provide your response in a JSON list containing each proposed
    evaluation.
Return only the required JSON list output without additional text.
```

Listing 4: Prompt template used for performance prediction in the LLM_Generate function.

```
Suggest 8 new candidate point(s) for maximizing a blackbox function in
     a 6-dimensional search space.

Below are some examples of previously evaluated points with their
    corresponding function values:
[
    {
        "x1": 0.034,
        "x2": 0.287,
        "x3": 0.773,
        "x4": 0.175,
        "x5": 0.755,
        "x6": 0.608,
        "value": -37.093
    },
    {
        "x1": 0.199,
        "x2": 0.433,
        "x3": 0.405,
        "x4": 0.779,
```

```
        "x5": 0.186,
        "x6": 0.594,
        "value": -37.84
    },
    {
        "x1": 0.447,
        "x2": 0.342,
        "x3": 0.97,
        "x4": 0.087,
        "x5": 0.115,
        "x6": 0.533,
        "value": -44.52
    },
    {
        "x1": 0.949,
        "x2": 0.127,
        "x3": 0.659,
        "x4": 0.546,
        "x5": 0.049,
        "x6": 0.265,
        "value": -33.067
    }
]

The search space is defined by the following bounding boxes:
   x1_min: 0.492, x1_max: 1.000
   x2_min: 0.000, x2_max: 1.000
   x3_min: 0.000, x3_max: 1.000
   x4_min: 0.000, x4_max: 1.000
   x5_min: 0.000, x5_max: 1.000
   x6_min: 0.000, x6_max: 1.000

Based on the examples above, suggest candidate points that balance
    exploration (sampling new regions) with exploitation (focusing on
    promising areas where function values are good). Each candidate
    point must lie within the specified bounding boxes. In addition,
    predict an estimated function value for each candidate.

Return the suggestions in the following JSON format exactly, without
    any additional text:
[{"x1": float, "x2": float, "x3": float, "x4": float, "x5": float, "x6
    ": float, "value": float}]
```

Listing 5: Prompt example used for simultaneous candidate generation and performance prediction in the LLM_Generate function for NAS-Bench-201.