SAGELLM-BO: HIGH-DIMENSIONAL BAYESIAN OPTIMIZATION VIA LARGE LANGUAGE MODEL-GUIDED SEARCH GEOMETRY

Anonymous authors
Paper under double-blind review

ABSTRACT

Bayesian Optimization (BO) is a powerful framework for optimizing expensive black-box functions, yet its performance deteriorates in high-dimensional settings. Instead of operating over the full domain, prior works aim to define a suitable *search geometry*, a structured region within which BO can operate more efficiently, typically in the form of a low-dimensional subspace and/or a local trust region. However, these methods often rely on problem-specific heuristics and predefined adaptation rules, limiting their robustness. Recent BO methods have shown that Large Language Models (LLMs) can assist the BO process, but they do not explicitly address high-dimensional search and can incur substantial inference costs in such settings. In this work, we propose SageLLM-BO, a novel high-dimensional BO algorithm that leverages the reasoning capabilities of LLMs in a cost-efficient manner to guide the search geometry selection, replacing fixed adaptation rules with adaptive LLM decisions informed by optimization indicators. We further introduce a methodology for constructing a set of example search geometries with their associated quality estimates, and various optimization indicators to guide the LLMs in selecting an effective search geometry. Extensive experiments on synthetic and real-world benchmarks demonstrate that SageLLM-BO consistently matches or outperforms state-of-the-art non-LLM and LLM-based high-dimensional BO methods while substantially reducing LLM cost.

1 INTRODUCTION

Bayesian Optimization (BO) (Garnett, 2023) is a powerful method for optimizing expensive black-box objective functions. Its applications include but are not limited to hyperparameter optimization (Turner et al., 2021; Wang et al., 2024), neural architecture search (Ru et al., 2021; Shen et al., 2023), physics (Roussel et al., 2024), material design (González-Duque et al., 2024; Xu et al., 2025), and robotics (Mayr et al., 2022; Berkenkamp et al., 2023).

Despite being powerful, BO is typically limited to low-dimensional settings (e.g., fewer than 20 dimensions) due to the curse of dimensionality (Eriksson et al., 2019; Papenmeier et al., 2023; Hoang et al., 2025). As dimensionality increases, surrogate models become harder to fit accurately under limited budgets, while acquisition functions tend to overexplore due to high uncertainty in the search space. A promising direction is to avoid searching the full high-dimensional space and instead define an appropriate *search geometry* - a structured region of the original search space within which BO can find the global optimum of the objective function more efficiently. Two primary strategies for constructing search geometries are commonly adopted. Subspace embedding methods assume that the objective function admits a low-dimensional structure and perform BO within reduced (low-dimensional) subspaces defined by a projection of the original high-dimensional search space (Nayebi et al., 2019; Papenmeier et al., 2023; Ngo et al., 2025). In contrast, trust region methods partition the original high-dimensional domain into smaller local regions within which BO is performed, encouraging exploitation in promising areas (Eriksson et al., 2019; Wang et al., 2020; Namura & Takemori, 2025). The two strategies are complementary and can be combined by constructing trust regions within low-dimensional subspaces, further improving BO performance (Papenmeier et al., 2023). Despite their effectiveness, these methods rely heavily on manually designed heuristics to define effective search geometries, limiting their robustness across diverse problems.

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.

Recent advances in Large Language Models (LLMs) have demonstrated strong capabilities in reasoning, pattern recognition, and sequential decision-making (Wei et al., 2022; Shinn et al., 2023; Feng et al., 2026). These abilities stem from extensive pre-training on large-scale corpora that encode vast structured knowledge, including optimization principles. Motivated by this potential, several studies have integrated LLMs into BO to guide the optimization process and reduce reliance on manually designed heuristics (Liu et al., 2024; Cissé et al., 2025; Suwandi et al., 2025; Ngo et al., 2026). However, these existing LLM-guided BO methods primarily target low- or moderate-dimensional problems, and their effectiveness in high-dimensional settings remains largely unexplored. Moreover, some methods require high-dimensional optimization data as LLM inputs and/or generate high-dimensional candidates as outputs (Liu et al., 2024; Suwandi et al., 2025), leading to substantial token cost and potentially prohibitive computational overhead when multiple LLM calls are required per query (Liu et al., 2024). See Appendix F.1 for a comparison.

In this paper, we propose a novel LLM-guided BO algorithm for high-dimensional optimization that follows the approach of defining an effective search geometry to improve optimization efficiency. Rather than relying on fixed and manually designed adaptation heuristics to define the search geometry, we leverage LLM reasoning via in-context learning to adaptively define the search geometry. Specifically, when a search geometry needs to be updated, we provide the LLMs with the relevant BO knowledge, a set of example search geometries with their quality estimates, and multiple optimization indicators, including the historical optimization data from previous iterations and the current surrogate model statistics. We further introduce a methodology to construct a rich and effective set of example search geometries through two complementary mechanisms: subspace embedding and trust region strategies, integrating various effective techniques from both paradigms. To maintain reasonable LLM cost, SageLLM-BO uses compact optimization summaries instead of the full observed dataset, generates search geometry configurations rather than high-dimensional candidate solutions, and queries the LLM only when search geometry updates are triggered. We name our method *Adaptive SeArch GEometry Guided By Large Language Model for High-Dimensional Bayesian Optimization* (SageLLM-BO). Our experimental results demonstrate that SageLLM-BO consistently outperforms state-of-the-art high-dimensional and LLM-based BO methods across a range of synthetic and real-world benchmark problems.

We summarize our contributions as follows.

- We propose SageLLM-BO, a novel high-dimensional BO algorithm that uses LLM reasoning to adaptively guide the search geometry via optimization indicators and example search geometries. We further introduce a methodology to construct a diverse set of example search geometries by integrating subspace embedding and trust region strategies.
- We develop a cost-efficient LLM design for high-dimensional BO that uses compact optimization summaries, generates search geometry configurations rather than high-dimensional solutions, and invokes the LLM only when search geometry needs update.
- We empirically show that SageLLM-BO outperforms the state-of-the-art baselines on a comprehensive set of synthetic and real-world benchmark problems.

2 Background & Related Works

2.1 Bayesian Optimization

Without loss of generality, we consider a minimization problem of an *expensive black-box* objective function $f : \mathcal{X} \to \mathbb{R}$ where $\mathcal{X} \subset \mathbb{R}^{d_\mathcal{X}}$ is a $d_\mathcal{X}$-dimensional search space. The goal of Bayesian Optimization (BO) is to find the global minimum $\mathbf{x}^*$ of f *using the least function evaluations*: $\mathbf{x}^* = \arg\min_{\mathbf{x}\in\mathcal{X}} f(\mathbf{x})$. BO proceeds in an iterative fashion (Garnett, 2023). At iteration t, it trains a surrogate model from the observed dataset $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{t-1}$ to approximate the objective function f, and selects the next evaluation point $\mathbf{x}_t$ by maximizing an acquisition function $\alpha : \mathcal{X} \to \mathbb{R}$.

2.2 High-Dimensional BO with Effective Search Geometry

In this section, we present two common techniques to define effective search geometries for high-dimensional optimization: subspace embedding and trust region methods.

Subspace Embedding. Subspace embedding methods define an effective search geometry by assuming that the objective admits a low-dimensional *active subspace* $\mathcal{Z}$ with dimensionality $d_{\mathcal{Z}} < d_{\mathcal{X}}$ and a projection $h : \mathcal{X} \to \mathcal{Z}$, such that the high-dimensional objective f can be represented by a low-dimensional function $g : \mathcal{Z} \to \mathbb{R}$ with $g(h(\mathbf{x})) = f(\mathbf{x})$. Since $\mathcal{Z}$ is unknown in practice, subspace embedding methods instead construct a low-dimensional search subspace $\mathcal{P} \subset \mathbb{R}^{d_{\mathcal{P}}}$, $d_{\mathcal{P}} < d_{\mathcal{X}}$, together with a mapping $\phi : \mathcal{P} \to \mathcal{X}$. BO is performed in $\mathcal{P}$ by optimizing $\mathbf{p} \in \arg\max_{\mathbf{p} \in \mathcal{P}} \alpha(\mathbf{p})$, while ϕ reconstructs the full-dimensional candidate $\mathbf{x} \in \mathcal{X}$. Here, $\mathbf{p}$ denotes the selected important variables, i.e., those expected to affect the objective function the most. The remaining variables $\mathbf{q}$ form a complementary subspace $\mathcal{Q}$. The mapping ϕ can be represented by a fill-in strategy $\mathcal{F} : \mathcal{P} \to \mathcal{Q}$, such that $\mathbf{q} = \mathcal{F}(\mathbf{p})$ and $\mathbf{x} = (\mathbf{p}, \mathbf{q}) = (\mathbf{p}, \mathcal{F}(\mathbf{p}))$. Existing subspace embedding methods mainly follow two paradigms. First, *variable selection* identifies important variables via principled mechanisms (e.g., GP lengthscales) and use best-k or random fill-in strategies (Song et al., 2022; Hoang et al., 2025). Second, *random linear embedding* leverages random linear mappings as fill-in strategies for computing unimportant variables from randomly sampled important variables (Nayebi et al., 2019; Papenmeier et al., 2022). Further details are provided in Appendix A.1.

Trust Region. Trust region methods define a localized search geometry by restricting BO to a smaller local region $\mathcal{R} \subset \mathcal{X}$, referred to as a trust region, within which BO can perform more efficiently. Trust regions improve optimization efficiency by enabling more accurate local surrogate modeling while focusing exploitation on promising regions likely to contain the global optimum (Eriksson et al., 2019; Ngo et al., 2024). Hyperrectangular trust regions are widely used due to their strong empirical performance, but typically rely on predefined heuristics to adapt the region size and shape, as well as when and how aggressively it expands or shrinks (Eriksson et al., 2019). Details on different trust region methods are in Appendix A.2.

2.3 LLM-Guided Bayesian Optimization

With the rapid advancement of Large Language Models (LLMs), recent works have increasingly incorporated LLMs into BO. LLAMBO (Liu et al., 2024) applies LLMs to warm-starting, surrogate modeling, and candidate sampling, while LLM-Process (Requeima et al., 2024) constructs LLM-based surrogate models. CAKE (Suwandi et al., 2025) employs LLM-guided evolutionary operators to design GP kernels. However, these methods do not explicitly address the curse of dimensionality and may incur substantial costs in high-dimensional settings by using the full observed dataset or generating high-dimensional candidates. Xiang & Lu (2025) target high-dimensional BO but requires task-specific fine-tuning of the LLM. LMABO (Ngo et al., 2026) reduces token costs using summarized optimization information to select acquisition functions, yet lacks an explicit high-dimensional mechanism. In contrast, we adapt common high-dimensional search geometry mechanisms while maintaining reasonable LLM costs. We compare SageLLM-BO with two state-of-the-art methods LLAMBO and LMABO in both optimization performance and token cost. Other approaches are not included due to their prohibitively high costs (Suwandi et al., 2025), task-specific fine-tuning (Xiang & Lu, 2025), or weaker performance than LMABO (Requeima et al., 2024).

3 Proposed Method

In this section, we present SageLLM-BO, a high-dimensional BO method that leverages LLMs in a cost-efficient manner to adaptively define the search geometry via subspace embedding (Sec. 3.1) and trust region (Sec. 3.2) mechanisms. We then describe the full pipeline of the proposed method in Sec. 3.3. An overview of the proposed method is illustrated in Fig. 1 and Alg. 1.

3.1 LLM-Guided Subspace Embedding

We now propose an LLM-guided subspace embedding mechanism that adaptively constructs an effective low-dimensional subspace $\mathcal{P}$. Given a high-dimensional domain $\mathcal{X} \subset \mathbb{R}^{d_{\mathcal{X}}}$, an embedded subspace $\mathcal{P}$ is defined by three parameters. First, the *subspace dimensionality* $d_{\mathcal{P}}$ determines the number of important variables. A small value may fail to capture the objective's underlying structure, whereas a large value exacerbates the curse of dimensionality. Second, the *important variable index set* $\mathbf{P}$, with $|\mathbf{P}| = d_{\mathcal{P}}$, specifies the dimensions retained from the $\mathcal{X}$, where $\mathbf{P} \subset \mathbf{X}$ and $\mathbf{X} = \{1, \ldots, d_{\mathcal{X}}\}$ denotes the index set of all variables. Incorrectly selecting $\mathbf{P}$ may focus the search

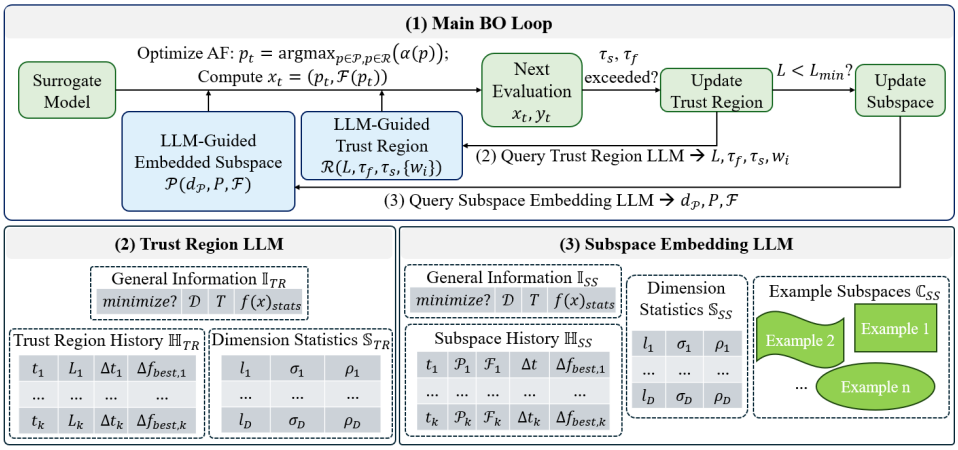


Figure 1: Illustration of our SageLLM-BO algorithm. (1) The main BO loop is conducted within the LLM-guided embedded subspace $\mathcal{P}$ inside the LLM-guided trust region $\mathcal{R}$. (2) The LLM-guided trust region $\mathcal{R}$ is updated when the failure/success threshold τ_s, τ_f is violated, and the update is based on $\mathbb{I}_{TR}$, $\mathbb{H}_{TR}$ and $\mathbb{S}_{TR}$. (3) The LLM-guided subspace $\mathcal{P}$ is updated when the current trust region shrinks too small, and the update is based on $\mathbb{I}_{SS}$, $\mathbb{H}_{SS}$, $\mathbb{S}_{SS}$, and $\mathbb{C}_{SS}$.

on less influential variables and degrade optimization performance. Optimizing the acquisition function within the subspace indexed by $\mathbf{P}$ yields the important variables $\mathbf{p}$. Third, the *fill-in strategy* $\mathcal{F}$ determines the remaining variables as $\mathbf{q} = \mathcal{F}(\mathbf{p})$. Their index set $\mathbf{Q}$ is defined as $\mathbf{Q} = \mathbf{X} \setminus \mathbf{P}$. Thus, a subspace is characterized by $\mathcal{P}(d_\mathcal{P}, \mathbf{P}, \mathcal{F})$, and the full high-dimensional vector is $\mathbf{x} = (\mathbf{p}, \mathbf{q})$. Next, we describe the information provided to the LLM to determine the subspace configuration $(d_\mathcal{P}, \mathbf{P}, \mathcal{F})$, including example subspaces (Sec. 3.1.1) and optimization indicators (Sec. 3.1.2).

3.1.1 The Set of Example Embedded Subspaces $\mathbb{C}_{SS}$

In this section, we propose a methodology to construct a set of example subspaces by combining different choices of $d_\mathcal{P}$, $\mathbf{P}$, $\mathcal{F}$, together with their corresponding quality estimates, which are provided to the LLM to reason about and generate an effective subspace configuration.

The Subspace Dimensionality $d_\mathcal{P}$. We generate m approximately equally spaced candidate dimensionalities over $(d_0, d_\mathcal{X}]$, encouraging progressive expansion from the current dimensionality d_0. Starting from a low-dimensional subspace improves optimization efficiency, while gradually increasing $d_\mathcal{P}$ increases the chance of capturing relevant dimensions (Papenmeier et al., 2022; 2023). Formally, $d_\mathcal{P} \in \{d_0 + \lfloor (d_\mathcal{X} - d_0)i/(m+1) \rceil \, ; i = 1, \ldots, m\}$. A larger m provides a richer set of example subspaces but increases both quality-evaluation computational cost and LLM token costs. See Appendix F.3 for the sensitivity analysis of m.

The Important Variable Index Set P. We consider two complementary strategies. The random strategy $\mathbf{P}_{\texttt{rand}}$ promotes unbiased exploration by uniformly sampling $d_\mathcal{P}$ distinct indices from $\mathbf{X}$ (Papenmeier et al., 2022), i.e., $\mathbf{P}_{\texttt{rand}} = \{k_i \mid i = 1, \ldots, d_\mathcal{P}\}$, where k_i are sampled without replacement from $\{1, \ldots, d_\mathcal{X}\}$. In contrast, the lengthscale strategy $\mathbf{P}_{\texttt{len}}$ exploits the inverse GP lengthscales $\{\rho_i\}_{i=1}^{d_\mathbf{X}}$ from the full-dimensional surrogate $\mathcal{GP}_\mathcal{X}$ trained on the current observed dataset. Since larger inverse lengthscales indicate greater sensitivity (Hoang et al., 2025), $\mathbf{P}_{\texttt{len}}$ selects the $d_\mathcal{P}$ indices with the largest ρ_i, i.e., $\mathbf{P}_{\texttt{len}} = \{k_i \mid i = 1, \ldots, d_\mathcal{P}\}$ such that $\rho_i \geq \rho_j$ for all $i \in \mathbf{P}_{\texttt{len}}$ and $j \notin \mathbf{P}_{\texttt{len}}$. This prioritizes dimensions estimated to have the strongest influence on the objective. However, under limited evaluation budgets, high-dimensional GP lengthscale estimates may be unreliable, thus $\mathbf{P}_{\texttt{rand}}$ provides a complementary exploratory alternative.

The Fill-in Strategy $\mathcal{F}$. We consider three complementary fill-in strategies for determining the unimportant variables $\mathbf{q} = [q_i]_{i=1}^{d_\mathcal{Q}}$. The random strategy $\mathcal{F}_{\texttt{rand}}$ promotes exploration by independently sampling each variable from its feasible range (Hoang et al., 2025), i.e., $q_i \sim \mathcal{U}([\texttt{l}_i, \texttt{u}_i])$, where $\texttt{l}_i$ and $\texttt{u}_i$ denote the lower and upper bounds of the i-th variable. The best-k strategy $\mathcal{F}_{\texttt{bestk}}$ promotes exploitation by sampling from the corresponding coordinates of the current best-k solu-

tions (Song et al., 2022), $q_i \sim \mathcal{U}(\mathcal{K}_i)$, where $\mathcal{K}_i = \{x_{i,j}\}_{j=1}^{k}$ contains the i-th coordinate values of the best-k data points $\{\mathbf{x}_j\}_{j=1}^{k}$ ranked by objective value from the current observed dataset. The linear strategy $\mathcal{F}_{\texttt{lin}}$ uses a random linear embedding to jointly determine important and unimportant variables, thereby capturing their interaction (Nayebi et al., 2019). Specifically, the full vector $\mathbf{x} \in \mathcal{X}$ is constructed from $\mathbf{p} \in \mathcal{P}$ using a sparse embedding matrix $\mathbf{E} \in \{0, \pm 1\}^{d_{\mathcal{X}} \times d_{\mathcal{P}}}$, such that $\mathbf{x} = \mathbf{E}\mathbf{p}$, with $\mathbf{q}$ given by the subvector of $\mathbf{x}$ indexed by $\mathbf{Q}$. Together, these strategies provide different exploration-exploitation trade-off.

The Set of Example Embedded Subspaces $\mathbb{C}_{SS}$. By combining different choices of the three subspace components, we construct a diverse pool of example subspaces and estimate their quality before providing them to the LLM. For each example subspace $\mathcal{P}(d_{\mathcal{P}}, \mathbf{P}, \mathcal{F})$, we first identify its most promising candidate $\hat{\mathbf{x}}$ by optimizing the acquisition function over the $\mathcal{X}$ subject to the constraints induced by $\mathcal{P}(d_{\mathcal{P}}, \mathbf{P}, \mathcal{F})$: $\hat{\mathbf{x}} \in \arg\max_{\mathbf{x} \in \mathcal{X}|\mathcal{P}} \alpha_{\mathcal{X}}(\mathbf{x})$, where $\alpha_{\mathcal{X}}$ is computed based on $\mathcal{GP}_{\mathcal{X}}$ trained on the current observed dataset (see Appendix D.1 for implementation details). This ensures all example subspaces are evaluated under the same surrogate $\mathcal{GP}_{\mathcal{X}}$ for consistent comparison. We then evaluate each $\mathcal{P}$ via three indicators evaluated at $\hat{\mathbf{x}}$: (1) the acquisition value $\alpha_{\mathcal{X}}(\hat{\mathbf{x}})$ estimates the quality of the best candidate attainable under the corresponding subspace, (2) the posterior mean $\mu_{\mathcal{X}}(\hat{\mathbf{x}})$ reflects the predicted objective value of $\hat{\mathbf{x}}$, and (3) the posterior standard deviation $\sigma_{\mathcal{X}}(\hat{\mathbf{x}})$ quantifies predictive uncertainty to reason about the exploration exploitation trade-off. Thus, each example subspace in $\mathbb{C}_{SS}$ is represented as $\{\mathcal{P}(d_{\mathcal{P}}, \mathbf{P}, \mathcal{F}), \alpha_{\mathcal{X}}(\hat{\mathbf{x}}), \mu_{\mathcal{X}}(\hat{\mathbf{x}}), \sigma_{\mathcal{X}}(\hat{\mathbf{x}})\}$.

3.1.2 Optimization Indicators For Subspace Embedding LLM

In addition to the example subspace set $\mathbb{C}_{SS}$, we provide the LLM with other optimization indicators, which are designed for cost-efficient high-dimensional BO.

The General Optimization Information $\mathbb{I}_{SS}$. We first provide $\mathbb{I}_{SS}$, which summarizes the current optimization state, including the problem dimensionality, maximum and remaining evaluation budgets, optimization goal (minimization or maximization), and a compact summary of the current observed dataset $\mathcal{D}_t = \{(\mathbf{x}_i, y_i)\}_{i=1}^{t}$. Providing the full dataset would require $\mathcal{O}(d_{\mathcal{X}} t)$ values and become costly in high-dimensional settings. Instead, we characterize the spatial distribution of observations using their Euclidean distances to the latest point, $h_i = \|\mathbf{x}_t - \mathbf{x}_i\|_2$, then summarize both $\{h_i\}$ and the objective values $\{y_i\}$ using their minimum, maximum, mean, and standard deviation. This provides compact information about search behavior and objective-value distribution while keeping the summary size independent of the dimensionality $d_{\mathcal{X}}$ and the number of observations t.

The Subspace History $\mathbb{H}_{SS}$. Next, we provide $\mathbb{H}_{SS}$, which summarizes the performance of previously explored subspaces $\{\mathcal{P}_i\}_{i=1}^{k}$. For each $\mathcal{P}_i$, we record its starting iteration t_i, subspace information $(d_{\mathcal{P},i}, P_i, \mathcal{F}_i)$, duration Δt_i, and incumbent improvement $\Delta f_{\text{best},i} = f_{\text{best},i-1} - f_{\text{best},i}$ where $f_{\text{best},i}$ denotes the incumbent value after exploring the i-th subspace. Together, the improvement and duration characterize how effectively each subspace used its evaluation budget. By linking past subspace configurations $(d_{\mathcal{P},i}, P_i, \mathcal{F}_i)$ to their performance ($\Delta f_{\text{best},i}$ over Δt_i iterations), $\mathbb{H}_{SS}$ helps the LLM decide whether the next subspace should make minor or substantial adjustments.

The Dimension Statistics $\mathbb{S}_{SS}$. Finally, we provide dimension-wise statistics $\mathbb{S}_{SS}$ to guide selection of the important variable set $\mathbf{P}$. For each dimension i, we compute three complementary quantities: (1) the GP lengthscale l_i from the full-dimensional surrogate $\mathcal{GP}_{\mathcal{X}}$, where smaller values indicate greater sensitivity; (2) the absolute Spearman rank correlation $|\xi_i|$ between $\{x_{j,i}\}_{j=1}^{t}$ and $\{f_j\}_{j=1}^{t}$, measuring model-independent monotonic association with the objective (Spearman, 1961); and (3) the empirical standard deviation $s_i = \text{std}(\{x_{j,i}\}_{j=1}^{t})$, measuring search coverage. A small s_i indicates limited exploration, making its estimated importance (l_i and $|\xi_i|$) less informative and suggesting greater scope for further exploration. Together, these statistics capture surrogate sensitivity, model-dependent association, and exploration coverage to guide the selection of $\mathbf{P}$.

3.1.3 The Overall LLM-Guided Subspace Embedding

When the Subspace Embedding LLM is triggered, we construct a new embedded subspace by prompting the LLM using the example subspace set $\mathbb{C}_{SS}$, the general optimization information $\mathbb{I}_{SS}$, the subspace history $\mathbb{H}_{SS}$, and the dimension statistics $\mathbb{S}_{SS}$. The prompt first assigns the LLM the role of an "expert in Bayesian Optimization", then describes the structure of $\mathbb{C}_{SS}$, $\mathbb{I}_{SS}$, $\mathbb{H}_{SS}$, and

$\mathbb{S}_{SS}$. Finally, the prompt enforces a structured output format to ensure reliable parsing. An example prompt and LLM response are provided in Appendix H. The LLM outputs an embedded subspace configuration $\mathcal{P}(d_{\mathcal{P}}, \mathbf{P}, \mathcal{F})$, within which we optimize the acquisition function to obtain the important variables $\mathbf{p}$. The remaining variables $\mathbf{q}$ are then determined by the selected fill-in strategy $\mathcal{F}$, yielding $\mathbf{x} = (\mathbf{p}, \mathbf{q}) \in \mathcal{X}$ for objective function evaluation.

3.2 LLM-Guided Trust Region

In this section, we propose an LLM-guided mechanism for adaptively defining the trust region during optimization. In SageLLM-BO, the trust region $\mathcal{R}$ lies within a subspace $\mathcal{P} \subseteq \mathcal{X}$ generated by the Subspace Embedding LLM and is defined as a hyperrectangle centered at the incumbent $\mathbf{p}^* \in \mathcal{P}$: $\mathcal{R} = \{\mathbf{p} \in \mathcal{P} : |p_i - p_i^*| \leq L w_i,\ i = 1, \ldots, d_{\mathcal{P}}\}$. The base side length L controls the overall search scale, balancing broader exploration and local exploitation, while the weights $\{w_i\}$ adapt the shape of the region to differences in sensitivity across dimensions. In addition, the trust region relies on the success and failure thresholds τ_s and τ_f to control the its adaptation rate by determining how many consecutive iterations of incumbent improvement or stagnation is required before expanding or shrinking the region. Thus, a trust region is characterized as $\mathcal{R}(L, \{w_i\}, \tau_s, \tau_f)$. Next, we describe the optimization indicators provided to the LLM to determine the configuration $(L, \{w_i\}, \tau_s, \tau_f)$, including the general information, subspace history, and dimension statistics.

3.2.1 Optimization Indicators For Trust Region LLM

We introduce three cost-efficient optimization indicators for the Trust Region LLM, similar to those for the Subspace Embedding LLM $\{\mathbb{I}_{SS}, \mathbb{H}_{SS}, \mathbb{S}_{SS}\}$. Due to space constraints, we only present the key information here, with further details and rationales provided in Appendix B. First, the *general optimization information* $\mathbb{I}_{TR}$ reuses the compact optimization dataset summaries from $\mathbb{I}_{SS}$, including the problem dimensionality, optimization goal, and evaluation budget. Second, the *trust region history* $\mathbb{H}_{TR}$ summarizes the evolution and performance of trust regions $\{\mathcal{R}_i\}_{i=1}^{k}$ explored so far within the current subspace by recording, for each $\mathcal{R}_i$, its starting iteration t_i, configuration $(L_i, w_{j,i}, \tau_{s,i}, \tau_{f,i})$, duration Δt_i, and improvement in the incumbent objective value $\Delta f_{\text{best},i}$. Finally, the *dimension statistics* $\mathbb{S}_{TR}$ guide the weight ratios $\{w_i\}$ using the GP lengthscale l_i, absolute Spearman correlation $|\xi_i|$, and empirical standard deviation s_i for each dimension in $\mathcal{P}$.

3.2.2 The Overall LLM-Guided Trust Region

Given a subspace $\mathcal{P}$ generated by the Subspace Embedding LLM, we infer the trust region configuration $\mathcal{R}(L, \{w_i\}, \tau_s, \tau_f)$ within $\mathcal{P}$ using the optimization indicators described in Sec. 3.2.1, namely $\mathbb{I}_{TR}$, $\mathbb{H}_{TR}$, $\mathbb{S}_{TR}$. Since L, $\{w_i\}$, τ_s, and τ_f control different aspects of the trust region and depend on different optimization indicators, we update them separately through different queries and prompts to the Trust Region LLM (see Alg. 1 for more details). When updating one parameter, the others retain their previous values. Specifically, whenever a trust region update is triggered, either by consecutive successes or failures, the side length L is updated using $\mathbb{I}_{TR}$ and $\mathbb{H}_{TR}$. If the update is triggered by consecutive successes, τ_s is further updated, otherwise τ_f is further updated, both using $\mathbb{I}_{TR}$ and $\mathbb{H}_{TR}$. In contrast, the weight ratios w_i are updated every n_w iterations using $\mathbb{S}_{TR}$. This periodic update allows adaptation to dimension-wise information while limiting LLM token cost. See Appendix F.4 for a sensitivity analysis of the parameter n_w.

Similar to the Subspace Embedding LLM, each Trust Region LLM prompt provides the associated optimization information ($\mathbb{I}_{TR}$, $\mathbb{H}_{TR}$, $\mathbb{S}_{TR}$) and enforce a structured output format of the quantity to be inferred. Depending on the triggering condition, the LLM outputs either the side length update ratio r (where $L \leftarrow Lr$), success threshold τ_s, failure threshold τ_f, or raw weight ratios $\{\tilde{w}_i\}$, which is normalized $w_i = \tilde{w}_i / (\prod_j \tilde{w}_j)^{1/d_{\mathcal{P}}}$ before defining the trust region $\mathcal{R}(L, \{w_i\}, \tau_s, \tau_f)$ for subsequent optimization. Example prompts and LLM responses are provided in Appendix H.

3.3 Overall Method

Overall, SageLLM-BO operates as summarized in Alg. 1 and illustrated in Fig. 1. At each iteration, given the current subspace $\mathcal{P}(d_{\mathcal{P}}, \mathbf{P}, \mathcal{F})$ and trust region $\mathcal{R}(L, w_i, \tau_s, \tau_f)$, we fit a low-dimensional surrogate $\mathcal{GP}_{\mathcal{P}}$ to the projected dataset (line 5) and optimize the acquisition function within $\mathcal{P}$

subject to $\mathcal{R}$ to obtain $\hat{\mathbf{p}}$ (line 6). Details of the projection are provided in Appendix D.1. The fill-in strategy $\mathcal{F}$ determines the remaining variables $\hat{\mathbf{q}}$, yielding $\mathbf{x} = (\hat{\mathbf{p}}, \hat{\mathbf{q}})$ (line 7), which is evaluated and appended to $\mathcal{D}$ (line 8). The trust region is updated according to its triggering conditions. After τ_s consecutive successes, the Trust Region LLM updates the side length through a ratio r (and $L \leftarrow Lr$) and adjusts τ_s (line 11); after τ_f consecutive failures, it similarly updates r and τ_f (line 14). The weights $\{w_i\}$ are updated every n_w iterations (line 18). If $L < L_{\min}$, the current subspace is terminated and the Subspace Embedding LLM constructs a new $\mathcal{P}(d_{\mathcal{P}}, \mathbf{P}, \mathcal{F})$ (line 21). This process continues until the evaluation budget T is exhausted. Together with the compact summaries in $\mathbb{I}_{SS}$ and $\mathbb{I}_{TR}$, which avoid linear growth of the LLM input size with $d_{\mathcal{X}}$ and t, the trigger conditions further improve the scalability of SageLLM-BO.

It's worth noting that SageLLM-BO deliberately excludes semantic information about the objective function, such as variable meanings or application context. This improves robustness across diverse black-box problems by avoiding reliance on task-specific knowledge. Moreover, since the LLM's role is to construct an effective search geometry, optimization indicators such as example geometries and past performance are more directly relevant than semantic descriptions.

4 Experiments

4.1 Experimental Settings

Baseline Methods. We compare **SageLLM-BO** against both state-of-the-art non-LLM high-dimensional BO methods and LLM-based BO methods. Specifically, the non-LLM high-dimensional BO baselines include: (1) *Standard BO*; (2) random subspace embedding methods: *HESBO* (Nayebi et al., 2019), *ALEBO* (Letham et al., 2020), *BAxUS* (Papenmeier et al., 2022), *Bounce* (Papenmeier et al., 2023), *BOIDS* (Ngo et al., 2025), (3) variable selection subspace embedding methods: *LassoBO* (Hoang et al., 2025), *RDUCB* (Ziomek & Ammar, 2023), (4) trust region methods: *TuRBO* (Eriksson et al., 2019), *CMA-BO* (Ngo et al., 2024). The LLM-based BO baselines include: *LMABO* (Ngo et al., 2026), *LLAMBO* (Liu et al., 2024). We run all methods 10 times and report the mean and standard error. Unless stated otherwise, all methods use 1000 iterations and 20 Latin hypercube initial points (Eriksson et al., 2019; Papenmeier et al., 2022). For ALEBO and HESBO, we set two different target dimension $d_{\mathcal{P}} = 10$ and $d_{\mathcal{P}} = 20$, as in prior works (Papenmeier et al., 2022; Ngo et al., 2025). For ALEBO, due to the prohibitive memory and runtime cost, we run 500 iterations, as in prior works (Hoang et al., 2025; Ngo et al., 2025). For LLAMBO, due to the extremely prohibitive LLM token cost, we run 50 iterations and report results in Appendix F.1.

For SageLLM-BO settings, we use GPs with Matern 5/2 kernel and Thompson Sampling acquisition functions, similar to prior works (Eriksson et al., 2019; Papenmeier et al., 2022). For the LLM, we use OpenAI GPT-4.1 (Achiam et al., 2023) for all LLM queries. We set $m = 3$ when generating the example subspaces (Sec. 3.1.1) and $n_w = 10$ for the update frequency of the trust region weights $\{w_i\}$ (Sec. 3.2.2). In the sequel (Sec. 4.3), we also show that SageLLM-BO is robust to the choice of LLM (e.g., GPT-4.1, Mistral-Large-3 and Claude-Haiku-4.5), and the parameters m and n_w. Detailed implementation of SageLLM-BO and the baselines are in Appendix D.

Benchmark Problems. We evaluate SageLLM-BO on 3 synthetic and 3 real-world benchmark problems with dimensionalities ranging from 100 to 180. The synthetic benchmarks are *Ackley-100D*, *Rastrigin-100D*, and *Rosenbrock-100D* (Eriksson et al., 2019; Song et al., 2022; Hoang et al., 2025). The real-world benchmarks are *HalfCheetah-102D*, *Pusher-161D*, and *LassoDNA-180D*. HalfCheetah-102D and Pusher-161D are reinforcement learning tasks implemented using the MuJoCo benchmark suite (Brockman et al., 2016), where the objective is to minimize the negative reward. LassoDNA-180D is a hyperparameter optimization task based on a microbiology DNA dataset, using the implementation from (Šehić et al., 2022). These synthetic and real-world problems are commonly used in high-dimensional BO studies (Eriksson et al., 2019; Nayebi et al., 2019; Papenmeier et al., 2022; Song et al., 2022; Ngo et al., 2025). Further details are in Appendix E.

4.2 Comparison with Baselines

Optimization Performance. Fig. 2 compares SageLLM-BO with all baselines on six benchmark problems. On the synthetic problems, SageLLM-BO generally achieves faster convergence and bet-

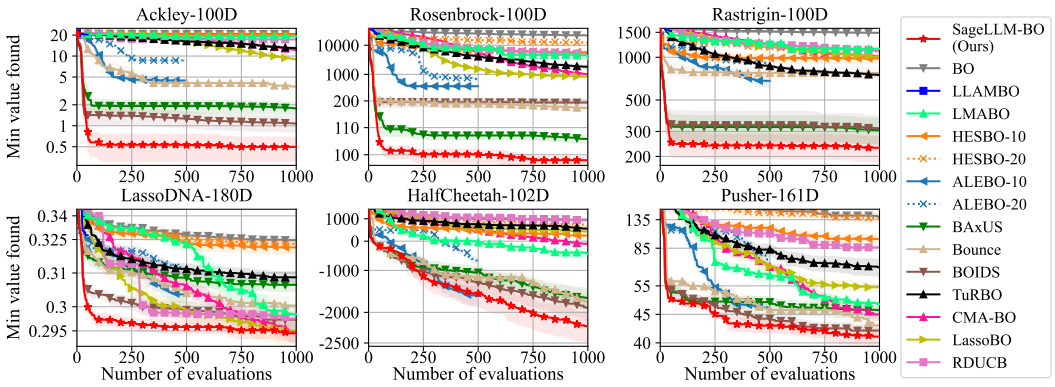


Figure 2: Comparison of SageLLM-BO against the state-of-the-art baselines on six benchmark problems. Due to prohibitive runtime and memory required, we run ALEBO-10 and ALEBO-20 with 500 iterations. Due to prohibitive LLM cost, we run LLMABO with 50 iterations, with details in Appendix F.1. Overall, SageLLM-BO outperforms all baselines substantially on most problems.

ter final performance. In particular, on the highly multi-modal Rastrigin-100D, BAxUS and BOIDS converge prematurely, whereas SageLLM-BO continues to improve. On the real-world problems, SageLLM-BO remains competitive: it overtakes BAxUS and BOIDS on HalfCheetah-102D after about 300 evaluations, outperforms BOIDS throughout Pusher-161D, and reaches a similar final solution to LassoBO and CMA-BO on LassoDNA-180D while converging faster in the first 200 evaluations. Existing LLM-based methods LMABO and LLAMBO (Fig. 4, Appendix F.1) are also outperformed by SageLLM-BO and several non-LLM high-dimensional BO methods. Overall, SageLLM-BO's improvement over heuristic subspace methods (BAxUS, Bounce) and trust region methods (TuRBO, CMA-BO) support the benefit of adaptively defining search geometry with an LLM rather than fixed heuristics. The improvement of SageLLM-BO over LMABO and LLAMBO further highlights the value of explicitly addressing high dimensionality in LLM-guided BO.

LLM Inference Costs. We compare the costs of SageLLM-BO, LLAMBO, and LMABO in Table 1 of Appendix F.1. Using the same GPT-4.1 model, SageLLM-BO incurs substantially lower inference costs. LLAMBO requires 93k–210k tokens per iteration and 10.17–23.09 USD for only 50 iterations, largely because it repeatedly provides the full observed dataset to the LLM and generates high-dimensional candidates. LMABO is considerably cheaper, but still uses 942–977 tokens per iteration due to per-iteration LLM queries. In contrast, SageLLM-BO uses only 279–374 tokens per iteration and costs 0.58–0.80 USD over the full 1000-evaluation budget. Overall, SageLLM-BO demonstrates substantially better LLM cost scalability for high-dimensional optimization.

4.3 Ablation Studies And Sensitivity Analyses

We conduct two ablation studies to analyze the effectiveness of the LLM-guided components in SageLLM-BO, and three sensitivity analyses of the LLM choice, parameter m, and parameter n_w.

Effectiveness of the Subspace Embedding LLM. In Fig. 3, we investigate the effectiveness of the proposed subspace embedding LLM by comparing SageLLM-BO against 3 variants. The first variant, "*No Subspace*", removes the subspace component and optimizes directly in $\mathcal{X}$. The second variant, "*Random Subspace*", uniformly samples a subspace $\mathcal{P}^*$ from the example set $\mathbb{C}_{SS}$, i.e., $\mathcal{P}^* \sim \mathcal{U}(\mathbb{C}_{SS})$. The third variant, "*AcqF Subspace*", selects the candidate with the largest acquisition quality score $\alpha_{\mathcal{P}}$. For each $\mathcal{P} \in \mathbb{C}_{SS}$, we define $\alpha_{\mathcal{P}} = \alpha_{\mathcal{X}}(\hat{\mathbf{x}}) = \max_{\mathbf{x} \in \mathcal{X}|\mathcal{P}} \alpha_{\mathcal{X}}(\mathbf{x})$, where $\hat{\mathbf{x}}$ is the most promising candidate under $\mathcal{P}$ (Sec. 3.1.1). The selected subspace is then $\mathcal{P}^* \in \arg\max_{\mathcal{P} \in \mathbb{C}_{SS}} \alpha_{\mathcal{P}}$. This provides a strong non-LLM baseline that uses the same acquisition-based quality indicator as $\mathbb{C}_{SS}$, but ignores the additional optimization context provided by $\mathbb{I}_{SS}$, $\mathbb{H}_{SS}$, and $\mathbb{S}_{SS}$. Note that in all these variants, the Trust Region LLM remains unchanged. Fig. 3 shows that SageLLM-BO outperforms or matches other variants, suggesting that reasoning jointly over $\mathbb{C}_{SS}$ and other indicators enables more effective subspace adaptation than alternative heuristics.

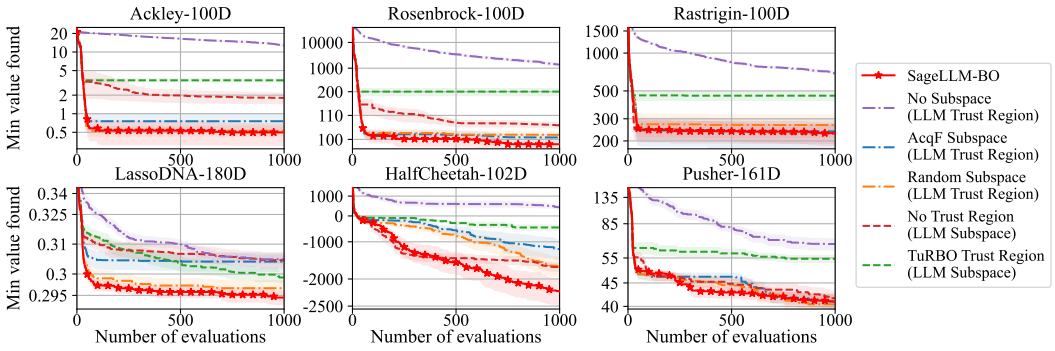


Figure 3: Ablation study on the LLM-guided subspace embedding and LLM-guided trust region components. SageLLM-BO matches or outperforms other subspace embedding and trust region heuristics, demonstrating the effectiveness of our LLM-guided components

Effectiveness of the Trust Region LLM. Additionally, in Fig. 3, we also investigate the effectiveness of the proposed trust region LLM by comparing SageLLM-BO against 2 variants. The first variant, "*No Trust Region*", removes the trust region and optimizes over the entire subspace. The second variant, "*TuRBO Trust Region*", replaces the Trust Region LLM with the TuRBO heuristic (Eriksson et al., 2019). Note that in all these variants, the Subspace Embedding LLM remains unchanged. Fig. 3 shows that SageLLM-BO performs better than or comparably to both. By conditioning on the provided indicators $\mathbb{I}_{TR}$, $\mathbb{H}_{TR}$, $\mathbb{S}_{TR}$, the LLM can adaptively control the trust region size and shape via the side length and weights, and its responsiveness to optimization progress through the success and failure thresholds, yielding more effective local search than fixed heuristics.

Sensitivity Analysis. We further analyze the sensitivity of SageLLM-BO to the choice of LLM (Appendix F.2), m (Appendix F.3), and n_w (Appendix F.4). Overall, SageLLM-BO remains robust across the evaluated settings, supporting its effectiveness across a broad range of configurations.

5 Discussion and Limitations

Despite the promising results of SageLLM-BO against various baselines on benchmark problems, several challenges remain for further investigation. First, extending SageLLM-BO to incorporate more sophisticated high-dimensional optimization techniques (Wang et al., 2020; Song et al., 2022; Ngo et al., 2024) is interesting but non-trivial. More complex techniques may require the LLM to understand richer optimization mechanisms and make more structured decisions, increasing prompt complexity, token consumption, and the difficulty in obtaining reliable LLM outputs. Second, prompt design in SageLLM-BO could benefit from automated prompt optimization techniques (Yang et al., 2023; Guo et al., 2023). However, this also non-trivial: direct prompt evaluation may require repeated expensive black-box evaluations, while surrogate-based evaluation may be unreliable and involve repeated computationally expensive GP fitting and acquisition function optimization. Hence, principled and efficient prompt optimization remains a direction for future works.

6 Conclusion

In this work, we present SageLLM-BO, an LLM-guided high-dimensional BO algorithm. Our key idea is to leverage LLM reasoning to define an effective search geometry as a local region within a low-dimensional subspace. By providing the LLMs with a set of example subspaces and various optimization indicators, SageLLM-BO can find a suitable search geometry in which BO can operate more efficiently. SageLLM-BO also develops a cost-efficient design that provides compact optimization information to the LLMs, reducing token costs. Our experimental results show that SageLLM-BO outperforms state-of-the-art solutions on a wide range of synthetic and real-world benchmark problems. The code implementation of SageLLM-BO can be found here `https://anonymous.4open.science/r/sagellm-bo/`.

AI USE STATEMENT

In this work, we used generative AI tools for providing feedback on research methodology and experiments. We have not used generative AI tools for generating synthetic datasets, developing conceptual frameworks, implementing the methods, supporting qualitative and thematic data analysis, cleaning and reformatting dataset, interpreting the results. Developing theoretical models, formulating mathematical claims, proving mathematical claims, writing of proofs, proposing and refining hypothesis, assisting with translation are not applicable to this work. Additionally, we used generative AI tools for editing the research paper to improve readability and proposing a title or keywords for the research paper. We have reviewed all AI-assisted work. We evaluated the LLM-generated feedback on the research methodology and experimental design, using it to identify potential methodological issues and to inform the design of additional experiments to support our claims. We also reviewed all LLM-assisted writing throughout the manuscript and manually verified that the content accurately reflects our proposed methods, experiments, and findings. In addition, we used an LLM to suggest candidate acronyms for the proposed method, from which we selected the final acronym SageLLM-BO. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

The implementation of our proposed algorithm SageLLM-BO can be found here `https://anonymous.4open.science/r/sagellm-bo/`. Detailed settings for reproducibility can be found in Appendix D.1.

REFERENCES

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

Maximilian Balandat, Brian Karrer, Daniel R. Jiang, Samuel Daulton, Benjamin Letham, Andrew Gordon Wilson, and Eytan Bakshy. BoTorch: A Framework for Efficient Monte-Carlo Bayesian Optimization. In *Advances in Neural Information Processing Systems 33*, 2020.

Felix Berkenkamp, Andreas Krause, and Angela P Schoellig. Bayesian optimization with safety constraints: safe and automatic parameter tuning in robotics. *Machine learning*, 112(10):3713–3747, 2023.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

Abdoulatif Cissé, Xenophon Evangelopoulos, Vladimir V. Gusev, and Andrew I. Cooper. Language-based bayesian optimization research assistant (bora). In *Proceedings of the Thirty-Fourth International Joint Conference on Artificial Intelligence*, IJCAI '25, 2025.

Samuel Daulton, David Eriksson, Maximilian Balandat, and Eytan Bakshy. Multi-objective bayesian optimization over high-dimensional search spaces. In *Uncertainty in Artificial Intelligence*, pp. 507–517. PMLR, 2022.

David Eriksson and Martin Jankowiak. High-dimensional Bayesian optimization with sparse axis-aligned subspaces. In *Proceedings of the Thirty-Seventh Conference on Uncertainty in Artificial Intelligence (UAI)*, volume 161, 2021.

David Eriksson and Matthias Poloczek. Scalable constrained bayesian optimization. In *International Conference on Artificial Intelligence and Statistics*, pp. 730–738. PMLR, 2021.

David Eriksson, Michael Pearce, Jacob Gardner, Ryan D Turner, and Matthias Poloczek. Scalable Global Optimization via Local Bayesian Optimization. In *NeuRIPS*, 2019.

Jiazhan Feng, Shijue Huang, Xingwei Qu, Ge Zhang, Yujia Qin, Baoquan Zhong, Chengquan Jiang, Jinxin Chi, and Wanjun Zhong. Retool: Reinforcement learning for strategic tool use in llms. In *International Conference on Learning Representations*, volume 2026, pp. 37909–37926, 2026.

Jacob Gardner, Geoff Pleiss, Kilian Q Weinberger, David Bindel, and Andrew G Wilson. Gpytorch: Blackbox matrix-matrix gaussian process inference with gpu acceleration. *Advances in neural information processing systems*, 31, 2018.

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

Miguel González-Duque, Richard Michael, Simon Bartels, Yevgen Zainchkovskyy, Søren Hauberg, and Wouter Boomsma. A survey and benchmark of high-dimensional bayesian optimization of discrete sequences. *Advances in Neural Information Processing Systems*, 37:140478–140508, 2024.

Qingyan Guo, Rui Wang, Junliang Guo, Bei Li, Kaitao Song, Xu Tan, Guoqing Liu, Jiang Bian, and Yujiu Yang. Connecting large language models with evolutionary algorithms yields powerful prompt optimizers. In *The Twelfth International Conference on Learning Representations*, 2023.

Vu Viet Hoang, Sunil Gupta, Vu Nguyen, et al. High dimensional bayesian optimization using lasso variable selection. In *International Conference on Artificial Intelligence and Statistics*, pp. 3106–3114. PMLR, 2025.

Ben Letham, Roberto Calandra, Akshara Rai, and Eytan Bakshy. Re-examining linear embeddings for high-dimensional bayesian optimization. In *NeurIPS*, volume 33, 2020.

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.

Matthias Mayr, Faseeh Ahmad, Konstantinos Chatzilygeroudis, Luigi Nardi, and Volker Krueger. Skill-based multi-objective reinforcement learning of industrial robot tasks with planning and knowledge integration. In *2022 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pp. 1995–2002. IEEE, 2022.

Nobuo Namura and Sho Takemori. Regional expected improvement for efficient trust region selection in high-dimensional bayesian optimization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 19624–19632, 2025.

Amin Nayebi, Alexander Munteanu, and Matthias Poloczek. A Framework for Bayesian Optimization in Embedded Subspaces. In *ICML*, volume 97. PMLR, 2019.

Giang Ngo, Dat Phan Trong, Dang Nguyen, Sunil Gupta, and Svetha Venkatesh. Adaptive acquisition selection for bayesian optimization with large language models. In *Proceedings of the 13th International Conference on Learning Representations*, 2026.

Lam Ngo, Huong Ha, Jeffrey Chan, Vu Nguyen, and Hongyu Zhang. High-dimensional bayesian optimization via covariance matrix adaptation strategy. *Transactions on Machine Learning Research*, 2024.

Lam Ngo, Huong Ha, Jeffrey Chan, and Hongyu Zhang. Boids: High-dimensional bayesian optimization via incumbent-guided direction lines and subspace embeddings. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 19659–19667, 2025.

Leonard Papenmeier, Luigi Nardi, and Matthias Poloczek. Increasing the scope as you learn: Adaptive bayesian optimization in nested subspaces. *NeurIPS*, 35, 2022.

Leonard Papenmeier, Luigi Nardi, and Matthias Poloczek. Bounce: Reliable high-dimensional bayesian optimization for combinatorial and mixed spaces. *Advances in Neural Information Processing Systems*, 36:1764–1793, 2023.

James Requeima, John Bronskill, Dami Choi, Richard Turner, and David K Duvenaud. Llm processes: Numerical predictive distributions conditioned on natural language. *Advances in Neural Information Processing Systems*, 37:109609–109671, 2024.

Ryan Roussel, Auralee L Edelen, Tobias Boltz, Dylan Kennedy, Zhe Zhang, Fuhao Ji, Xiaobiao Huang, Daniel Ratner, Andrea Santamaria Garcia, Chenran Xu, et al. Bayesian optimization algorithms for accelerator physics. *Physical review accelerators and beams*, 27(8):084801, 2024.

Binxin Ru, Xingchen Wan, Xiaowen Dong, and Michael Osborne. Interpretable neural architecture search via bayesian optimisation with weisfeiler-lehman kernels. In *International Conference on Learning Representations*, 2021.

Kenan Šehić, Alexandre Gramfort, Joseph Salmon, and Luigi Nardi. Lassobench: A high-dimensional hyperparameter optimization benchmark suite for lasso. In *International Conference on Automated Machine Learning*, pp. 2–1. PMLR, 2022.

Yu Shen, Yang Li, Jian Zheng, Wentao Zhang, Peng Yao, Jixiang Li, Sen Yang, Ji Liu, and Bin Cui. Proxybo: Accelerating neural architecture search via bayesian optimization with zero-cost proxies. In *Proceedings of the AAAI conference on artificial intelligence*, volume 37, pp. 9792–9801, 2023.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in neural information processing systems*, 36:8634–8652, 2023.

Lei Song, Ke Xue, Xiaobin Huang, and Chao Qian. Monte carlo tree search based variable selection for high dimensional bayesian optimization. In *NeurIPS*, 2022.

Charles Spearman. The proof and measurement of association between two things. *The American Journal of Psychology*, 1961.

S. Surjanovic and D. Bingham. Virtual library of simulation experiments: Test functions and datasets. Retrieved February 19, 2026, from `http://www.sfu.ca/~ssurjano`, 2013.

Richard Cornelius Suwandi, Feng Yin, Juntao Wang, Renjie Li, Tsung-Hui Chang, and Sergios Theodoridis. Adaptive kernel design for bayesian optimization is a piece of cake with llms. In *Advances in neural information processing systems*, 2025.

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 *NeurIPS 2020 Competition and Demonstration Track*, pp. 3–26. PMLR, 2021.

Xingchen Wan, Vu Nguyen, Huong Ha, Binxin Ru, Cong Lu, and Michael A Osborne. Think global and act local: Bayesian optimisation over high-dimensional categorical and mixed search spaces. *ICML 38*, 2021.

Linnan Wang, Rodrigo Fonseca, and Yuandong Tian. Learning Search Space Partition for Black-box Optimization using Monte Carlo Tree Search. In *NeurIPS*, volume 33, 2020.

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. *Journal of Machine Learning Research*, 25(212):1–83, 2024. URL `http://jmlr.org/papers/v25/23-0269.html`.

Ziyu Wang, Frank Hutter, Masrour Zoghi, David Matheson, and Nando De Feitas. Bayesian optimization in a billion dimensions via random embeddings. *Journal of Artificial Intelligence Research*, 55:361–387, 2016.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

Haotian Xiang and Qin Lu. Adapting llms for high-dimensional bayesian optimization. In *2025 59th Asilomar Conference on Signals, Systems, and Computers*, pp. 1188–1192. IEEE, 2025.

Pengcheng Xu, Yingying Ma, Wencong Lu, Minjie Li, Wenyue Zhao, and Zhilong Dai. Multi-objective optimization in machine learning assisted materials design and discovery. *Journal of Materials Informatics*, 5(2):N–A, 2025.

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*, 2023.

Juliusz Krzysztof Ziomek and Haitham Bou Ammar. Are random decompositions all we need in high dimensional bayesian optimisation? In *International Conference on Machine Learning*, pp. 43347–43368. PMLR, 2023.

A Appendix

A Detailed Background

A.1 Details of Subspace Embedding Methods

Subspace embedding methods define an effective search geometry by assuming the existence of a low-dimensional subspace (*active subspace*) $\mathcal{Z}$ with effective dimensionality $d_{\mathcal{Z}} < d_{\mathcal{X}}$ and a projection mapping $h : \mathcal{X} \to \mathcal{Z}$ such that $\forall \mathbf{x} \in \mathcal{X}$, there exists $\mathbf{z} \in \mathcal{Z}$ with $\mathbf{z} = h(\mathbf{x})$. Therefore, instead of optimizing the high-dimensional function f, we can optimize the low-dimensional function $g : \mathcal{Z} \to \mathbb{R}$ such that $g(h(\mathbf{x})) = f(\mathbf{x})$. In practice, as the active subspace $\mathcal{Z}$ is unknown, subspace embedding methods define the search geometry through a low-dimensional subspace $\mathcal{P} \subset \mathbb{R}^{d_{\mathcal{P}}}$, $d_{\mathcal{P}} < d_{\mathcal{X}}$ and a projection mapping $\phi : \mathcal{P} \to \mathcal{X}$. Optimization is then performed in the low-dimensional subspace $\mathcal{P}$ by maximizing the acquisition function $\mathbf{p} \in \arg\max_{\mathbf{p} \in \mathcal{P}} \alpha(\mathbf{p})$, while the projection ϕ allows reconstruction of the full vector $\mathbf{x} \in \mathcal{X}$ from $\mathbf{p}$ for function evaluation. Constructing $\mathcal{P}$ typically involves identifying a subset of important variables $\mathbf{p}$, i.e., those expected to exert the greatest influence on the objective function. The projection mapping ϕ defines a fill-in strategy $\mathcal{F} : \mathcal{P} \to \mathcal{Q}$ that assigns values to the remaining unimportant variables $\mathbf{q}$, where $\mathcal{Q}$ is the complementary subspace of $\mathcal{P}$ consisting of the unimportant variables $\mathbf{q} \in \mathcal{Q}$. The full high-dimensional vector $\mathbf{x}$ can be reconstructed as $\mathbf{x} = (\mathbf{p}, \mathbf{q}) = (\mathbf{p}, \mathcal{F}(\mathbf{p}))$ for function evaluation.

Existing subspace embedding approaches mainly follow two paradigms. Variable selection methods identify important variables via principled mechanisms. MCTS-VS (Song et al., 2022) employs Monte Carlo Tree Search to identify important variables, while LassoBO (Hoang et al., 2025) estimates importance using the GP lengthscales with LASSO regularization. Their fill-in strategies $\mathcal{F}$ typically rely on random or best-k sampling. In contrast, random linear embedding methods sample important variables randomly while imposing linear mappings between important and unimportant variables. HESBO (Nayebi et al., 2019) fixes the subspace dimensionality, whereas BAxUS (Papenmeier et al., 2022) and Bounce (Papenmeier et al., 2023) adapt it heuristically over iterations. BOIDS (Ngo et al., 2025) employs BAxUS' mapping and further restricts the search to one-dimensional structures. These methods use a linear fill-in strategy, defined via the employed linear mappings. While effective in certain scenarios, both embedding paradigms rely on fixed or heuristic design choices, thus may lack robustness across diverse objective functions. In this work, we address this limitation with an LLM-guided mechanism.

A.2 Details of Trust Region Methods

Trust region methods define a localized search geometry by restricting BO to a smaller local region $\mathcal{R} \subset \mathcal{X}$, referred to as a trust region, within which BO can perform more efficiently. By iteratively refining and relocating these regions, the challenging high-dimensional optimization problem is effectively decomposed into a sequence of local optimization problems, which can be solved more efficiently. Trust region methods provide several advantages. First, limiting optimization to a small region enables the surrogate model to better approximate the objective function (Eriksson et al., 2019). Second, the region can be interpreted as having a high probability of containing the global optimum, thereby focusing exploitation in promising areas (Ngo et al., 2024).

While the trust regions have been defined using various geometric shapes (Wang et al., 2020; Ngo et al., 2024), hyperrectangular regions remain the most common due to their strong empirical performance, as originally introduced in TuRBO (Eriksson et al., 2019). TuRBO centers a hyperrectangular trust region at the incumbent and computes its side lengths using the GP lengthscales scaled by a global size parameter L. The size L is updated based on the optimization progress. Let τ_s

and τ_f denote the success and failure thresholds, respectively. If the number of consecutive success n_s reaches τ_s, the trust region expands by doubling its size $L \leftarrow \min(2L, L_{\max})$ where $L_{\max}$ is a predefined maximum size. If the number of consecutive failure n_f reaches τ_f, the trust region shrink by halving its size $L \leftarrow L/2$. The algorithm restarts when L becomes smaller than a minimum size, i.e., $L < L_{\min}$. This framework has been adopted in many other methods, including SCBO (Eriksson & Poloczek, 2021) for constrained optimization problems, MORBO (Daulton et al., 2022) for multi-objective optimization problems, CASMOPOLITAN (Wan et al., 2021) for categorical and mixed-variable domains, and BAxUS (Papenmeier et al., 2022) for integrating with a subspace embedding mechanism. The trust region methods typically rely on predefined heuristics to control L, which requires tuning across different BO algorithms and diverse objective functions. For example, TuRBO employs a heuristic that scales τ_f linearly with the dimensionality, while CASMOPOLITAN fixes $\tau_f = 40$ and BAxUS proposes to sublinearly scales τ_f for faster shrinkage. In this work, we leverage this hyperrectangular trust region mechanism to define the search geometry and propose an LLM-guided mechanism to dynamically adjust the trust region size L at each iteration.

B Detailed Methodology For SageLLM-BO

B.1 Optimization Indicators For Trust Region LLM

The General Optimization Information $\mathbb{I}_{TR}$ Similar to the Subspace Embedding LLM, we provide the Trust Region LLM with general optimization information $\mathbb{I}_{TR}$ to guide its trust region decisions. Specifically, $\mathbb{I}_{TR}$ reuses the optimization and compact dataset summaries introduced in Sec. 3.1.2. The optimization summary includes the problem dimensionality, optimization goal, and evaluation budget, indicating the overall setting of the task. These indicators help the LLM determine the trust region geometry and update behavior. For example, the remaining budget and observed progress can inform whether the search should favor broader exploration or more localized refinement, and how aggressively the trust region should respond to successes or failures. By using the same compact representation as $\mathbb{I}_{SS}$, $\mathbb{I}_{TR}$ remains scalable with respect to both problem dimensionality and the number of optimization iterations.

The Trust Region History $\mathbb{H}_{TR}$ Next, we provide the LLM with the trust region history $\mathbb{H}_{TR}$, which summarizes the evolution and performance of trust regions explored within the current subspace. Similar to the subspace history $\mathbb{H}_{SS}$ (Sec. 3.1.2), for each trust region $\mathcal{R}_i$, we record its starting iteration t_i, configuration $(L_i, w_{j,i}, \tau_{s,i}, \tau_{f,i})$, duration Δt_i, and improvement in the incumbent objective value $\Delta f_{\text{best},i}$. The configuration captures the search geometry and update behavior, while the duration and improvement together indicate how effectively that configuration guided the search. By linking previous trust region configurations to their observed outcomes, $\mathbb{H}_{TR}$ helps the LLM reason about an appropriate search scope through L and the responsiveness of the trust region through τ_s and τ_f. We maintain $\mathbb{H}_{TR}$ only within the current subspace, since a change in subspace alters the underlying search geometry, making trust region configurations and their observed performance across different subspaces incompatible.

The Dimension Statistics $\mathbb{S}_{TR}$ Finally, we provide the LLM with dimension statistics $\mathbb{S}_{TR}$ to guide the shape of the trust region through the weight ratios w_i. Similar to $\mathbb{S}_{SS}$ (Sec. 3.1.2), we compute three complementary statistics for each dimension i in the original space $\mathcal{X}$: the GP lengthscale l_i, the empirical standard deviation s_i, and the Spearman rank correlation coefficient ξ_i. While $\mathbb{S}_{SS}$ uses these statistics to identify dimensions for subspace construction, $\mathbb{S}_{TR}$ uses them to determine how the overall search range L should be distributed across dimensions. Specifically, l_i reflects surrogate-estimated sensitivity, ξ_i captures the empirical association between each dimension and the objective, and s_i indicates how extensively each dimension has been explored and provides support for interpreting the other indicators. Together, these statistics provide complementary evidence for assigning $\{w_i\}$, allowing the LLM to allocate different search ranges across dimensions according to their observed characteristics rather than relying on a predefined scaling rule.

C SageLLM-BO Pseudocode

We provide the pseudocode of the SageLLM-BO algorithm.

Algorithm 1 SageLLM-BO Algorithm

1: **Input:** Function $f(\cdot)$, budget T, space $\mathcal{X}$
2: **Output:** The optimum $\mathbf{x}^*$
3: Initialize dataset $\mathcal{D}$, trust region $\mathcal{R}(L, \{w_i\}, \tau_s, \tau_f)$, subspace $\mathcal{P}(d_\mathcal{P}, \mathbf{P}, \mathcal{F})$, and success/failure counters n_s, n_f
4: **while** budget T not depleted **do**
5: Build low-dimensional surrogate model $\mathcal{GP}_\mathcal{P}$ from the projected dataset ▷ Appendix D.1
6: Optimize acquisition function $\hat{\mathbf{p}} = \arg\max_{\mathbf{p}\in\mathcal{P},\, \mathbf{p}\in\mathcal{R}} \alpha(\mathbf{p})$
7: Compute unimportant variables $\hat{\mathbf{q}} = \mathcal{F}(\hat{\mathbf{p}})$
8: Evaluate $y = f(\mathbf{x})$, where $\mathbf{x} = (\hat{\mathbf{p}}, \hat{\mathbf{q}})$; update $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\mathbf{x}, y)\}$
9: Update success/failure counters n_s, n_f
10: **if** $n_s \geq \tau_s$ **then**
11: Query the Trust Region LLM to update L and τ_s
12: Reset success/failure counters
13: **else if** $n_f \geq \tau_f$ **then**
14: Query the Trust Region LLM to update L and τ_f
15: Reset success/failure counters
16: **end if**
17: **if** every n_w iterations **then**
18: Query the Trust Region LLM to update the weight ratios $\{w_i\}$
19: **end if**
20: **if** $L < L_{\min}$ **then**
21: Query the Subspace Embedding LLM to update $\mathcal{P}(d_\mathcal{P}, \mathbf{P}, \mathcal{F})$; re-initialize trust region
22: **end if**
23: **end while**
24: Return $\mathbf{x}^* = \arg\min_{\mathbf{x}_i\in\mathcal{D}} \{y_i\}_{i=1}^T$

D Details of Implementation

D.1 SageLLM-BO Implementation

D.1.1 Surrogate Model Settings

For the surrogate model, following Papenmeier et al. (2022); Ngo et al. (2025), we implement the GP via `GpyTorch` (Gardner et al., 2018) and `BoTorch` (Balandat et al., 2020). Specifically, we use a Matérn 5/2 kernel with ARD lengthscales in the interval $[0.005, 10.0]$, and output scale in the interval $[0.05, 20.0]$. The Gaussian likelihood is modeled with a Homoskedastic noise within the interval $[0.0005, 0.2]$. These settings apply for both GPs: full-dimensional $\mathcal{GP}_\mathcal{X}$ and low-dimensional $\mathcal{GP}_\mathcal{P}$. $\mathcal{GP}_\mathcal{X}$ is trained on the full-dimensional observed dataset $\mathcal{D}$, and $\mathcal{GP}_\mathcal{P}$ is trained on the low-dimensional dataset projected from $\mathcal{D}$. The projection depends on the fill-in strategy and is discussed in the next paragraph.

The high-dimensional GP surrogate $\mathcal{GP}_\mathcal{X}$ (for computing the example subspace $\mathbb{C}_{SS}$ and the dimension statistics $\mathbb{S}$) is trained on the high-dimensional observed dataset $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^t$. The low-dimensional GP surrogate $\mathcal{GP}_\mathcal{P}$(for the main BO process - Alg. 1, line 5) is trained in the current low-dimensional subspace $\mathcal{P}(d_\mathcal{P}, \mathbf{P}, \mathcal{F})$. Since the observed dataset $\mathcal{D}$ is high-dimensional, we first project the observed inputs $\mathbf{x}_i$ into $\mathcal{P}$ according to the current fill-in strategy $\mathcal{F}$. For the 'random' and 'best-k' strategies, $\mathcal{P}$ is defined by the important dimensions $\mathbf{P}$, and the low-dimensional representation is obtained by retaining the corresponding coordinates, i.e., $\mathbf{p}_i = \mathbf{x}_i[\mathbf{P}]$. For the 'linear' strategy, following Ngo et al. (2024), we project each observation onto the embedded subspace using the Moore–Penrose pseudoinverse of the embedding matrix $\mathbf{E}$. Specifically, for $\mathbf{x}_i \in \mathcal{X}$, its low-dimensional representation is $\mathbf{p}_i = \mathbf{E}^\dagger \mathbf{x}_i = (\mathbf{E}^\intercal \mathbf{E})^{-1} \mathbf{E}^\intercal \mathbf{x}_i$. Note that following Nayebi et al. (2019); Papenmeier et al. (2022), the embedding matrix $\mathbf{E}$ is constructed from the important variables $\mathbf{P}$ by mapping each important dimension $p_i \in \mathbf{P}$ to approximately $\lfloor d_\mathcal{X}/d_\mathcal{P} \rfloor$ dimensions in the original space $\mathcal{X}$, each with a random sign. In other words, the selected indices $\mathbf{P}$ serve as anchors for the $d_\mathcal{P}$ embedded coordinates, while the remaining original dimensions are assigned to these embedded coordinates through the sparse random embedding matrix $\mathbf{E}$. The resulting projected dataset $\mathcal{D}_\mathcal{P} = \{(\mathbf{p}_i, y_i)\}_{i=1}^t$ is then used to train the low-dimensional GP surrogate.

D.1.2 Acquisition Function Settings

For the acquisition function of the main BO process, we employ Thompson Sampling (TS). To optimize the acquisition function, we first generate $n_{\text{cand}} = 5000$ candidate points in the low-dimensional subspace $\mathcal{P}$. We then draw a sample from the GP posterior at these candidate points and select the candidate with the minimum sampled value as the next low-dimensional point $\hat{\mathbf{p}}$. Formally, $\alpha(\mathbf{p}) = -\tilde{f}(\mathbf{p})$, where $\tilde{f}$ is sampled from the GP trained on the projected observed dataset $\mathcal{D}_{\mathcal{P}}$. The corresponding high-dimensional point $\hat{\mathbf{x}}$ is subsequently constructed by retaining the selected values of the important variables and determining the remaining unimportant variables according to the current fill-in strategy $\mathcal{F}$, as mentioned in Sec. 3.1.1.

To optimize the acquisition function when constructing the example subspaces in $\mathbb{C}_{SS}$, we optimize a Thompson Sampling acquisition function over the full-dimensional search space $\mathcal{X}$, subject to the subspace constraints defined by $\mathcal{P}(d_{\mathcal{P}}, \mathbf{P}, \mathcal{F})$: $\hat{\mathbf{x}} \in \arg\max_{\mathbf{x} \in \mathcal{X}|\mathcal{P}} \alpha_{\mathcal{X}}(\mathbf{x})$,where $\alpha_{\mathcal{X}}(\cdot)$ is constructed from the high-dimensional GP surrogate $\mathcal{GP}_{\mathcal{X}}$. To approximately solve this problem, we first generate $n_{\text{cand}} = 5000$ candidate points $\mathbf{p}_{\text{cand}}$ in the low-dimensional subspace $\mathcal{P}$. These candidates are then mapped back to the full-dimensional space using the fill-in strategy $\mathcal{F}$, yielding the full-dimensional candidates $\mathbf{x}_{\text{cand}}$. We evaluate these candidates using a sample drawn from the posterior of $\mathcal{GP}_{\mathcal{X}}$ and select the candidate with the minimum sampled value as the most promising point for the current example subspace.

D.1.3 Trust Region And Subspace Embedding Settings

For the trust region configuration, we set $L_{\min} = 0.5^7$ and $L_{\max} = 1.6$ as the minimum and maximum allowable trust region side lengths, respectively. While $L_{\max}$ limits the maximum value of L, i.e., $L \leftarrow \min(Lr, L_{\max})$, $L_{\min}$ determines when the current trust region is discarded. We initialize the trust region parameters as $L = 0.8$, $\tau_f = d_{\mathcal{P}}$, $\tau_s = 3$, and $w_i = 1$ for all $i = 1, \ldots, d_{\mathcal{P}}$. Regarding the success counter n_s and failure counter n_f in Alg. 1, a success is determined by whether the newly observed function value $f^{(t)}$ improves upon the current incumbent $f_{\text{best}}^{(t-1)}$ by a small tolerance ε to account for noise. Specifically, if $f^{(t)} < f_{\text{best}}^{(t-1)} - \varepsilon \left| f_{\text{best}}^{(t-1)} \right|$, we update $n_s \leftarrow n_s + 1$ and reset $n_f \leftarrow 0$. Otherwise, we update $n_f \leftarrow n_f + 1$ and reset $n_s \leftarrow 0$. For the embedded subspace, we initialize $d_{\mathcal{P}} = 1$, randomly sample $\mathbf{P}$, and set $\mathcal{F} = \mathcal{F}_{\text{lin}}$. These settings and initialization choices are consistent with those commonly used in TuRBO (Eriksson et al., 2019) and BAxUS (Papenmeier et al., 2022).

D.1.4 LLM Settings

For both subspace embedding LLM and trust region LLM, we use the API from OpenAI GPT-4.1 version 2025-04-14 via the Azure deployment with temperature of 0.0 and other parameters kept as default (Ngo et al., 2026). For the frequency of LLM trust region call to update the weight ratio, we set $n_w = 10$, we provide the the sensitivity analysis in Appendix F.4. For the "best-k" strategy, we set k such that it accounts for the top 5% of the best solution found so far.

D.1.5 Other Settings

We implement SageLLM-BO in Python (version 3.10). We use Miniconda (version 23.3.1) to install Python packages. We provide in the code supplementary a `.yml` file to install the required packages for running our proposed methods.

D.2 Baseline Implementations

We run all baselines in Python using their open-sourced code. When evaluating the benchmark functions, we use the same implementation for fair comparison.

TuRBO (Eriksson et al., 2019). We use default settings from the paper and implementation, including minimum and maximum trust region size of $\{2^{-7}; 1.6\}$, trust region split ratio of 2. We use the open-sourced implementation at `https://github.com/uber-research/TuRBO`.

BAxUS (Papenmeier et al., 2022). We use the default hyperparameter settings, including bin size $b = 3$, budget until input dimension is equal to the maximum evaluation. The budget management is kept by default. The trust region management is kept by default, which is similar to TuRBO. We use the open-sourced implementation at `https://github.com/LeoIV/BAxUS`.

Bounce (Papenmeier et al., 2023). We use the default hyperparameter settings, including bin size $b = 2$, budget until input dimension is equal to the maximum evaluation. The budget management is kept by default, which is tighter than that of BAxUS. We use the open-sourced implementation at `https://github.com/lpapenme/bounce`.

HESBO (Nayebi et al., 2019). We compare with HESBO given two different settings of target dimension $d_{\mathcal{P}} = 10$ and $d_{\mathcal{P}} = 20$. We use the open-sourced implementation at `https://github.com/aminnayebi/HesBO`.

ALEBO (Letham et al., 2020). We compare with ALEBO given two different settings of target dimension $d_{\mathcal{P}} = 10$ and $d_{\mathcal{P}} = 20$. We use the implementation from `ax-platform` at `https://github.com/facebook/Ax`.

CMA-BO (Ngo et al., 2024). We use default settings from the paper and code implementation, including population size λ, initial variance $\sigma_0 = 0.3(u - l)$ and BO optimizer. We use the open-sourced implementation at `https://github.com/LamNgo1/cma-meta-algorithm`.

RDUCB (Ziomek & Ammar, 2023). We use the default settings from the paper and code implementation. We use the open-sourced implementation at `https://github.com/huawei-noah/HEBO/tree/master/RDUCB`.

BOIDS (Ngo et al., 2025). We use the default settings from the paper and code implementation, including the subspace embedding scheme. We use the open-sourced implementation at `https://github.com/LamNgo1/boids`.

LassoBO (Hoang et al., 2025). We use the default settings from the paper and code implementation. We use the open-sourced implementation at `https://github.com/Fsoft-AIC/LassoBO`.

LLAMBO (Liu et al., 2024). We use the default settings from the paper and code implementation. We use the open-sourced implementation at `https://github.com/tennisonliu/LLAMBO`.

LMABO (Ngo et al., 2026). We use the default settings from the paper and code implementation. We use the open-sourced implementation at `https://github.com/giang-n-ngo/lmabo`.

E Details of Benchmark Problems

Synthetic Problems. The three synthetic benchmark problems, *Ackley-100D*, *Rosenbrock-100D* and *Rastrigin-100D*, are the common synthetic functions widely used as optimization benchmark problems (Surjanovic & Bingham, 2013). The *Ackley-100D* problem is based on the Ackley function [1], with 100 variables affecting the function values. We consider the hypercube domain $[-32.768, 32.768]^{100}$. The *Rastrigin-100D* problem is based on the Rastrigin function [2], with 100 variables affecting the function values. We consider the hypercube domain $[-5.12, 5.12]^{100}$. The *Rosenbrock-100D* problem is based on the Rosenbrock function [3], with 100 variables affecting the function values. We consider the hypercube domain $[-2.048, 2.048]^{100}$. These synthetic functions have been widely used in high-dimensional BO research (Wang et al., 2016; Eriksson et al., 2019; Letham et al., 2020; Eriksson & Jankowiak, 2021; Papenmeier et al., 2022; Ngo et al., 2024; Ziomek & Ammar, 2023; Hoang et al., 2025).

[1] `https://www.sfu.ca/~ssurjano/ackley.html`
[2] `https://www.sfu.ca/~ssurjano/rastr.html`
[3] `https://www.sfu.ca/~ssurjano/rosen.html`

HalfCheetah-102D and Pusher-161D. These benchmark tasks are reinforcement learning (RL) problems from the Gymnasium Mujoco suite implemented in the `gym` package (Brockman et al., 2016). The objective is to maximize the cumulative reward induced by a linear policy parameterized by a high-dimensional weight matrix. In the experiments, we minimize the negative reward. Specifically, *HalfCheetah-102D* is a 102D Mujoco RL problem that is created from the `Half-Cheetah-v4 environment`, with a hypercube bound $[-1.0, 1.0]^{102}$ and `num_rollouts=3`. *Pusher-161D* is a 161D Mujoco RL problem that is created from the `Pusher-v4 environment`, with a hypercube bound $[-1.0, 1.0]^{161}$ and `num_rollouts=3`. We use the implementation from Song et al., 2022. These RL tasks have been widely used in many BO works (Wang et al., 2020; Papenmeier et al., 2022; Ngo et al., 2025; Hoang et al., 2025).

LassoDNA-180D. This task is a hyperparameter optimization problem for Weighted LASSO (Least Absolute Shrinkage and Selection Operator) regression, implemented in the `LassoBench` package (Šehić et al., 2022). The goal is to tune a set of hyper-parameters to balance the least-square estimation and the penalty term for sparsity. Specifically, the *LassoDNA-180D* benchmark is a 180D hyper-parameter optimization problem that uses a DNA dataset from a microbiology problem. This benchmark problem has been widely used in many BO works (Papenmeier et al., 2022; Ziomek & Ammar, 2023; Hoang et al., 2025).

F Additional Experimental Results

F.1 Comparison with LLM-based BO method

We further compare SageLLM-BO with state-of-the-art LLM-based BO methods, including LLAMBO (Liu et al., 2024) and LMABO (Ngo et al., 2026), in terms of both optimization performance and LLM inference cost. The full results of LMABO (over 1000 iterations) are included in the main comparison in Fig. 2. For LLAMBO, evaluating the method over the full optimization budget is prohibitively expensive in our high-dimensional settings. We therefore evaluate LLAMBO on all six problems for 50 optimization iterations and report its performance together with its token consumption. As shown in Fig. 4, SageLLM-BO achieves substantially better optimization performance than LLAMBO and LMABO within the first 50 evaluations across all six problems. LLAMBO and LMABO generally make limited progress within this budget, whereas SageLLM-BO rapidly improves the objective value. Together with the full-budget comparison against LMABO in Fig. 2, these results highlight the importance of explicitly addressing the curse of dimensionality when incorporating LLMs into high-dimensional BO.

The computational cost further demonstrates an important scalability limitation of directly applying existing LLM-based BO approaches to high-dimensional problems. As reported in Table 1, LLAMBO requires approximately 93k–210k tokens per optimization iteration across the six problems, resulting in approximately 4.7–10.5 million tokens for only 50 iterations and a cost of 10.17–23.09 USD per experiment (at 50 iterations) with GPT-4.1 model. LMABO considerably reduces this cost, but still requires approximately 942–977 tokens per iteration. In contrast, SageLLM-BO requires only approximately 279–374 tokens per iteration, resulting in an estimated cost of only 0.58–0.80 USD over the full 1,000-evaluation budget. This difference illustrates that LLM-based BO methods can become prohibitively expensive in high dimensions if the interaction with the LLM is not designed to scale with the dimensionality and optimization history. In particular, repeatedly providing a large portion of the observed high-dimensional dataset to the LLM, or requiring the LLM to directly generate complete high-dimensional candidate solutions, can lead to substantial input and output token costs. In contrast, SageLLM-BO uses the LLM to reason over compact optimization statistics and to guide the low-dimensional search geometry, rather than directly processing the full dataset or generating high-dimensional candidates. Consequently, SageLLM-BO simultaneously maintains a substantially lower LLM inference cost and, as demonstrated in Figs. 4 and 2, achieves stronger optimization performance on high-dimensional problems.

F.2 Sensitivity Analysis to the Large Language Model Choice

In Fig. 5, we investigate the sensitivity of SageLLM-BO to the choice of LLM by comparing the default GPT-4.1 with Mistral-Large-3 and Claude-Haiku-4.5 across six benchmark problems. These

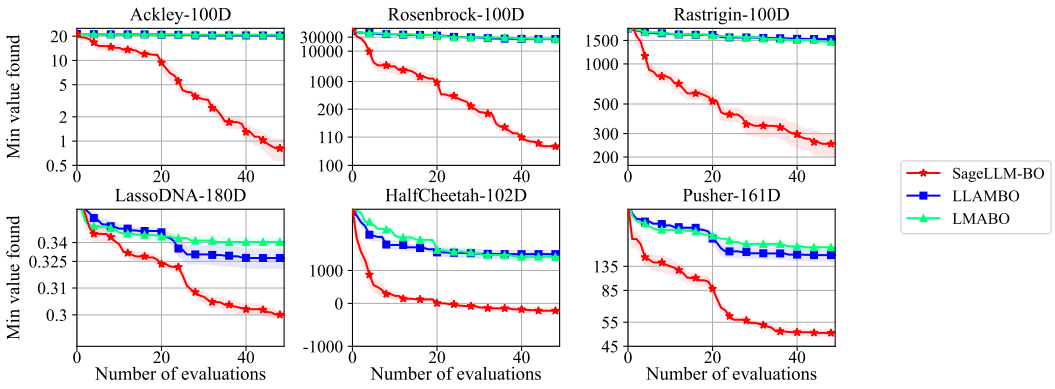


Figure 4: Comparison of SageLLM-BO with the LLM-based BO methods LLAMBO (Liu et al., 2024) and LMABO (Ngo et al., 2026) over the first 50 function evaluations. Due to the high inference cost of LLAMBO, the comparison is limited to 50 evaluations. SageLLM-BO achieves substantially faster optimization progress across all six high-dimensional problems.

Table 1: LLM inference cost comparison. LLAMBO is evaluated for 50 iterations due to its high inference cost, while LMABO and SageLLM-BO are evaluated for 1000 iterations.

Method	Ackley	Rosenbrock	Rastrigin	HalfCheetah	Lasso-DNA	Pusher
Average tokens per iteration						
LLAMBO	111485	109434	93646	123709	209879	195413
LMABO	948	977	976	973	967	942
SageLLM-BO	286	278	286	287	374	332
Total tokens per experiment						
LLAMBO	5574242	5471702	4682302	6185467	10493947	9770672
LMABO	947563	976594	976114	973034	966542	941532
SageLLM-BO	285751	278312	285715	286747	374339	331965
Average cost (USD) per experiment (GPT-4.1)						
LLAMBO	12.23	12.00	10.17	13.62	23.09	21.54
LMABO	2.46	2.51	2.50	2.50	2.50	2.45
SageLLM-BO	0.60	0.58	0.62	0.61	0.80	0.71

model families have also been adopted in prior LLM-based BO studies (Liu et al., 2024; Suwandi et al., 2025; Ngo et al., 2026). Overall, SageLLM-BO exhibits similar optimization behavior across the three LLMs, with no model consistently outperforming the others across all benchmarks. Although some performance differences arise on individual problems, the overall results indicate that the effectiveness of SageLLM-BO is not strongly dependent on a particular LLM. This robustness provides flexibility in selecting the underlying LLM according to practical considerations such as availability, computational cost, and deployment constraints.

F.3 Sensitivity Analysis to the Example Set Granularity

In Fig. 6, we investigate the sensitivity of SageLLM-BO to the parameter m, which controls the granularity used to divide the subspace dimensionality $d_{\mathcal{P}}$ when constructing the example subspace set $\mathbb{C}_{SS}$. We compare the default setting $m = 3$ against $m \in \{2, 4, 5\}$ across six benchmark problems, as shown in Fig. 6. Overall, SageLLM-BO exhibits robustness to the choice of m, with the default setting achieving competitive performance across the benchmarks. While different values of m can lead to better performance on individual problems, such as $m = 5$ on Lasso-DNA and $m = 2$ on Pusher, no alternative setting consistently outperforms $m = 3$ across all problems. These results suggest that SageLLM-BO is reasonably robust to the choice of m and does not require careful tuning of the example-set granularity. Moreover, a smaller m results in fewer examples in $\mathbb{C}_{SS}$ and therefore reduces the number of input tokens required by the LLM. Given the comparable perfor-

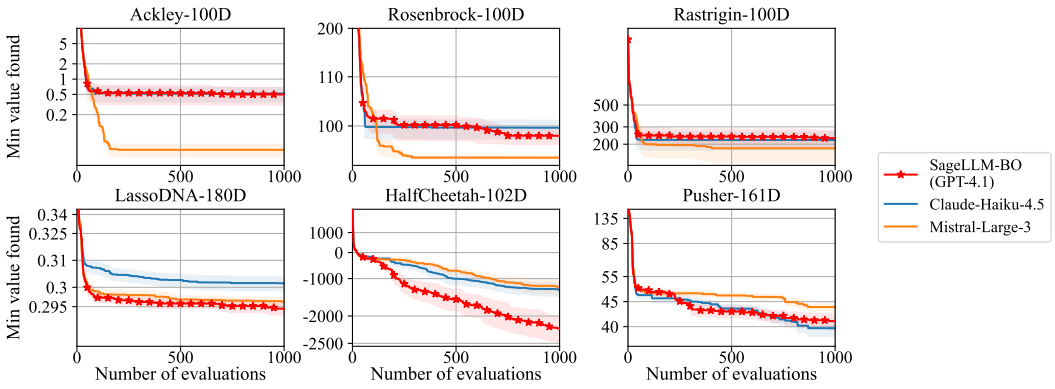


Figure 5: Sensitivity analysis to the choice of LLM (GPT-4.1, Mistral-Large-3, Claude-Haiku-4.5). While the relative performance varies across benchmarks, no LLM consistently outperforms the others, indicating that SageLLM-BO is generally robust to the choice of the underlying LLM.

mance across different values of m, a relatively small m provides a favorable trade-off between the informativeness of the example set and the LLM inference cost.

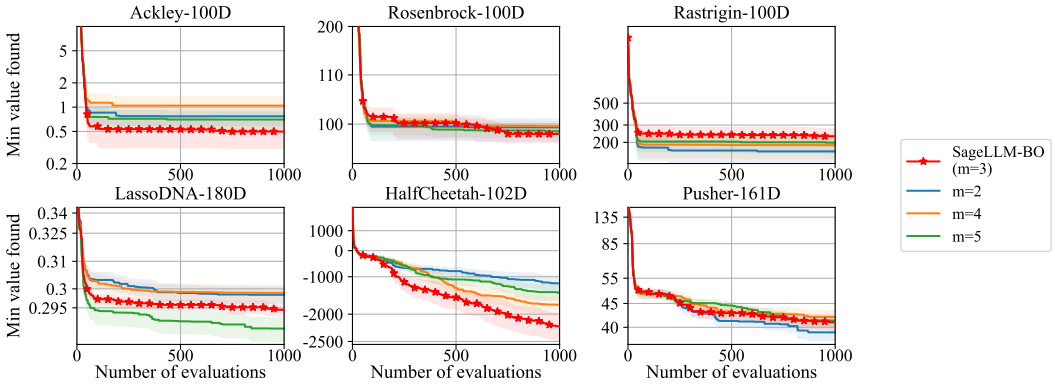


Figure 6: Sensitivity analysis to the example set granularity. Overall, SageLLM-BO is robust to the choice of m, hence a lower value m is recommended to reduce the number of input tokens required by the Subspace Embedding LLM.

F.4 Sensitivity Analysis to the Frequency Of Trust Region Weight Update

In Fig. 7, we investigate the sensitivity of SageLLM-BO to the parameter n_w, which control how often the Trust Region LLM is invoked to update the trust region weight ratio $\{w_i\}$. Reducing n_w will trigger the trust region LLM to ask for the update more, but it will increase the token cost. We compare the default setting $n_w = 10$ against $n_w \in \{5, 20\}$ across six benchmark problems, as shown in Fig. 7. Overall, SageLLM-BO is robust to the choice of n_w, allowing moderately infrequent updates such as $n_w = 10$ to reduce LLM calls without substantially sacrificing performance.

G Computational Complexity

Computing Infrastructure. We run experiments on a computing server with a CPU of type AMD EPYC 9534, 2450 MHz (total of 64 Cores, 128 Threads). Each experiment is allocated 8 CPUs and 32GB Memory, except for ALEBO which was granted up to 64GB Memory. The server is installed with the Operating System Ubuntu 20.04.3 LTS.

Theoretical Analysis. We provide a coarse computational complexity analysis of SageLLM-BO as follows. Let N denote the number of observations, d_x and $d_{\mathcal{P}}$ denote the dimensionalities of

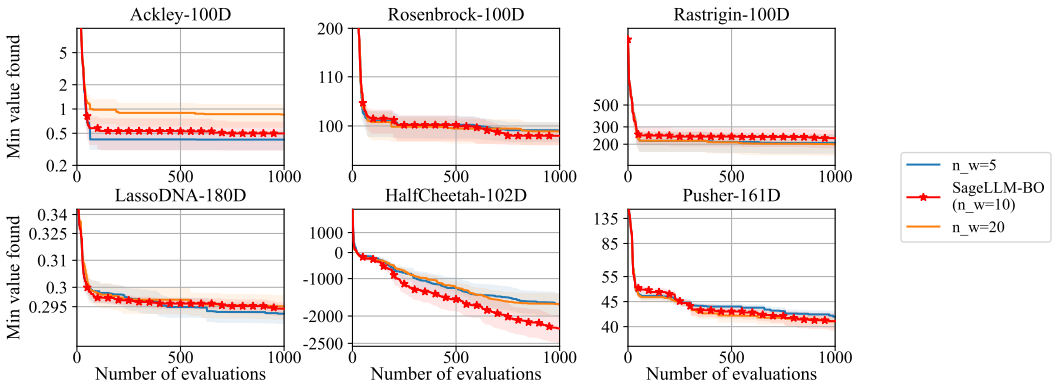


Figure 7: Sensitivity analysis to the frequency of trust region weight update. Overall, SageLLM-BO is robust to the choice of n_w, allowing moderately infrequent updates such as $n_w = 10$ to reduce Trust Region LLM calls without substantially sacrificing performance.

the original high-dimensional search space and the low-dimensional subspace, respectively, n_{cand} denote the number of candidates used for acquisition optimization, and m denote the number of groups used to construct the example subspace set $\mathbb{C}_{SS}$.

At each iteration, SageLLM-BO first projects the observed data into the current low-dimensional subspace and trains a GP surrogate. For the 'linear' fill-in strategy, projecting N high-dimensional observations into a $d_{\mathcal{P}}$-dimensional subspace requires approximately $\mathcal{O}(Nd_x d_{\mathcal{P}})$ operations, while the coordinate-based strategies ('random' and 'best-k') only require selecting the corresponding dimensions. Training a GP on the resulting low-dimensional observations has the standard complexity of $\mathcal{O}(N^3)$. To optimize the acquisition function, Thompson Sampling is performed over n_{cand} candidates in the low-dimensional subspace, resulting in a coarse complexity of $\mathcal{O}(n_{\text{cand}} N d_{\mathcal{P}})$ for evaluating the GP posterior over the candidate set.

For the Subspace Embedding LLM, the dominant additional computational cost arises from constructing the example subspace set $\mathbb{C}_{SS}$, while the computation of $\mathbb{I}_{SS}$, $\mathbf{S}_{SS}$, and $\mathbf{H}_{SS}$ introduces comparatively minor overhead. For each of the m candidate subspace dimensionalities, we consider two important variable selection strategies ('random' and 'lengthscale') and three fill-in strategies ('random', 'best-k' and 'linear'), resulting in $6m$ example subspace configurations. To evaluate all example subspaces' qualities, we train a full-dimensional GP, which incurs a standard GP training cost $\mathcal{O}(N^3)$. Each example subspace then requires n_{cand} candidates for Thompson Sampling acquisition function optimization, hence the coarse complexity is $\mathcal{O}(6mn_{\text{cand}} N d_{\mathcal{X}})$. Since m is kept small in practice, this overhead remains manageable. Importantly, this cost of the Subspace Embedding LLM is incurred only when a new subspace is constructed, rather than at every optimization iteration.

For the Trust Region LLM, the additional conventional computation is comparatively small, since constructing $\mathbb{I}_{TR}$, $\mathbf{S}_{TR}$, and $\mathbf{H}_{TR}$ only involves summarizing the current optimization state and trust region history. Similar to the Subspace Embedding LLM, this overhead is incurred only when a new trust region is required.

Finally, LLM inference introduces additional wall-clock costs that are not captured by the above asymptotic complexities. In an API-based implementation, the observed inference time can also depend on external factors such as API latency and network conditions.

Empirical Runtime. Additionally, we report the empirical runtime (seconds per iteration), excluding the black-box function evaluation time, of SageLLM-BO and all baselines in Table 2. For SageLLM-BO, the reported runtime includes the time spent waiting for LLM responses, which can be affected by external factors such as network conditions and API latency (Liu et al., 2024). Despite this additional overhead, SageLLM-BO maintains a competitive runtime across all six problems, requiring approximately 8–15 seconds per optimization iteration. Its runtime is comparable to several state-of-the-art non-LLM high-dimensional BO methods, such as BAxUS, while being substantially lower than computationally expensive baselines such as ALEBO-20 and RDUCB. These

results indicate that incorporating LLM guidance into SageLLM-BO does not introduce prohibitive wall-clock overhead, despite the additional latency associated with LLM inference.

Table 2: Runtime comparison (seconds per iteration). Despite the cost from LLM (SageLLM-BO full cost), which is affected by internet connectivity and API latency, the overall computational overhead of SageLLM-BO remains comparable to several state-of-the-art non-LLM high-dimensional BO methods. Note that token costs are reported in Table 1.

	Ackley 100D	Rosenbrock 100D	Rastrigin 100D	HalfCheetah 102D	LassoDNA 180D	Pusher 161D
SageLLM-BO	12.36	12.15	12.27	12.24	8.97	8.34
LLAMBO	27.16	26.47	21.48	30.49	51.57	51.88
LMABO	41.26	13.61	17.72	24.72	20.07	15.93
Standard BO	2.20	2.20	2.20	2.36	2.53	2.25
BAxUS	10.14	10.57	10.04	3.72	4.73	12.91
BOIDS	7.97	5.94	6.20	8.98	10.42	10.02
ALEBO-10	36.60	18.12	59.32	74.56	90.14	22.78
ALEBO-20	149.06	53.33	137.61	278.50	269.27	65.07
HESBO-10	4.2e-04	2.4e-04	2.2e-04	4.9e-04	7.1e-04	2.8e-04
HESBO-20	4.7e-04	2.4e-04	2.4e-04	5.1e-04	7.2e-04	2.6e-04
CMA-BO	2.48	2.44	2.44	2.64	3.09	2.76
TuRBO	6.61	6.67	6.65	5.97	7.21	6.85
LassoBO	9.21	7.67	10.06	10.98	14.86	12.86
RDUCB	111.51	36.83	34.67	73.38	33.91	81.11

H EXAMPLES OF PROMPTS FOR THE LLM

In this section, we provide example prompts and their corresponding responses for the subspace embedding LLM and the trust region LLM. These example prompts and responses are taken from the optimization process of the benchmark problem Ackley-100D used in the main text.

For the Subspace Embedding LLM, Figs. 8 and 9 present the prompt provided to the subspace embedding LLM, including the task instructions and optimization indicators. Specifically, "general_information", "previous_subspace_performance", dimension_statistics", and "example_subspace" correspond to $\mathbb{I}_{SS}$, $\mathbb{H}_{SS}$, $\mathbb{S}_{SS}$, and $\mathbb{C}_{SS}$, respectively. Fig. 10 shows the corresponding LLM response, where n_new_dim", "list_new_dim", and "fill_in" specify $d_{\mathcal{P}}$, $\mathbf{P}$, and $\mathcal{F}$, respectively, which together define the new subspace $\mathcal{P}$.

For the Trust Region LLM, Figs. 11, 13, 15 and 17 show the prompts that ask for the side length L, the failure threshold τ_f, the success threshold τ_s, and the weight ratio $\{w_i\}$, respectively. The corresponding LLM responses are shown in Figs. 12, 14, 16, and 18. For the dimension-wise weights, directly asking the LLM to generate a high-dimensional list of floating-point values can be cumbersome and error-prone. Instead, we provide the current GP-derived weights as a reference and ask the LLM to return "adjustments" only for dimensions whose weights should be modified, while the weights of all omitted dimensions remain unchanged.

The Subspace Embedding LLM - Prompt Instruction

You are an expert in Bayesian Optimization, specifically tasked with a subspace embedding mechanism for handling high-dimensional problems. The subspace embedding mechanism enables optimization on low-dimensional subspaces to mitigate the high dimensionality. The low-dimensional subspaces need to be carefully constructed from a list of important dimensions, and a strategy to generate the remaining unimportant dimensions. Your task is to identify a new subspace so as to maximize optimization performance (minimize the objective value).
I will provide the following information for you to analyze the current optimization process.
1. "general_information": The general information including: "objective": minimize or maximize, "D_original": the problem original dimensionality; "D_subspace": the current subspace dimensionality; "n_eval": the number of evaluations so far; "n_eval_remaining": the number of remaining evaluations left; "previous_subspace_performance": the performance of previous subspaces.
2. "dimension_statistics": The statistics of each dimension including: "gp_relevance": A normalized importance score derived from the Gaussian Process lengthscale. Larger values indicate that the GP considers this dimension more influential; "abs_spearman": The absolute Spearman correlation between the dimension and the objective. Larger values indicate a stronger monotonic relationship with the objective; "std": The normalized standard deviation of this dimension among all observations. Smaller values indicate that observed values are concentrated within a narrower range of this dimension.
3. "example_subspace": Several subspaces randomly generated, each contains its properties and expected qualities. The properties include the dimension of the subspace("n_dim"), the list of important dimensions ("list_dim"), and the strategy to estimate the unimportant variables ("fill_in"). We only consider 3 common fill-in strategies: "bestk" fills unimportant dimensions by randomly sampled from the current best solutions (most exploitative), "random" samples unimportant dimensions uniformly at random (most explorative), and "linear" generates unimportant dimensions via a random linear projection from the important dimensions (more balanced). The expected qualities include "acqf_cand": the acquisition value of the best candidate data point x_cand, obtained by optimizing the Thompson Sampling acquisition function; "mean_cand" and "std_cand" present the Gaussian Process posterior mean and standard deviation evaluated at that best candidate x_cand. Use this information to reason and propose a new subspace that improves the optimization performance. First, select "n_new_dim" as an integer specifying the number new K dimensions, K does not have to be one of the example values. Then, select exactly K dimensions to populate the list of dimensions "list_dim". Finally, select a suitable "fill_in" strategy. Generally, increase the subspace dimensionality as the number of evaluations increases. Low-dimensional subspaces are useful early because they reduce optimization difficulty, but they may eventually restrict further improvement. When the current subspace has already consumed a substantial number of evaluations with little or no improvement, avoid another small incremental increase in dimensionality. Instead, prefer a substantially larger K, potentially up to K=100. The stronger the evidence of stagnation and the smaller the remaining evaluation budget, the more aggressively the subspace dimensionality should be increased. If the appropriate increase would bring K close to the original dimensionality, prefer K=100. A large K, especially K=100, reduces or removes the subspace restriction and allows more original dimensions to be optimized jointly when further low-dimensional optimization is unlikely to be effective. Intermediate dimensionalities should mainly be selected when the current subspace is still making meaningful progress or when there is sufficient evaluation budget remaining to justify another subspace stage. Use all signals to reason, and do not rely on any single signal. The output must be in JSON format as follows:
{"n_new_dim": K, "list_new_dim": [d1, d2, ..., dK], "fill_in": "linear"/"random"/"bestk"}
If K=100, then output the following:
{"n_new_dim": 100, "list_new_dim": [], "fill_in": ""}
Do not include any other information.

Figure 8: Example prompt for Subspace Embedding LLM - the instruction.

The Subspace Embedding LLM - Prompt Input

Here are all the information:


```
{
  "general_information": {
    "objective": "minimize",
    "D_original": 100,
    "D_subspace": 2,
    "n_eval": 45,
    "n_eval_remaining": 955,
    "current_fx_range": [
      0.2859,
      21.9682
    ],
    "previous_subspace_performance": [
      {
        "stage_subspace_dim": 2,
        "stage_start_iter": 0,
        "best_fx_improvement_during_stage": 20.0072,
        "fx_during_stage": "range=[0.2859, 21.9682], best=0.2859, mean=11.4598,
std=8.1846",
        "iter_budget_used_in_stage": 45,
        "stage_fill_in": "linear"
      }
    ]
  },
  "dimension_statistics": [
    {
      "dim": 0,
      "gp_relevance": 1.1978,
      "std": 0.0056,
      "abs_spearman": 0.0493
    },
    ...
    {
      "dim": 99,
      "gp_relevance": 1.1978,
      "std": 0.0056,
      "abs_spearman": 0.0493
    }
  ],
  "example_subspace": [
    {
      "n_dim": 26,
      "list_dim": [71, 86, 3, 41, 67, 97, 13, 37, 83, 34, 20, 24, 85, 5, 30, 75, 99, 94, 73, 50,
2, 62, 70, 21, 12, 52],
      "fill_in": "random",
      "acqf_cand": -15.3757,
      "mean_cand": 4.6393,
      "std_cand": 4.4271
    },
    ...
  ]
}
```


Figure 9: Example prompt for Subspace Embedding LLM - the input data.

The Subspace Embedding LLM - Response

{
"n_new_dim": 26,
"list_new_dim": [0, 35, 39, 21, 42, 45, 54, 58, 60, 70, 14, 72, 73, 11, 75, 9, 78, 7, 6, 83, 87, 88, 90, 91, 31, 25],
"fill_in": "bestk"
}

Figure 10: Example response for Subspace Embedding LLM.

The Trust Region LLM - Prompt For Trust Region Side Length L

You are an expert in Bayesian Optimization, specifically tasked with a trust region mechanism for handling high-dimensional problems.
The trust region mechanism is a strategy for handling a high-dimensional Bayesian Optimization problem by creating small local regions (trust regions) to enhance local modelling. The trust region is a hyper-rectangular subspace of the search domain, centered around the current best solution. Its extent is proportional to a size parameter.
Your task is to update the trust region size so that the trust region will help the optimization process achieve smaller function values.
I will provide you with a summary of the current Bayesian Optimization process.
Objective: minimization.
Problem dimensionality: 26.
The total number of evaluation so far (current iteration): 97.
The remaining evaluations: 903.
The objective function values observed so far, including the min=0.2859 (aka the current best solution), max=21.9682, mean=13.2219 and standard_deviation=5.9333.
The distance from the last observed point to all other points, including the min=1.3583 and max=2.8368, mean=1.9863 and standard_deviation=0.4332. The distance from the last observed point to the current best data point is 1.4768. These values characterize the locality of recent search behavior. All distances are computed in the search space $[0,1]^{26}$. For reference, the distance between $[0]^{26}$ to $[1]^{26}$ is 5.0990.
The current trust region size is 0.4800. The history of the size is:

[{
"stage_start_iter": 46,
"tr_size": "0.8000",
"note": "Initialization",
"best_fx_improvement_during_stage": "0.0000",
"stage_duration": 25
}
{
"stage_start_iter": 71,
"tr_size": "0.4800",
"note": "Shrink from 0.8000 with ratio 0.6000",
"best_fx_improvement_during_stage": "0.0000",
"stage_duration": 26
}]

Currently, the optimization process has triggered a shrinkage of trust region. Choose a shrink ratio smaller than 1.0 to be multiplied with the current size 0.4800, where smaller values produce stronger shrinkage (e.g., 0.5 halves the trust region). You must adapt the ratio according to the given optimization state. Your output must a float specifying the ratio. Do not include any other information.

Figure 11: Example prompt for Trust Region LLM - the side length L.

The Trust Region LLM - Response For Trust Region Side Length L

0.6

Figure 12: Example response for Trust Region LLM - the side length L.

The Trust Region LLM - Prompt For Trust Region Failure Threshold τ_f

You are an expert in Bayesian Optimization, specifically tasked with a trust region mechanism for handling high-dimensional problems.
The trust region mechanism is a strategy for handling a high-dimensional Bayesian Optimization problem by creating small local regions (trust regions) to enhance local modelling. The trust region is a hyper-rectangular subspace of the search domain, centered around the current best solution. Its extent is proportional to a size parameter.
Your task is to choose a failure threshold for the current trust region stage so that the trust region will help the optimization process achieve smaller function values. The threshold will remain fixed until the next trust region shrinks or restarts. The threshold counts consecutive unsuccessful Bayesian Optimization iterations, each iteration evaluates 1 data point.
I will provide you with a summary of the current Bayesian Optimization process:
Objective: minimization.
Problem dimensionality: 26.
The total number of evaluation so far (current iteration): 97.
The remaining evaluations: 903.
The objective function values observed so far, including the min=0.2859 (aka the current best solution), max=21.9682, mean=13.2219 and standard_deviation=5.9333.
The distance from the last observed point to all other points, including the min=1.3583 and max=2.8368, mean=1.9863 and standard_deviation=0.4332. The distance from the last observed point to the current best data point is 1.4768. These values characterize the locality of recent search behavior. All distances are computed in the search space $[0,1]^{26}$. For reference, the distance between $[0]^{26}$ to $[1]^{26}$ is 5.0990.
The current trust region size is 0.4800. The history of the size is:

```
[{
   "stage_start_iter": 46,
   "tr_size": "0.8000",
   "note": "Initialization",
   "best_fx_improvement_during_stage": "0.0000",
   "failure_threshold_during_stage": 26,
   "stage_duration": 25
}
{
   "stage_start_iter": 71,
   "tr_size": "0.4800",
   "note": "Shrink from 0.8000 with ratio 0.6000",
   "best_fx_improvement_during_stage": "0.0000",
   "failure_threshold_during_stage": 26,
   "stage_duration": 26
}]
```

Suggest a failure threshold representing the number of consecutive failures before updating the trust region again. The threshold is a positive integer smaller than the remaining evaluations 903. You must adapt the threshold based on the given optimization state. If you want to increase the threshold, consider whether the remaining budget is enough to tolerate consecutive failures. If you want to decrease the threshold, do not change the threshold abruptly since the optimization algorithm requires a number of evaluations to stabilize. Higher dimensional problem requires more evaluations to stabilize. Your output must be an integer specifying the threshold. Do not include any other information.

Figure 13: Example prompt for Trust Region LLM - the failure threshold τ_f.

The Trust Region LLM - Response For Trust Region Failure Threshold τ_f

26

Figure 14: Example response for Trust Region LLM - the failure threshold τ_f.

The Trust Region LLM - Prompt For Trust Region Success Threshold τ_s

You are an expert in Bayesian Optimization, specifically tasked with a trust region mechanism for handling high-dimensional problems.
The trust region mechanism is a strategy for handling a high-dimensional Bayesian Optimization problem by creating small local regions (trust regions) to enhance local modelling. The trust region is a hyper-rectangular subspace of the search domain, centered around the current best solution. Its extent is proportional to a size parameter.
Your task is to choose a success threshold for the current trust region stage so that the trust region will help the optimization process achieve smaller function values. The threshold will remain fixed until the next trust region resizes or restarts. The threshold counts consecutive successful Bayesian Optimization iterations, each iteration evaluates 1 data point.
I will provide you with a summary of the current Bayesian Optimization process:
Objective: minimization.
Problem dimensionality: 2.
The total number of evaluation so far (current iteration): 23.
The remaining evaluations: 977.
The objective function values observed so far, including the min=3.9773 (aka the current best solution), max=21.9682, mean=18.1909 and standard_deviation=5.6875.
The distance from the last observed point to all other points, including the min=0.0496 and max=1.1889, mean=0.7146 and standard_deviation=0.3186. The distance from the last observed point to the current best data point is 0.0000, i.e., the last observed point is currently the best solution found. These values characterize the locality of recent search behavior. All distances are computed in the search space $[0,1]^2$. For reference, the distance between $[0]^2$ to $[1]^2$ is 1.4142.
The current trust region size is 0.8000. The history of the size is:

```
[
  {
    "stage_start_iter": 20,
    "tr_size": "0.8000",
    "note": "Initialization",
    "best_fx_improvement_during_stage": "10.5301",
    "success_threshold_during_stage": 3,
    "stage_duration": 3
  }
]
```

Suggest a success threshold representing the number of consecutive successes before updating the trust region again. The threshold is a positive integer smaller than the remaining evaluations 977. You must adapt the threshold based on the given optimization state.
Your output must be an integer specifying the threshold. Do not include any other information.

Figure 15: Example prompt for Trust Region LLM - the success threshold τ_s.

The Trust Region LLM - Response For Trust Region Success Threshold τ_s

3

Figure 16: Example response for Trust Region LLM - the success threshold τ_s.

The Trust Region LLM - Prompt For Trust Region Weight Ratio $\{w_i\}$

You are an expert in Bayesian Optimization, specifically tasked with a trust region mechanism for handling high-dimensional problems.
The trust region mechanism is a strategy for handling a high-dimensional Bayesian Optimization problem by creating small local regions (trust regions) to enhance local modelling. The trust region is a hyper-rectangular subspace of the search domain, centered around the current best solution. Its extent is proportional to a size parameter L scaled with the weights computed from Gaussian Process lengthscales.
The Gaussian Process (GP) provides the weights for each dimension based on its learned lengthscales. Treat these GP weights as a reference rather than a fixed rule. Your task is to suggest replacement weights when the optimization statistics indicate that the current trust region geometry may be inappropriate. A replacement weight directly replaces the reference GP weight of the corresponding trust region dimension. A replacement weight smaller than the current GP weight narrows that dimension relative to the GP recommendation, while a replacement weight larger than the current GP weight widens it.
For each dimension, I will provide:
- GP normalized weight (gp_weight): The weights computed from the GP lengthscales, governing the relative width of the trust region.
- Coordinate-wise standard deviation (standard_deviation) computed from the previous 98 evaluations. Each input dimension is scaled to [0, 1]. A larger standard deviation suggests that the current dimension is actively being explored, while a smaller value suggests that the search in this dimension remains concentrated.
- Absolute Spearman correlation (spearman_correlation) between the coordinate values and the observed objective values, computed from all 98 previous evaluations: A larger absolute Spearman correlation suggests that changes in this dimension are more strongly associated with changes in the objective, although this evidence may be noisy. Treat them as weak supporting evidence.
Here are the information:
dimension | gp_weight | standard_deviation | spearman_correlation
0 | 0.4029 | 0.3204 | 0.0354
1 | 0.0658 | 0.1030 | 0.0084
2 | 0.2986 | 0.1735 | 0.0422
...
25 | 0.4695 | 0.3116 | 0.0233
Return replacement weights only for dimensions where the evidence strongly supports changing the GP weight. All omitted dimensions will retain their original GP weights.
Your output must only be in JSON format, where the values are replacement weights:
{
  “adjustments”: {“dimension_index”: replacement_weight}
}
If no dimension should be changed, output:
{
  “adjustments”: {}
}

Figure 17: Example prompt for Trust Region LLM - the weight ratios $\{w_i\}$.

The Trust Region LLM - Response For Trust Region Weight Ratio $\{w_i\}$

{“adjustments”: {“1”: 0.5, “2”: 0.5, “14”: 0.5, “17”: 0.5, “20”: 0.5, “21”: 0.5}}

Figure 18: Example response for Trust Region LLM - the weight ratios $\{w_i\}$.