Generate-and-Select: A Simple Framework for Bayesian Optimization with LLM Proposals

Shorya Sharma[1], **Peter I. Frazier**[2], **Raul Astudillo**[1]

[1]Mohamed bin Zayed University of Artificial Intelligence, [2]Cornell University

Abstract

Combining Bayesian optimization (BO) with large language models (LLMs) has emerged as an effective approach for optimization in structured discrete domains. However, many existing methods tightly couple generation and optimization, making it difficult to isolate which components drive performance. We study a simple *generate–and–select* (GaS) framework in which an LLM proposes candidates and a surrogate model selects evaluation points using a standard acquisition rule. This formulation treats the LLM as a black-box proposal mechanism and applies BO over a growing pool of generated candidates without modifying either component. It also provides a simple perspective on hybrid optimization: the generator defines an effective search space, while the acquisition rule governs selection within it. Empirically, GaS matches or outperforms LLM-based optimizers and a representative BO–LLM baseline across semantic search, molecular optimization, symbolic regression, and program synthesis tasks. We further provide a theoretical analysis that formalizes the roles of generation and selection through a regret decomposition. Together, these results suggest that a substantial portion of the gains in hybrid methods arises from how generative models restrict the search space and establish GaS as a strong, simple, and interpretable baseline.

1. Introduction

Recent work combines Bayesian optimization (BO) (Frazier, 2018; Garnett, 2023) with large language models (LLMs) for optimization in structured discrete domains, including molecular design, program synthesis, and language-based search (Agarwal et al., 2025; Gómez-Bombarelli et al., 2018). These approaches aim to leverage the complementary strengths of LLMs as flexible generators of structured candidates and BO as a principled framework for uncertainty-aware decision making.

Existing methods span a spectrum. At one end, LLMs are used as standalone optimizers based on prompting and heuristic search (Brahmachary et al., 2025; Chen et al., 2023; Guo et al., 2024; Lange et al., 2024; Liu et al., 2024b; Yang et al., 2024). At the other end, LLMs are tightly integrated into BO pipelines (Menet et al., 2026; Ranković and Schwaller, 2025). Between these extremes, a growing class of methods combines LLM-based proposal generation with acquisition-based selection (Agarwal et al., 2025; Ballew et al., 2025; Chang et al., 2025; Liu et al., 2024b). Additional discussion of related work is provided in Appendix A.

This work revisits this interaction and asks a simpler question: *how far can we get with a minimal and fully decoupled integration of LLM-based generation and standard BO machinery?*

To answer this question, we consider an iterative *generate–and–select (GaS)* procedure (Figure 1). At each iteration, an LLM proposes a set of candidate solutions, which are accumulated into a

growing pool. A surrogate model then applies an acquisition function over this pool to select points for evaluation. This formulation treats the LLM as a proposal mechanism and BO as a selection strategy, cleanly decoupling generation from selection without modifying either component.

This perspective also provides a useful theoretical lens. Regret naturally decomposes into (i) a *support error* arising from restricting optimization to generator-reachable points, (ii) a *discovery error* capturing how quickly high-quality candidates are proposed, and (iii) a *selection error* corresponding to BO over the candidate pool. We develop this analysis in Appendix C.

Empirically, GaS is highly competitive. Across semantic search, molecular optimization, symbolic regression, and program synthesis tasks, it consistently matches or outperforms LLM-based optimizers and a more complex BO–LLM baseline. Together with the theoretical analysis, these results suggest that a substantial portion of the benefit of hybrid methods arises from how generative models define the effective search space. Thus, while tighter integration may provide additional gains in some settings, a decoupled formulation already captures much of the benefit.

More broadly, our findings highlight the importance of clear and modular baselines when combining generative models with BO. As the space of hybrid methods grows, such baselines are essential for understanding which components drive performance and for enabling fair comparisons.

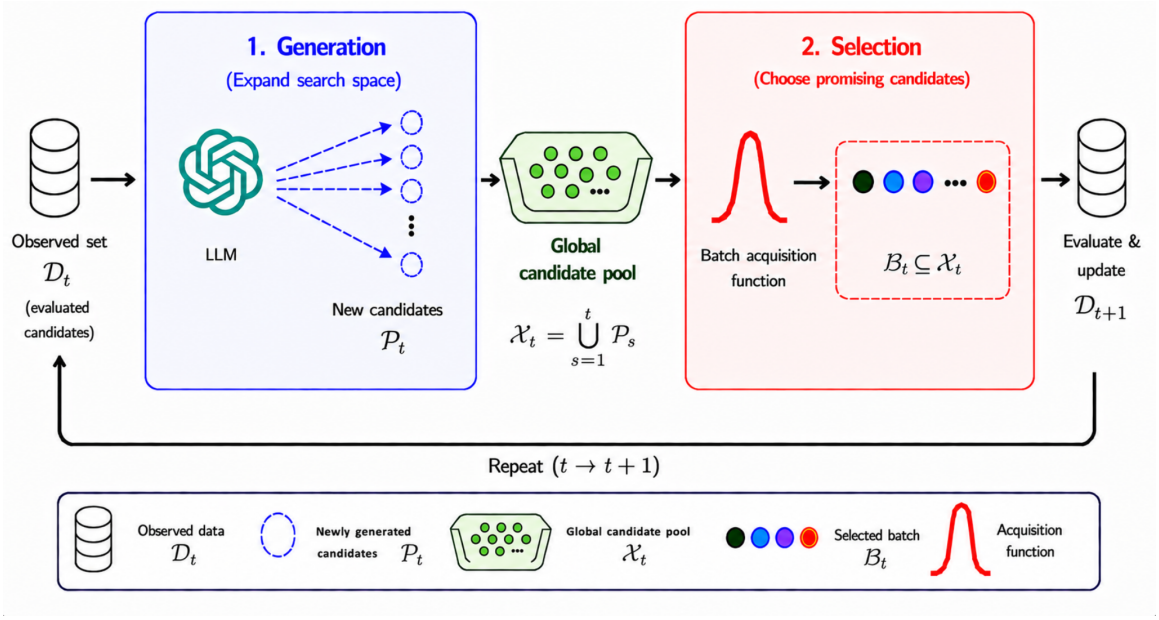


Figure 1: Overview of the GaS framework. At each iteration, an LLM proposes candidate solutions conditioned on the current context, which are added to a cumulative pool. An acquisition function, defined by a surrogate model fit to all previously observed data, is then applied over this pool to select a subset of candidates for evaluation. The resulting observations are incorporated into the dataset, which in turn updates the context used by the LLM in the next iteration.

Contributions. Our main contributions are as follows:

- We introduce a simple generate–and–select formulation for combining BO with LLMs, in which LLMs act as proposal mechanisms and standard acquisition functions are used for selection over a cumulative pool of candidates. This formulation cleanly separates generation from selection and does not require modifying either component (Section 3).
- We demonstrate empirically on tasks including semantic search and molecular optimization that this approach consistently matches or outperforms more complex integration strategies, establishing it as a strong and easily reproducible baseline for comparison (Section 4).

- We provide a theoretical analysis that decomposes regret into support, discovery, and selection terms, offering a simple lens for understanding the roles of generation and optimization in GaS (Appendix C).

2. Problem setting

Optimization objective. We consider the problem of optimizing an expensive function $f : \mathcal{X} \to \mathbb{R}$, where $\mathcal{X}$ is a discrete structured domain, such as molecules or programs. Evaluations of $f(x)$ are costly, and the goal is to identify high-performing inputs under a limited budget. At iteration t, we maintain a dataset $\mathcal{D}_t = \{(x_i, y_i)\}_{i=1}^{n_t}, \quad y_i = f(x_i) + \varepsilon_i$. Based on $\mathcal{D}_t$, we select a batch of q new points to evaluate in order to maximize the best observed value of f within the budget.

Challenges of structured domains. In many applications, the search space $\mathcal{X}$ is extremely large and poorly structured, making direct optimization impractical. For example, in molecular or protein design, most sequences are invalid or biologically implausible, so evaluating arbitrary candidates is inefficient and unlikely to yield useful information. Similar challenges arise in other structured domains, where only a small fraction of candidates correspond to meaningful or high-quality solutions.

Generative model. To address this, we assume access to a generative model $\mathcal{G}$, implemented as an LLM, which induces a conditional distribution over candidate solutions, $p_{\mathcal{G}}(\cdot \mid \mathcal{C}_t)$, where $\mathcal{C}_t$ is a context constructed from the current dataset $\mathcal{D}_t$. This allows the generator to bias search toward valid and promising regions based on prior knowledge and observed data.

Surrogate model and acquisition rule. To guide selection, we assume access to an embedding function $\phi : \mathcal{X} \to \mathbb{R}^d$ and a probabilistic surrogate model $p(f \mid \mathcal{D}_t)$, such as a Gaussian process defined over the embedding space. The surrogate induces an acquisition rule $\pi_t(\cdot \mid \mathcal{D}_t)$, a distribution over batches of q evaluation points, from which a batch is sampled as $\mathcal{B}_t \sim \pi_t(\cdot \mid \mathcal{D}_t)$. This formulation captures both Thompson sampling Kandasamy et al. (2018); Russo and Van Roy (2014) and classical acquisition function maximization, the latter corresponding to a degenerate distribution concentrated on a single batch.

Key problem. The central challenge is to combine candidate generation and acquisition-based selection to enable efficient and reliable search under a limited evaluation budget.

3. Method: Generate–and–Select

We propose a *generate–and–select (GaS)* framework for combining LLMs with BO. This method alternates between using the LLM as a proposal mechanism to generate candidate solutions and using a surrogate model to select which candidates to evaluate via the acquisition rule.

3.1. Overview

At iteration t, we are given a dataset $\mathcal{D}_t = \{(x_i, y_i)\}_{i=1}^{n_t}$. The method proceeds in two steps:

1. **Proposal generation.** Use the LLM to generate a set of r candidate solutions conditioned on a context $\mathcal{C}_t$.
2. **Acquisition-based selection.** Apply the acquisition rule $\pi_t(\cdot \mid \mathcal{D}_t)$ over the cumulative pool of generated candidates and select a batch of q points for evaluation.

The selected candidates are evaluated using the objective f, and the dataset is updated accordingly.

3.2. Proposal generation

The LLM defines a conditional distribution over candidate solutions of the form $p_{\mathcal{G}}(\cdot \mid \mathcal{C}_t)$, where the context $\mathcal{C}_t$ conditions generation. In general, $\mathcal{C}_t$ may combine task-specific prior information (e.g., problem descriptions or domain knowledge) with information from the current dataset $\mathcal{D}_t$. In our implementation, $\mathcal{C}_t$ consists of a task description together with the k previously evaluated inputs with the largest observed values in $\mathcal{D}_t$. These inputs are ordered from worst to best according to their observed values y_i, but the numerical values themselves are not revealed.

Given this context, we sample a set of r candidates $\mathcal{P}_t = \{x_j\}_{j=1}^{r} \sim p_{\mathcal{G}}(\cdot \mid \mathcal{C}_t)$, which forms a proposal set biased toward regions of the search space that are both plausible under the generator and informed by prior observations. We maintain a cumulative pool of generated candidates $\mathcal{X}_t := \bigcup_{s=1}^{t} \mathcal{P}_s$, which aggregates all proposals generated up to iteration t. This allows the algorithm to reuse previously generated candidates rather than discarding them after a single iteration.

3.3. Acquisition-based selection

At iteration t, the acquisition rule $\pi_t(\cdot \mid \mathcal{D}_t)$ is applied over the cumulative pool $\mathcal{X}_t$, defining a distribution over size-q subsets of candidate points. A batch is then sampled as $\mathcal{B}_t \sim \pi_t(\cdot \mid \mathcal{D}_t)$. The selected batch is evaluated using the true objective, and the dataset is updated:

$$\mathcal{D}_{t+1} = \mathcal{D}_t \cup \{(x, y) : y = f(x) + \varepsilon,\ x \in \mathcal{B}_t\}.$$

Instantiations. We instantiate the acquisition rule π_t using both acquisition-based optimization and posterior sampling. Specifically, we consider q-noisy expected improvement (qNEI) (Letham et al., 2019) and Thompson sampling (qTS) (Kandasamy et al., 2018). While both methods are standard in BO, we include detailed definitions and implementation details in Appendix B.

4. Experiments

We evaluate Generate-and-Select (GaS) on four discrete optimization tasks spanning natural language, symbolic reasoning, and structured search: semantic search (Semantle), molecular optimization (MolOpt), program synthesis (1D-ARC), and symbolic regression (SymReg). Across tasks, we measure sample efficiency under fixed objective evaluation budgets.

4.1. Experimental setup

Overview. All methods use the same generator, initial datasets, and evaluation budgets unless otherwise specified. At each iteration, the surrogate selects a batch of q candidates for evaluation. The generator proposes r candidates, and GaS(AF, r) denotes acquisition rule AF and generation size r. Results are averaged over 5 seeds and reported with standard error. Across tasks, variance is low and the performance gap is consistent across seeds, indicating that the gains are robust.

Generator. We use `Mistral-large-2407` via AWS Bedrock for all methods, with prompts consisting of a task description and method-specific exemplars. Additional generator variants are evaluated in Appendix D.2. The specific prompts used for each task are provided in Appendix D.1.4.

Surrogate. For BO-based methods, we fit a Gaussian process (GP) with a Matérn 5/2 kernel over task-specific embeddings $\phi(x)$, with hyperparameters optimized via marginal log-likelihood.

Acquisition rule. GaS uses either qTS or qNEI. Baselines use their corresponding selection mechanisms. Further implementation details are provided in Appendix D.1.

4.2. Tasks

We evaluate on four tasks, three of which (Semantle, MolOpt, and 1D-ARC) follow Agarwal et al. (2025). Performance is measured as the best objective value over time. Additional details are provided in Appendix D.1.

- **Semantle.** A word-search task where the objective returns cosine similarity to a hidden target word; a run is considered solved when the target is evaluated. We average across 10 different targets under a budget of 400 evaluations and batch size $q = 10$.
- **MolOpt.** Molecular optimization on Dockstring, combining drug-likeness and binding affinity. We evaluate 10 protein targets with 200 evaluations each and batch size $q = 5$.
- **1D-ARC.** A program synthesis task where candidates are programs evaluated on training examples. We evaluate 15 tasks with 200 evaluations each and batch size $q = 5$.
- **SymReg.** Symbolic regression with negative mean squared error as the objective. We evaluate 20 target expressions with 200 evaluations each and batch size $q = 10$.

4.3. Baselines

We compare against three baselines representing different approaches to LLM-based optimization.

- **LLM-ES.** An evolutionary search method in which the LLM generates candidate mutations and selection is performed via tournament selection (Menet et al., 2026; Romera-Paredes et al., 2024).
- **OPRO.** A prompt-based optimizer that conditions generation on high-scoring candidates and iteratively refines solutions via in-context learning (Yang et al., 2024).
- **BOPRO.** A latent Bayesian optimization method that fits a Gaussian process in embedding space and proposes candidates by optimizing an acquisition function in this space followed by decoding to the original domain (Agarwal et al., 2025).

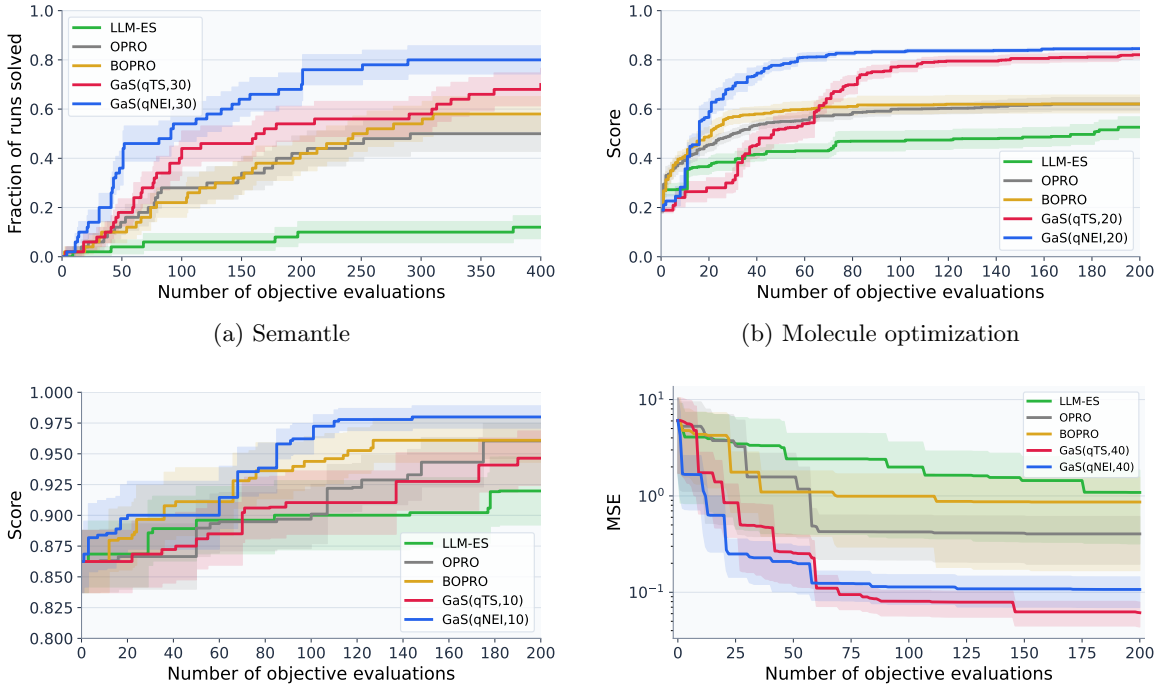
LLM-ES and OPRO fall into the category of LLMs as standalone optimizers, while BOPRO belongs to the class of methods that use LLMs as proposal mechanisms within BO. We do not include more tightly integrated approaches, such as ToSFiT (Menet et al., 2026), which require internal access to the LLM and are prohibitively expensive at the scale (123B parameters) used in our experiments.

4.4. Main results

Figure 2 shows that GaS matches or outperforms the strongest baselines across the evaluated tasks, achieving both faster improvement and stronger final performance. These gains hold across diverse domains, indicating that the approach generalizes beyond specific task structures.

Compared to LLM-based optimizers (OPRO and LLM-ES), GaS yields substantial improvements in sample efficiency, highlighting the importance of uncertainty-aware selection beyond heuristic prompting. Compared to BOPRO, which tightly integrates LLMs within a latent BO framework, GaS achieves stronger performance despite its simpler, fully decoupled design. We hypothesize that this is due to the mismatch between latent-space optimization and discrete decoding, which can lead to inefficient exploration compared to directly selecting among generated candidates.

Across tasks, GaS(qNEI) consistently performs best, with GaS(qTS) also outperforming baselines. On Semantle, GaS(qNEI) solves $\sim$80% of instances versus $\sim$58% for BOPRO and $\sim$50% for OPRO. On MolOpt, GaS reaches $\sim$0.84 versus $\sim$0.6 for baselines. On 1D-ARC, GaS(qNEI) reaches $\sim$0.98 compared to $\sim$0.96 for baselines. On SymReg (log scale), GaS reduces error to $\sim 10^{-1}$, while baselines plateau at significantly higher values.



(a) Semantle

(b) Molecule optimization

(c) Program synthesis

(d) Symbolic regression

Figure 2: Performance across tasks. Curves show mean best-so-far values with standard error. GaS achieves faster improvement and stronger final performance overall. Higher is better except for symbolic regression (log MSE). GaS(AF, r) denotes acquisition function AF and generation size r.

We ablate key design choices of GaS (full results in Appendix D.2). The results identify three main contributors to performance: high-quality candidate generation, iterative pool refreshment, and acquisition-based selection. Adaptive generation substantially outperforms fixed-pool optimization, highlighting the importance of continually updating the search space. Increasing the number of generated candidates further improves performance, albeit with diminishing returns, while acquisition-based selection provides additional gains beyond generation alone.

5. Conclusion

We introduced Generate-and-Select (GaS), a simple framework for combining BO with LLMs in which an LLM proposes candidate solutions and a surrogate model selects evaluation points via a standard acquisition rule. We also provided a theoretical perspective that decomposes regret into support, discovery, and selection terms, clarifying the distinct roles of generation and optimization.

Across various discrete optimization tasks, GaS consistently matches or outperforms more complex approaches. Together, the empirical and theoretical results suggest that much of the benefit of BO–LLM methods arises from how generative models define the effective search space, with BO serving as an effective mechanism for allocating evaluations within it.

More broadly, our findings highlight the value of simple, modular baselines for understanding and improving the interaction between generative models and Bayesian optimization. Future work may explore tighter integration between generation and selection, as well as methods for improving generator coverage and exploration.

References

Dhruv Agarwal, Manoj Ghuhan Arivazhagan, Rajarshi Das, Sandesh Swamy, Sopan Khosla, and Rashmi Gangadharaiah. Searching for optimal solutions with LLMs via Bayesian optimization. In *International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=aVfDrl7xDV.

Matthew Ballew et al. LLM-based Bayesian optimization for prompt search. *arXiv preprint*, 2025.

Shuvayan Brahmachary, Subodh M. Joshi, Aniruddha Panda, Kaushik Koneripalli, Arun Kumar Sagotra, Harshil Patel, Ankush Sharma, Ameya D. Jagtap, and Kaushic Kalyanaraman. Large language model-based evolutionary optimizer: Reasoning with elitism. *Neurocomputing*, 622:129272, 2025. doi: https://doi.org/10.1016/j.neucom.2024.129272. URL https://www.sciencedirect.com/science/article/pii/S0925231224020435.

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

Angelica Chen, David Dohan, and David So. Evoprompting: Language models for code-level neural architecture search. *Advances in Neural Information Processing Systems*, 36:7787–7817, 2023.

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

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

Rafael Gómez-Bombarelli, Jennifer N. Wei, David Duvenaud, José Miguel Hernández-Lobato, Benjamín Sánchez-Lengeling, Dennis Sheberla, Jorge Aguilera-Iparraguirre, Timothy D. Hirzel, Ryan P. Adams, and Alán Aspuru-Guzik. Automatic chemical design using a data-driven continuous representation of molecules. *ACS Central Science*, 4(2):268–276, 2018.

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 *International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=ZG3RaNIsO8.

Kirthevasan Kandasamy, Akshay Krishnamurthy, Jeff Schneider, and Barnabás Póczos. Parallelised Bayesian optimisation via thompson sampling. In *International Conference on Artificial Intelligence and Statistics*, pages 133–142, 2018.

Robert Lange, Yingtao Tian, and Yujin Tang. Large language models as evolution strategies. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 579–582, 2024. doi: 10.1145/3638530.3654238.

Benjamin Letham, Brian Karrer, Guilherme Ottoni, and Eytan Bakshy. Constrained Bayesian optimization with noisy experiments. *Bayesian Analysis*, 14(2):495–519, 2019.

Shengcai Liu, Caishun Chen, Xinghua Qu, Ke Tang, and Yew-Soon Ong. Large language models as evolutionary optimizers. In *IEEE Congress on Evolutionary Computation*, pages 1–8, 2024a.

Tennison Liu, Nicolás Astorga, Nabeel Seedat, and Mihaela van der Schaar. Large language models to enhance Bayesian optimization. In *International Conference on Learning Representations*, 2024b. URL https://openreview.net/forum?id=OOxotBmGol.

Natalie Maus, Haydn Jones, Juston Moore, Matt J Kusner, John Bradshaw, and Jacob Gardner. Local latent space Bayesian optimization over structured inputs. *Advances in Neural Information Processing Systems*, 35:34505–34518, 2022.

Nicolas Menet, Aleksandar Terzic, Michael Hersche, Andreas Krause, and Abbas Rahimi. Thompson sampling via fine-tuning of LLMs. In *International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=u6fc354gfS.

Henry Moss, David Leslie, Daniel Beck, Javier Gonzalez, and Paul Rayson. BOSS: Bayesian optimization over string spaces. *Advances in Neural Information Processing Systems*, 33:15476–15486, 2020.

Changyong Oh, Jakub Tomczak, Efstratios Gavves, and Max Welling. Combinatorial Bayesian optimization using the graph cartesian product. *Advances in Neural Information Processing Systems*, 32, 2019.

Bojana Ranković and Philippe Schwaller. Gollum: Gaussian process optimized LLMs—Reframing LLM finetuning through Bayesian optimization. In *ICLR 2025 Workshop on World Models*, 2025.

Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M Pawan Kumar, Emilien Dupont, Francisco JR Ruiz, Jordan S Ellenberg, Pengming Wang, Omar Fawzi, et al. Mathematical discoveries from program search with large language models. *Nature*, 625(7995):468–475, 2024.

Daniel Russo and Benjamin Van Roy. Learning to optimize via posterior sampling. *Mathematics of Operations Research*, 39(4):1221–1243, 2014.

James Wilson, Frank Hutter, and Marc Deisenroth. Maximizing acquisition functions for Bayesian optimization. *Advances in Neural Information Processing Systems*, 31, 2018.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V. Le, Denny Zhou, and Xinyun Chen. Large language models as optimizers. In *International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=Bb4VGOWELI.

A. Related work

Recent work has explored multiple ways of combining LLMs with BO for search in structured and discrete domains. These approaches differ primarily in the role assigned to the LLM within the optimization pipeline, ranging from standalone optimizers to tightly integrated components of the BO model. Between these extremes lies a growing class of methods that use LLMs as proposal mechanisms, combining generative search with acquisition-based selection. Our approach falls within this intermediate regime, treating the LLM as a proposal distribution and applying standard BO selection over generated candidates. We also briefly discuss prior work on Bayesian optimization in structured domains that does not rely on LLM-based proposal mechanisms.

A.1. LLMs as standalone optimizers

A line of work treats LLMs themselves as optimization algorithms, without explicit surrogate modeling or uncertainty quantification. These methods iteratively propose improved solutions based on previously evaluated examples using in-context learning Yang et al. (2024). Related work explores evolutionary formulations in which LLMs act as mutation and crossover operators in population-based search Brahmachary et al. (2025); Lange et al. (2024); Liu et al. (2024a). These approaches rely on heuristic search and do not explicitly model uncertainty, which can limit sample efficiency and lead to suboptimal search behavior in complex domains. In contrast, our approach augments LLM-based generation with a surrogate model and acquisition function, enabling uncertainty-aware and systematically guided optimization.

A.2. LLMs as proposal mechanisms for BO

A natural extension of LLM-based search is to combine generation with principled selection mechanisms from BO. In this setting, LLMs are used to produce candidate solutions, while a surrogate model and acquisition function guide which candidates to evaluate. Prior work has explored

using LLMs to propose promising regions of the search space and to warm-start BO Liu et al. (2024b), as well as iteratively refining LLM-generated solutions through BO-driven feedback Agarwal et al. (2025). Related approaches combine LLM-driven exploration with surrogate-based exploitation Chang et al. (2025), and recent work specialized in prompt optimization follows a similar generate–then–select structure Ballew et al. (2025).

Among these, Agarwal et al. (2025) is most closely related to our setting. This method performs BO in a continuous latent space and uses the resulting acquisition-maximizing latent vector to guide generation by constructing prompts from previously observed solutions that are nearest to this latent target. This results in a coupled procedure in which acquisition optimization directly shapes the generation process. In contrast, our approach treats the LLM as an explicit proposal mechanism that generates candidates in the original input space and applies a standard acquisition function to select evaluation points from this set, avoiding latent-space optimization and task-specific decoding strategies.

A.3. LLMs integrated into the BO model

At the most tightly coupled end of this spectrum, LLMs are incorporated directly into the BO machinery, either as part of the surrogate model or as a learned decision policy. Prior work constructs Gaussian process models over representations derived from LLM embeddings Ranković and Schwaller (2025), while other approaches parameterize BO policies using language models Menet et al. (2026). These methods can be effective but typically require access to model internals or fine-tuning. In contrast, our approach treats the LLM as a black-box generator and uses standard BO components without modification.

A.4. Bayesian optimization over structured domains without LLM-based proposals

A substantial body of work studies Bayesian optimization in structured domains such as graphs, sequences, programs, and molecular representations without relying on LLM-based proposal mechanisms. Prior approaches either define surrogate models directly over discrete structures (Moss et al., 2020; Oh et al., 2019), or map structured objects into continuous latent or embedding spaces (Gómez-Bombarelli et al., 2018; Maus et al., 2022). A key challenge in these methods is that optimization occurs in a continuous or embedded space and requires a decoding step to obtain valid structured objects, which can introduce errors. In contrast, our approach generates candidates directly in the original domain and restricts optimization to a dynamic candidate set.

B. Acquisition Rules

In this section, we provide detailed definitions and implementations of the acquisition rules used in our experiments. Recall that at iteration t, the surrogate model induces an acquisition rule

$$\pi_t(\cdot \mid \mathcal{D}_t),$$

which selects a batch $\mathcal{B}_t \subseteq \mathcal{X}_t$ of size q based on the current dataset. We consider two instantiations: q-noisy expected improvement (qNEI) and batch Thompson sampling (qTS).

B.1. Batch noisy expected improvement (qNEI)

In the presence of observation noise, expected improvement is defined relative to the posterior distribution over latent function values at previously evaluated points. Let $\mathbf{f}_t = (f(x_i))_{i=1}^{n_t}$ denote the latent function values corresponding to the dataset $\mathcal{D}_t$, and let $\mathbf{f}_\mathcal{B} = (f(x))_{x\in\mathcal{B}}$ denote the latent values at a candidate batch $\mathcal{B}$.

The q-noisy expected improvement is defined as

$$\alpha_{\mathrm{qNEI}}(\mathcal{B};\mathcal{D}_t) := \mathbb{E}\left[\max\Big(\max_{x\in\mathcal{B}} f(x) - \max_{i\le n_t} f(x_i),\, 0\Big) \;\middle|\; \mathcal{D}_t\right],$$

where the expectation is taken with respect to the joint posterior over $\mathbf{f}_t$ and $\mathbf{f}_{\mathcal{B}}$.

Batch construction. Selecting the optimal batch requires solving

$$\mathcal{B}_t \in \operatorname*{arg\,max}_{\substack{\mathcal{B}\subseteq\mathcal{X}_t\\ |\mathcal{B}|=q}} \alpha_{\mathrm{qNEI}}(\mathcal{B};\mathcal{D}_t),$$

which is combinatorial. In practice, we approximate this using a greedy procedure: starting from the empty set, we iteratively add the point that maximizes the marginal gain in acquisition value conditioned on previously selected points. This approach is justified by the submodularity of qNEI, which yields a constant-factor approximation guarantee relative to the optimal batch (Wilson et al., 2018).

B.2. Batch Thompson sampling (qTS)

Thompson sampling defines a stochastic acquisition rule based on sampling from the surrogate posterior. At iteration t, we draw q independent sample functions

$$\tilde{f}_1,\ldots,\tilde{f}_q \overset{\text{i.i.d.}}{\sim} p(f \mid \mathcal{D}_t),$$

and use each sampled function to select one point from the candidate pool $\mathcal{X}_t$.

Batch construction. To construct a batch of size q, we use the standard independent-sample batch Thompson sampling procedure:

1. Initialize $\mathcal{B}_t \leftarrow \emptyset$.
2. Draw independent posterior samples
$$\tilde{f}_1,\ldots,\tilde{f}_q \overset{\text{i.i.d.}}{\sim} p(f \mid \mathcal{D}_t).$$
3. For $i = 1,\ldots,q$:
   - Select
   $$x_i \in \operatorname*{arg\,max}_{x\in\mathcal{X}_t\setminus\mathcal{B}_t} \tilde{f}_i(x),$$
   - Update $\mathcal{B}_t \leftarrow \mathcal{B}_t \cup \{x_i\}$.

C. Regret analysis

In this section, we provide a regret analysis of the generate–and–select procedure described in Section 3. The goal is to isolate the contributions of (i) the generator, which proposes candidates, and (ii) the acquisition-based selection rule, which chooses which candidates to evaluate.

C.1. Setup

We consider Bayesian optimization over a finite structured domain $\mathcal{X}$. The environment is parameterized by a latent variable $\theta \sim \mu$, which determines both the objective function f_θ and associated task-side information. We write $\eta = \eta(\theta)$ for the information available to the generator.

The objective is a function

$$f_\theta : \mathcal{X} \to \mathbb{R},$$

which is deterministic conditional on θ.

At each iteration $t = 1, \ldots, T$, the algorithm selects a point $x_t \in \mathcal{X}$ and observes

$$y_t = f_\theta(x_t) + \varepsilon_t.$$

The noise sequence $\{\varepsilon_t\}$ is conditionally independent given θ and the selected points, and is σ-sub-Gaussian, i.e., for all $\lambda \in \mathbb{R}$,

$$\mathbb{E}[\exp(\lambda \varepsilon_t) \mid \mathcal{D}_t, x_t, \theta] \leq \exp\left(\frac{\lambda^2 \sigma^2}{2}\right).$$

Let

$$\mathcal{D}_t = \{(x_i, y_i)\}_{i=1}^{t-1}$$

denote the dataset available prior to iteration t.

Generator model. We assume access to a generative model (LLM) that proposes batches of candidate points. The generator depends on the task instance through the available information $\eta = \eta(\theta)$.

Let $\mathfrak{D}$ denote the space of generator-side data objects, and define a context-construction map

$$\tau : \mathcal{H} \times \mathfrak{D} \to \mathcal{C}.$$

At iteration t, the algorithm forms a context

$$\mathcal{C}_t := \tau(\eta, \mathcal{D}_t).$$

Given this context, the generator produces a batch of candidates

$$\mathcal{P}_t \sim p_{\mathcal{G}}(\cdot \mid \mathcal{C}_t),$$

where $\mathcal{P}_t \subseteq \mathcal{X}$ and $|\mathcal{P}_t| = r$ for some fixed r.

Filtration and independence assumption. Let $\mathcal{F}_t$ denote the sigma-algebra capturing the algorithmic history up to iteration t. Concretely, $\mathcal{F}_t$ is generated by the task-side information η, the generated candidate batches $\mathcal{P}_1, \ldots, \mathcal{P}_t$, the selected points $x_1, \ldots, x_{t-1}$, and the corresponding observations $y_1, \ldots, y_{t-1}$. Thus, $\mathcal{F}_t$ represents all information available immediately before selecting x_t, including the cumulative candidate pool $\mathcal{X}_t$. We assume that the generator randomness is conditionally independent of the latent function values $f_\theta(x)$ at unobserved points, given $\mathcal{F}_t$. In particular, while the generator may depend on task-side information $\eta(\theta)$ and the observed dataset $\mathcal{D}_t$, it does not depend on unobserved values of the objective function beyond what is revealed through $\mathcal{F}_t$.

Generator-reachable support. Since the generator produces batches, its support is defined over subsets of $\mathcal{X}$. For a given task instance η, define the set of reachable batches as

$$\mathcal{S}_{\mathcal{G}}(\eta) := \bigcup_{d \in \mathfrak{D}} \operatorname{supp}\big(p_{\mathcal{G}}(\cdot \mid \tau(\eta, d))\big),$$

where each element of $\mathcal{S}_{\mathcal{G}}(\eta)$ is a subset of $\mathcal{X}$.

We define the generator-reachable support over points as

$$\mathcal{X}_{\mathcal{G}}(\eta) := \bigcup_{S \in \mathcal{S}_{\mathcal{G}}(\eta)} S.$$

That is, $\mathcal{X}_{\mathcal{G}}(\eta)$ consists of all points that can appear in at least one batch generated by the model under some admissible context.

By construction,

$$\mathcal{X}_t \subseteq \mathcal{X}_{\mathcal{G}}(\eta) \qquad \text{for all } t.$$

We assume that $\mathcal{X}_{\mathcal{G}}(\eta)$ is nonempty almost surely.

C.2. Regret decomposition

Let

$$x^*(\theta) \in \arg\max_{x \in \mathcal{X}} f_\theta(x)$$

denote a global maximizer over the full domain, with fixed tie-breaking. The (Bayesian) regret is defined as

$$\operatorname{Regret}(T) := \mathbb{E}\left[\sum_{t=1}^{T} \big(f_\theta(x^*(\theta)) - f_\theta(x_t)\big)\right],$$

where the expectation is taken over $\theta \sim \mu$, the observation noise, and the internal randomness of the algorithm.

The generator induces a restricted search space $\mathcal{X}_{\mathcal{G}}(\eta) \subseteq \mathcal{X}$, consisting of all points that can be proposed under some admissible context. Within this set, we define the best generator-reachable point

$$x^*_{\mathcal{G}}(\theta) \in \arg\max_{x \in \mathcal{X}_{\mathcal{G}}(\eta)} f_\theta(x).$$

At iteration t, the algorithm evaluates points drawn from the cumulative set of generated candidates

$$\mathcal{X}_t = \bigcup_{s=1}^{t} \mathcal{P}_s,$$

which is a random subset satisfying

$$\mathcal{X}_t \subseteq \mathcal{X}_{\mathcal{G}}(\eta).$$

We define the best point discovered up to time t as

$$x^*_t(\theta) \in \arg\max_{x \in \mathcal{X}_t} f_\theta(x).$$

These three points define a nested sequence of increasingly restrictive search spaces,

$$\mathcal{X} \supseteq \mathcal{X}_{\mathcal{G}}(\eta) \supseteq \mathcal{X}_t,$$

with corresponding optima

$$x^*(\theta), \quad x^*_{\mathcal{G}}(\theta), \quad x^*_t(\theta).$$

We now decompose the regret into three terms that correspond to the gaps between these optima.

Proposition 1 (Regret decomposition)**.** *For each iteration t,*

$$f_\theta(x^*(\theta)) - f_\theta(x_t) = \underbrace{f_\theta(x^*(\theta)) - f_\theta(x^*_{\mathcal{G}}(\theta))}_{\text{support error}} + \underbrace{f_\theta(x^*_{\mathcal{G}}(\theta)) - f_\theta(x^*_t(\theta))}_{\text{discovery error}} + \underbrace{f_\theta(x^*_t(\theta)) - f_\theta(x_t)}_{\text{selection error}}.$$

Consequently,

$$\mathrm{Regret}(T) = R_{\mathrm{sup}}(T) + R_{\mathrm{dis}}(T) + R_{\mathrm{sel}}(T),$$

where

$$R_{\mathrm{sup}}(T) := T\,\mathbb{E}\big[f_\theta(x^*(\theta)) - f_\theta(x^*_{\mathcal{G}}(\theta))\big],$$

$$R_{\mathrm{dis}}(T) := \sum_{t=1}^{T} \mathbb{E}\big[f_\theta(x^*_{\mathcal{G}}(\theta)) - f_\theta(x^*_t(\theta))\big],$$

and

$$R_{\mathrm{sel}}(T) := \sum_{t=1}^{T} \mathbb{E}\big[f_\theta(x^*_t(\theta)) - f_\theta(x_t)\big].$$

Proof. Add and subtract $f_\theta(x^*_{\mathcal{G}}(\theta))$ and $f_\theta(x^*_t(\theta))$ in the instantaneous regret:

$$\begin{aligned} f_\theta(x^*(\theta)) - f_\theta(x_t) = {} & f_\theta(x^*(\theta)) - f_\theta(x^*_{\mathcal{G}}(\theta)) \\ & + f_\theta(x^*_{\mathcal{G}}(\theta)) - f_\theta(x^*_t(\theta)) \\ & + f_\theta(x^*_t(\theta)) - f_\theta(x_t). \end{aligned}$$

Summing over $t = 1, \ldots, T$ and taking expectations yields the result. □

On the support error term. The support error $R_{\mathrm{sup}}(T)$ depends on the gap between the global optimum and the best point within the generator-reachable support. We characterize this term via the expected support gap

$$\varepsilon_{\mathrm{sup}} = \mathbb{E}\big[f_\theta(x^*(\theta)) - f_\theta(x^*_{\mathcal{G}}(\theta))\big]\,,$$

which yields the exact relation $R_{\mathrm{sup}}(T) = T\,\varepsilon_{\mathrm{sup}}$. This formulation reflects the average-case quality of the generator-reachable support and accommodates stochastic models in which the optimal point may lie outside the support with small probability. In practice, $\varepsilon_{\mathrm{sup}}$ captures how well the generator concentrates on high-quality regions of the search space, and small values correspond to settings where the support restriction induces only a minor loss relative to the global optimum.

C.3. Bounding the discovery regret

We now analyze the discovery error term, which captures how quickly the generator proposes high-quality points within its reachable support $\mathcal{X}_{\mathcal{G}}(\eta)$.

Recall that the discovery error arises from the gap between the best point reachable by the generator, $x^*_{\mathcal{G}}(\theta)$, and the best point discovered so far, $x^*_t(\theta)$. This term is nonzero until the optimal generator-reachable point is proposed.

The key question is therefore: how long does it take for the generator to propose $x^*_{\mathcal{G}}(\theta)$?

Discovery probability. We assume that the generator assigns a uniformly positive probability to proposing the optimal point within its reachable support.

Assumption 1 (Supported optimum discovery)**.** *There exists $p_{\mathrm{dis}} > 0$ such that for all iterations t,*

$$\mathbb{P}\big(x^*_{\mathcal{G}}(\theta) \in \mathcal{P}_t \mid \mathcal{F}_{t-1}, \theta\big) \geq p_{\mathrm{dis}}$$

almost surely on the event $\{\tau_{\mathcal{G}} \geq t\}$.

Interpretation and implications of the discovery assumption. The discovery assumption requires that the generator maintains nonvanishing probability of proposing high-quality reachable points. In practice, this depends on the interplay between diversity in the generator outputs—induced by the prompt context and sampling—and exploration in the acquisition rule, which shapes the data used to construct that context. When these components are sufficiently exploratory, it is reasonable to expect the discovery probability to remain bounded away from zero. This perspective also suggests that more refined analyses could replace the uniform bound p_{dis} with a data-dependent quantity, potentially yielding sharper, problem-dependent regret guarantees that reflect adaptivity to the evolving context.

Discovery time. Define the discovery time

$$\tau_{\mathcal{G}} := \inf\{t \geq 1 : x^*_{\mathcal{G}}(\theta) \in \mathcal{P}_t\},$$

i.e., the first time at which the optimal generator-reachable point is proposed.

Lemma 2. *Suppose Assumption 1 holds. Then, for every $t \geq 1$,*

$$\mathbb{P}(\tau_{\mathcal{G}} > t) \leq (1 - p_{\text{dis}})^t.$$

Proof. Let

$$E_t = \{x^*_{\mathcal{G}}(\theta) \notin \mathcal{P}_t\}.$$

By the definition of $\tau_{\mathcal{G}}$,

$$\{\tau_{\mathcal{G}} > t\} = E_t \cap \{\tau_{\mathcal{G}} > t-1\}.$$

Therefore,

$$\begin{aligned}\mathbb{P}(\tau_{\mathcal{G}} > t) &= \mathbb{P}(E_t, \tau_{\mathcal{G}} > t-1) \\ &= \mathbb{E}\big[\mathbf{1}_{\{\tau_{\mathcal{G}} > t-1\}} \mathbb{P}(E_t \mid \mathcal{F}_{t-1}, \theta)\big] \\ &\leq (1 - p_{\text{dis}}) \mathbb{P}(\tau_{\mathcal{G}} > t-1).\end{aligned}$$

Iterating the inequality gives

$$\mathbb{P}(\tau_{\mathcal{G}} > t) \leq (1 - p_{\text{dis}})^t.$$

□

Discovery regret bound. We now bound the contribution of discovery to the cumulative regret. Define the reward range within the generator-reachable support:

$$R_{\mathcal{G}}(\theta) := f_\theta(x^*_{\mathcal{G}}(\theta)) - \min_{x \in \mathcal{X}_{\mathcal{G}}(\eta)} f_\theta(x).$$

Proposition 3 (Discovery regret)**.** *Under the discovery condition with parameter $p_{\text{dis}} > 0$,*

$$R_{\text{dis}}(T) \leq \sum_{t=1}^{T} \mathbb{E}\left[R_{\mathcal{G}}(\theta)\, \mathbf{1}\{\tau_{\mathcal{G}} > t\}\right].$$

If $\mathbb{E}[R_{\mathcal{G}}(\theta)] < \infty$, then

$$R_{\text{dis}}(T) \leq \frac{\mathbb{E}[R_{\mathcal{G}}(\theta)]}{p_{\text{dis}}}.$$

Proof. If $x^*_{\mathcal{G}}(\theta) \in \mathcal{X}_t$, then $x^*_t(\theta) = x^*_{\mathcal{G}}(\theta)$, and the discovery error at iteration t is zero. Otherwise, the discovery error is at most $R_{\mathcal{G}}(\theta)$. Therefore,

$$f_\theta(x^*_{\mathcal{G}}(\theta)) - f_\theta(x^*_t(\theta)) \leq R_{\mathcal{G}}(\theta)\, \mathbf{1}\{\tau_{\mathcal{G}} > t\}.$$

Summing over t and taking expectations yields the first bound.

For the second bound, we condition on θ. By Assumption 1,

$$\mathbb{P}(\tau_{\mathcal{G}} > t \mid \theta) \leq (1 - p_{\text{dis}})^t.$$

Thus,

$$\begin{aligned}\mathbb{E}\left[R_{\mathcal{G}}(\theta)\,\mathbf{1}\{\tau_{\mathcal{G}} > t\}\right] &= \mathbb{E}_\theta[R_{\mathcal{G}}(\theta)\mathbb{P}(\tau_{\mathcal{G}} > t \mid \theta)] \\ &\leq \mathbb{E}[R_{\mathcal{G}}(\theta)]\,(1 - p_{\text{dis}})^t.\end{aligned}$$

Summing over $t = 1, \ldots, T$ gives

$$R_{\text{dis}}(T) \leq \mathbb{E}[R_{\mathcal{G}}(\theta)] \sum_{t=1}^{T} (1 - p_{\text{dis}})^t \leq \frac{\mathbb{E}[R_{\mathcal{G}}(\theta)]}{p_{\text{dis}}},$$

where the last inequality uses the bound on the geometric series. □

Interpretation. This result shows that discovery incurs only a constant regret penalty. The key quantity is the discovery probability p_{dis}, which measures how reliably the generator proposes the best point within its reachable support at each iteration. Larger batch sizes or more diverse generation strategies can increase p_{dis}, thereby reducing the discovery error. Importantly, this term does not scale with T: once the optimal point $x^*_{\mathcal{G}}(\theta)$ is generated, it remains in the candidate set and can be repeatedly selected by the acquisition rule.

C.4. Gaussian process specialization

We now specialize the general regret bound to the case where the surrogate model is a Gaussian process and selection is performed via Thompson sampling. This allows us to obtain explicit bounds for both the selection error and the discovery error.

Model. Assume that, conditional on the task-side information η, the objective satisfies

$$f_\theta \mid \eta \sim \mathcal{GP}(m, k),$$

defined on the finite domain $\mathcal{X}$. Observations satisfy

$$y_t = f_\theta(x_t) + \varepsilon_t,$$

with independent Gaussian noise

$$\varepsilon_t \sim \mathcal{N}(0, \sigma_\varepsilon^2), \qquad \sigma_\varepsilon^2 > 0.$$

Since $\mathcal{X}$ is finite, this is equivalent to a multivariate Gaussian prior over $(f_\theta(x))_{x \in \mathcal{X}}$. We also assume

$$k(x, x) \leq \kappa^2 \quad \text{for all } x \in \mathcal{X}.$$

Information gain. For any subset $A \subseteq \mathcal{X}$, define the T-step information gain

$$\Gamma_T(A) := \max_{x_1, \ldots, x_T \in A} I(f_A; y_{1:T}),$$

where $f_A = (f_\theta(x))_{x \in A}$.

Selection error bound. We now show that GP Thompson sampling guarantees extend to selection over the cumulative proposal pool.

Lemma 4 (GP Thompson sampling over predictable time-varying pools)**.** *At round t, draw*

$$\widetilde{f}_t \sim \mathbb{P}(f_\theta \mid \mathcal{F}_t),$$

and select

$$x_t \in \arg\max_{x \in \mathcal{X}_t} \widetilde{f}_t(x).$$

Let

$$x_t^*(\theta) \in \arg\max_{x \in \mathcal{X}_t} f_\theta(x)$$

denote the best point in the current pool. Then, up to logarithmic factors,

$$\mathbb{E}\left[\sum_{t=1}^{T}\big(f_\theta(x_t^*(\theta)) - f_\theta(x_t)\big)\right] \le \widetilde{O}\Big(\mathbb{E}\Big[\sqrt{T\,\Gamma_T(\mathcal{X}_{\mathcal{G}}(\eta))}\Big]\Big).$$

Proof. At each round t, condition on the history $\mathcal{F}_t$. By construction, the candidate pool $\mathcal{X}_t$ is $\mathcal{F}_t$-measurable. Moreover, by the conditional independence assumption, the generated candidates do not reveal unobserved values of the objective function beyond the task-side information η and observed data $\mathcal{D}_t$. Therefore, conditioning on $\mathcal{F}_t$ does not provide additional information about f_θ beyond $(\eta, \mathcal{D}_t)$, and the posterior used by Thompson sampling coincides with the standard GP posterior:

$$\mathbb{P}(f_\theta \mid \mathcal{F}_t) = \mathbb{P}(f_\theta \mid \eta, \mathcal{D}_t).$$

Conditional on $\mathcal{F}_t$, Thompson sampling is applied over the finite action set $\mathcal{X}_t$. Since this set is fixed given $\mathcal{F}_t$, the standard posterior-sampling identity applies: the maximizer of a posterior sample over $\mathcal{X}_t$ has the same conditional distribution as the maximizer of the true function f_θ over $\mathcal{X}_t$. Consequently, x_t and $x_t^*(\theta)$ have the same conditional distribution given $\mathcal{F}_t$. Therefore, the standard Gaussian process Thompson sampling analysis (e.g., Russo and Van Roy (2014)) applied to the finite action set $\mathcal{X}_t$ implies, up to logarithmic factors,

$$\mathbb{E}[f_\theta(x_t^*(\theta)) - f_\theta(x_t) \mid \mathcal{F}_t] \le \widetilde{O}(\mathbb{E}[\sigma_t(x_t) \mid \mathcal{F}_t]),$$

where $\sigma_t(x)$ is the posterior standard deviation under $\mathbb{P}(f_\theta \mid \mathcal{F}_t)$.

Summing over $t = 1, \ldots, T$ and taking expectations yields

$$\mathbb{E}\left[\sum_{t=1}^{T}\big(f_\theta(x_t^*(\theta)) - f_\theta(x_t)\big)\right] \le \widetilde{O}\left(\mathbb{E}\left[\sum_{t=1}^{T}\sigma_t(x_t)\right]\right).$$

By Cauchy–Schwarz,

$$\sum_{t=1}^{T}\sigma_t(x_t) \le \sqrt{T}\left(\sum_{t=1}^{T}\sigma_t^2(x_t)\right)^{1/2}.$$

Since $x_t \in \mathcal{X}_t \subseteq \mathcal{X}_{\mathcal{G}}(\eta)$ for every t, the standard GP information-gain bound implies

$$\sum_{t=1}^{T}\sigma_t^2(x_t) \le \widetilde{O}(\Gamma_T(\mathcal{X}_{\mathcal{G}}(\eta))).$$

Combining the previous two displays gives

$$\mathbb{E}\left[\sum_{t=1}^{T}\big(f_\theta(x_t^*(\theta)) - f_\theta(x_t)\big)\right] \le \widetilde{O}\Big(\mathbb{E}\Big[\sqrt{T\,\Gamma_T(\mathcal{X}_{\mathcal{G}}(\eta))}\Big]\Big),$$

as desired. □

Discovery error bound. We now bound the range of the objective over the generator-reachable support.

Lemma 5 (Range bound for finite-domain Gaussian processes)**.** *Under the Gaussian process model above,*

$$\mathbb{E}\big[R_{\mathcal{G}}(\theta)^2\big] \le 8\kappa^2\,\mathbb{E}\big[1+\log\big(2|\mathcal{X}_{\mathcal{G}}(\eta)|\big)\big]\,,$$

where

$$R_{\mathcal{G}}(\theta) := f_\theta(x^*_{\mathcal{G}}(\theta)) - \min_{x\in\mathcal{X}_{\mathcal{G}}(\eta)} f_\theta(x).$$

Consequently,

$$\frac{2\sqrt{\mathbb{E}[R_{\mathcal{G}}(\theta)^2]}}{p_{\mathrm{dis}}} \le \frac{4\sqrt{2}\,\kappa\sqrt{\mathbb{E}\big[1+\log\big(2|\mathcal{X}_{\mathcal{G}}(\eta)|\big)\big]}}{p_{\mathrm{dis}}}.$$

Proof. For a fixed value of η, the range $R_{\mathcal{G}}(\theta)$ is bounded by the range of the Gaussian process over the finite set $\mathcal{X}_{\mathcal{G}}(\eta)$:

$$R_{\mathcal{G}}(\theta) \le \max_{x\in\mathcal{X}_{\mathcal{G}}(\eta)} f_\theta(x) - \min_{x\in\mathcal{X}_{\mathcal{G}}(\eta)} f_\theta(x).$$

Since $f_\theta \mid \eta \sim \mathcal{GP}(m,k)$ and $k(x,x)\le\kappa^2$, each $f_\theta(x)$, conditional on η, is a Gaussian random variable with variance at most κ^2. Hence, for any finite set $A\subseteq\mathcal{X}$,

$$\mathbb{E}\left[\left(\max_{x\in A} f_\theta(x) - \min_{x\in A} f_\theta(x)\right)^2 \,\middle|\, \eta\right] \le 8\kappa^2\big(1+\log(2|A|)\big).$$

Applying this bound with $A=\mathcal{X}_{\mathcal{G}}(\eta)$ gives

$$\mathbb{E}\big[R_{\mathcal{G}}(\theta)^2 \,\big|\, \eta\big] \le 8\kappa^2\big(1+\log(2|\mathcal{X}_{\mathcal{G}}(\eta)|)\big).$$

Taking expectation over η yields

$$\mathbb{E}\big[R_{\mathcal{G}}(\theta)^2\big] \le 8\kappa^2\,\mathbb{E}\big[1+\log\big(2|\mathcal{X}_{\mathcal{G}}(\eta)|\big)\big]\,.$$

The final displayed inequality follows by substituting this bound into

$$\frac{2\sqrt{\mathbb{E}[R_{\mathcal{G}}(\theta)^2]}}{p_{\mathrm{dis}}}.$$

□

Main result. Combining the bounds on support, discovery, and selection yields the following explicit regret guarantee.

Theorem 6 (Explicit GP regret bound)**.** *Suppose the Gaussian process model above holds and the discovery condition holds with parameter $p_{\mathrm{dis}}>0$. Let $\varepsilon_{\mathrm{sup}}$ denote the expected support gap. Then*

$$\mathrm{Regret}(T) \le T\,\varepsilon_{\mathrm{sup}} + \frac{4\sqrt{2}\,\kappa\sqrt{\mathbb{E}\big[1+\log\big(2|\mathcal{X}_{\mathcal{G}}(\eta)|\big)\big]}}{p_{\mathrm{dis}}} + \widetilde{O}\Big(\mathbb{E}\Big[\sqrt{T\,\Gamma_T(\mathcal{X}_{\mathcal{G}}(\eta))}\Big]\Big)\,.$$

C.4.1. Deterministic separation: zero support error

We first consider a deterministic setting in which task-side information identifies a subset of the domain guaranteed to contain the global optimum. Let the domain decompose into N disjoint blocks,

$$\mathcal{X} = \bigsqcup_{j=1}^{N} \mathcal{X}_j, \qquad |\mathcal{X}_j| = m.$$

Let $\eta \in [N]$ be task-side information observed by the generator, and suppose the objective satisfies

$$f(j,u) = \Delta \mathbf{1}\{j = \eta\} + W_{j,u},$$

where $W_{j,u} \in [0,1]$ and $\Delta > 1$. Then for every $x \in \mathcal{X}_\eta$, $f(x) \geq \Delta$, while for every $x \notin \mathcal{X}_\eta$, $f(x) \leq 1$. Since $\Delta > 1$, all global maximizers lie in $\mathcal{X}_\eta$.

If the generator-reachable support satisfies

$$\mathcal{X}_{\mathcal{G}}(\eta) = \mathcal{X}_\eta,$$

then the best generator-reachable point is globally optimal, so

$$x^*_{\mathcal{G}} \in \arg\max_{x \in \mathcal{X}} f(x), \qquad \varepsilon_{\text{sup}} = 0.$$

Search-complexity comparison. This example is deterministic and does not require specifying a Bayesian prior. It illustrates the search-space reduction induced by task-side information. A method that does not use η must search over the full domain of size Nm, whereas GaS restricts search to the block $\mathcal{X}_\eta$ of size m. Thus, in an unstructured finite-domain search problem, the number of candidate evaluations required to exhaust the relevant action set is reduced from Nm to m, a factor-N reduction, while incurring no support error.

C.4.2. Bayesian deterministic separation: zero support error

We first consider a Bayesian setting in which task-side information identifies a subset of the domain guaranteed to contain the global optimum. Let

$$\mathcal{X} = \bigsqcup_{j=1}^{N} \mathcal{X}_j, \qquad |\mathcal{X}_j| = m,$$

and write points as $x = (j,u)$. Let

$$\eta \sim \text{Unif}([N])$$

denote task-side information available to the generator. Conditional on η, define

$$f_\theta(j,u) = \Delta \mathbf{1}\{j = \eta\} + W_{j,u},$$

where $\Delta > 1$ and $W_{j,u} \in [0,1]$. The variables $W_{j,u}$ may be deterministic or drawn from any prior supported on $[0,1]$.

For any point $x = (\eta, u) \in \mathcal{X}_\eta$,

$$f_\theta(x) \geq \Delta,$$

whereas for any point $x = (j,u) \notin \mathcal{X}_\eta$,

$$f_\theta(x) \leq 1.$$

Since $\Delta > 1$, every global maximizer lies in $\mathcal{X}_\eta$. Therefore, if the generator-reachable support satisfies

$$\mathcal{X}_{\mathcal{G}}(\eta) = \mathcal{X}_\eta,$$

then the best generator-reachable point is globally optimal and

$$\varepsilon_{\text{sup}} = \mathbb{E}\big[f_\theta(x^*) - f_\theta(x^*_{\mathcal{G}})\big] = 0.$$

Regret comparison. The comparison is between two Bayesian procedures. A standard Thompson sampling baseline does not observe η, operates under the marginal prior over the full domain, and must identify which of the N blocks contains the elevated mean. In an unstructured finite-domain setting, its effective search space has size Nm, leading to finite-domain posterior-sampling complexity of order

$$\widetilde{O}(\sqrt{T\,Nm}).$$

By contrast, GaS observes η through the generator-side task information and restricts the candidate support to $\mathcal{X}_\eta$. Since $\varepsilon_{\text{sup}} = 0$, the regret decomposition gives

$$\text{Regret}_{\text{GaS}}(T) \le O(1/p_{\text{dis}}) + \widetilde{O}(\sqrt{T\,m}).$$

Thus, in this Bayesian deterministic-separation example, GaS removes the dependence on the $N-1$ irrelevant blocks without incurring support error, reducing the dominant statistical term by a factor of $\sqrt{N}$.

C.4.3. Gaussian processes with task-side information

We now consider a Bayesian setting in which task-side information identifies a promising block only probabilistically. Let

$$\mathcal{X} = \bigsqcup_{j=1}^{N} \mathcal{X}_j, \qquad |\mathcal{X}_j| = m,$$

and write points as $x = (j, u)$. Let

$$\eta \sim \text{Unif}([N])$$

denote task-side information available to the generator but not to a standard Thompson sampling baseline. Conditional on η, define

$$f_\theta(j, u) = \Delta \mathbf{1}\{j = \eta\} + Z_j + W_{j,u},$$

where

$$Z_j \sim \mathcal{N}(0, \tau^2), \qquad W_{j,u} \sim \mathcal{N}(0, \sigma_W^2),$$

independently across j, u. Thus, before observing η, the prior over f_θ is a mixture over which block has elevated mean. Conditional on η, however,

$$f_\theta \mid \eta \sim \mathcal{GP}(m_\eta, k), \qquad m_\eta(j, u) = \Delta \mathbf{1}\{j = \eta\}.$$

The comparison is therefore between two Bayesian procedures. Standard Thompson sampling does not observe η, operates under the marginal mixture prior over f_θ, and selects from the full domain $\mathcal{X}$. GaS uses the task-side information η through the generator, which induces the conditional prior $f_\theta \mid \eta$ and restricts the candidate support to

$$\mathcal{X}_{\mathcal{G}}(\eta) = \mathcal{X}_\eta.$$

Support gap. Let

$$A := \max_u (Z_\eta + W_{\eta,u}), \qquad B := \max_{j \ne \eta, u} (Z_j + W_{j,u}).$$

Then the best generator-reachable value is

$$f_\theta(x^*_{\mathcal{G}}) = \Delta + A,$$

whereas the global optimum over the full domain is

$$f_\theta(x^*) = \max\{\Delta + A, B\}.$$

Hence

$$\varepsilon_{\text{sup}} = \mathbb{E}[(B - \Delta - A)_+].$$

This gap is nonzero only when random fluctuations outside the task-relevant block overcome the mean advantage Δ. Standard Gaussian tail bounds imply that, for $s^2 = \tau^2 + \sigma_W^2$ and $M = (N-1)m$,

$$\varepsilon_{\text{sup}} \le C_s\, M \exp\left(-\frac{\Delta^2}{2s^2}\right),$$

for a constant C_s depending on s. Thus the support error decays exponentially in the signal-to-noise ratio Δ/s.

Regret of GaS. Conditioned on η, GaS performs Bayesian optimization only over the block $\mathcal{X}_\eta$, which has size m. Applying the regret decomposition gives

$$\text{Regret}_{\text{GaS}}(T) \le T\varepsilon_{\text{sup}} + O(1/p_{\text{dis}}) + \widetilde{O}\left(\sqrt{T\,\Gamma_T(\mathcal{X}_\eta)}\right).$$

For a finite domain of size m, this yields the simplified scaling

$$\text{Regret}_{\text{GaS}}(T) = T\varepsilon_{\text{sup}} + O(1/p_{\text{dis}}) + \widetilde{O}(\sqrt{Tm}).$$

Regret of full-domain Thompson sampling. A standard Thompson sampling baseline that does not observe η must optimize under the marginal prior over all Nm points. Since the elevated block is unknown, its effective action space remains the full domain $\mathcal{X}$. Generic finite-domain posterior-sampling bounds therefore scale as

$$\text{Regret}_{\text{TS}}(T) = \widetilde{O}(\sqrt{T\,Nm}).$$

Equivalently, in information-gain notation, the selection complexity is governed by the full-domain quantity

$$\widetilde{O}\left(\sqrt{T\,\Gamma_T(\mathcal{X})}\right),$$

rather than $\Gamma_T(\mathcal{X}_\eta)$.

Comparison. Thus GaS replaces the full-domain statistical complexity

$$\widetilde{O}(\sqrt{T\,Nm})$$

with

$$\widetilde{O}(\sqrt{Tm}) + T\varepsilon_{\text{sup}} + O(1/p_{\text{dis}}).$$

When Δ/s is large enough that ε_{sup} is small, the dominant statistical term is reduced by a factor of $\sqrt{N}$. If $N = 2^L$ and $m = \text{poly}(L)$, this corresponds to an exponential improvement in L, up to the support and discovery terms.

C.5. Summary and implications

We summarize the main implications of the regret analysis for the generate–and–select framework.

Decomposition of regret. The regret admits the decomposition

$$\text{Regret}(T) = \underbrace{R_{\text{sup}}(T)}_{\text{support error}} + \underbrace{R_{\text{dis}}(T)}_{\text{discovery error}} + \underbrace{R_{\text{sel}}(T)}_{\text{selection error}}.$$

These terms correspond to successive gaps induced by the nested sequence of search spaces

$$\mathcal{X} \supseteq \mathcal{X}_{\mathcal{G}}(\eta) \supseteq \mathcal{X}_t.$$

Each term captures a distinct aspect of the method:

- The **support error** reflects the limitation of restricting optimization to the generator-reachable set $\mathcal{X}_{\mathcal{G}}(\eta)$.
- The **discovery error** measures how quickly the generator proposes high-quality candidates within this set.
- The **selection error** captures the statistical efficiency of the surrogate model and acquisition rule over the discovered candidates.

Role of the generator. The generator affects the regret through two quantities:

- the *support quality*, via the gap $f_\theta(x^*(\theta)) - f_\theta(x^*_{\mathcal{G}}(\theta))$;
- the *discovery probability* p_{dis}, which controls how quickly the best point in $\mathcal{X}_{\mathcal{G}}(\eta)$ is proposed.

A generator that concentrates mass on high-quality regions of the search space can simultaneously reduce both terms.

Role of the surrogate model. The surrogate model and acquisition rule determine the selection error. In the Gaussian process setting, this term scales with the information gain $\Gamma_T(\mathcal{X}_{\mathcal{G}}(\eta))$, leading to bounds of the form

$$\widetilde{O}\Big(\sqrt{T\,\Gamma_T(\mathcal{X}_{\mathcal{G}}(\eta))}\Big)\,.$$

Crucially, this depends only on the size and structure of the generator-reachable support, not on the full domain $\mathcal{X}$.

D. Additional experimental details and results

This section provides additional details and supporting results for the experimental evaluation in Section 4. We include task descriptions and ablation studies that further analyze the behavior of GaS. Together, these results provide a more detailed and causal understanding of the performance trends observed in the main paper.

D.1. Experimental details

We describe the experimental setup in detail, including task definitions and implementation specifics.

D.1.1. Tasks

Semantle. Semantle Agarwal et al. (2025) is a word-search task in which the goal is to identify a hidden English word using a limited number of guesses. After each guess, the optimizer receives only a scalar similarity score between the guessed word and the target word. Each problem instance consists of a target word together with an initial warm-start set of 20 words and their scores. For Semantle, we report the *fraction of runs solved* out of 50 total runs (10 target words $\times$ 5 warmstart sets). A run is considered solved once the method proposes the hidden target word. Thus, the curve reports the cumulative fraction of runs that have reached the target by a given number of black-box evaluations. We use $k = 10$ in-context examples and an evaluation batch size of $q = 10$ for all methods.

Molecular optimization. For molecular optimization, each candidate is a SMILES string. The objective is a scalarized score combining drug-likeness and binding affinity. Drug-likeness is measured by QED, while binding affinity is measured using Vina docking scores. Following Agarwal et al. (2025), Vina scores are normalized using the typical range $[-12, -3]$, negated so that larger values are better, and then combined with QED as

$$f(x) = 0.2 \cdot \mathrm{QED}(x) + 0.8 \cdot \mathrm{NNV}(x),$$

where $\mathrm{NNV}(x)$ denotes the negative normalized Vina score. Thus, higher objective values correspond to molecules with better combined drug-likeness and binding affinity. The initial warm-start set contains `3` molecules. We use $k = 10$ in-context examples and an evaluation batch size of $q = 5$ for all methods.

1D-ARC. 1D-ARC Agarwal et al. (2025) is a one-dimensional version of the Abstraction and Reasoning Corpus in which each instance provides a small set of input-output grid transformations. Following the hypothesis-and-program search formulation, the method must infer the underlying rule and generate a Python implementation that solves the training examples. The 1D-ARC-Hard subset is constructed by first running repeated sampling for 100 generations on each original 1D-ARC problem and then selecting instances that remain unsolved, making the benchmark more suitable for evaluating adaptive search methods. The initial warm-start set contains the same 100 (docstring, code) pairs and their scores. We use $k = 3$ in-context examples and an evaluation batch size of $q = 5$ for all methods.

Symbolic regression. Each candidate is a symbolic expression in the variable t. Expressions are validated via their abstract syntax tree (AST), allowing arithmetic operations, numeric constants, t, and the functions `sin`, `cos`, and `exp`; invalid expressions are rejected. Valid candidates are scored by negative mean squared error against target outputs, with optimal value 0. For surrogate modeling, expressions are embedded by evaluating them on a fixed 64-point grid over $[-5, 5]$. Each run is initialized with a curated warm-start library of `20` expressions containing polynomial, trigonometric, and exponential terms. We use $k = 10$ in-context examples and an evaluation batch size of $q = 10$ for all methods.

D.1.2. Runtime comparison

GaS incurs only modest computational overhead relative to existing baselines. Across tasks, per-iteration runtimes are comparable, with GaS typically incurring a small additional cost due to evaluating acquisition functions over larger candidate pools. These results indicate that the performance gains of GaS are not driven by substantially increased computational resources.

Table 1: Runtime comparison across tasks and methods. Values denote per-iteration time in seconds.

Task	BOPRO	OPRO	GaS-qNEI	GaS-qTS
Semantle	8.50	7.42	10.42	9.51
MolOpt	7.93	7.11	10.37	6.59
SymReg	13.58	9.83	17.19	16.80
1D-ARC	46.05	38.45	43.99	56.95

D.1.3. Implementation and compute environment

We implement all methods in `PyTorch`, using `BoTorch` for Gaussian process modeling and `Sentence-Transformers` for embedding models. LLM inference is performed via `AWS Bedrock`. Experiments are run on `Ubuntu 22.04.5 LTS` with 28 `vCPUs`, 177 GB RAM, and a single `NVIDIA H100 80GB` GPU.

D.1.4. Prompts

D.1.5. Warm-start Generation

Semantle

```
Your task is to guess a hidden test word from the English dictionary.
Start by guessing {n_cands} diverse words to maximize your chance of
guessing the hidden word. Be creative.

Do not guess any of the following words: {list of words repeated by the
LLM in the previous iteration}.
```

Dockstring

```
Your task is to find the optimal drug molecule that has both a high
druglikeness (QED) as well as a strong binding affinity (vina) to the
target protein. While both properties are important, binding affinity is
twice as important as the druglikeness. Here is the target protein: {
target_protein}.

Do not propose any of the following molecules: {list of molecules
repeated by the LLM in the previous iteration}.
```

1D-ARC

```
Given the following {len(task_demos)} examples of input-output grids of
integers from 0-9, your task is to determine the transformation logic
common to all examples and provide a python function that converts each
input grid into its corresponding output grid. Here are the examples:

INPUT #1:
[[1,1,1,1.....]]
OUTPUT #1:
[[1,1,1,1.....]]
...

Your functions should have the following signature:

python
def transform(input: np.ndarray) -> np.ndarray:
    # Your code here

The following packages are already imported so do not repeat them: `numpy
as np` and `itertools`

Start by guessing exactly {n_cands} diverse solutions that could solve
the task. Make sure to not repeat any previous docstring or code. Be
creative in the logic of your proposals!

Do not repeat any of the following guesses:

REPEAT #1:
<docstring>This function...</docstring>

python
def transform(input: np.ndarray) -> np.ndarray:
    ...
```

```
REPEAT #2:
...

(Note: directly start your response using the specified output format)
```

Symbolic Regression

```
Your task is to find a symbolic expression in terms of t that best fits the observed input-output
     pairs. Use only valid expressions over t with numbers, the operators +, -, *, /, and the
     functions sin, cos, and exp.

Here are the observed input-output pairs:
t = {t_vals}
y = {y_vals}

Start by guessing {n_cands} diverse expressions to maximize your chance of finding the best expression
     . Be creative.

Do not guess any of the following expressions: {list of expressions repeated by the LLM in the
     previous iteration}.

(Note: give only a list of expressions in the provided JSON format, e.g. {response: [expr1, expr2,
     ...]})
```

D.1.6. Solution Representation

To compute embeddings for the generated solutions, we first map the raw solution text to a template to provide additional context for sentence-embedding models when needed:

- Semantle: `"{x}"`
- 1D-ARC: `"Algorithm:{x}"`
- Dockstring: `"Protein target:{target} Molecule Candidate:{x}"`
- Symbolic Regression: `"Outputs on a fixed grid"`

D.1.7. Solution Generation

Semantle

```
Your task is to guess a hidden test word from the English dictionary. Use
the below series of your previous guesses (in increasing) order of their
similarity to the hidden word in terms of their *meaning*) to make a new
guess. Your new guess should not have been made before and should score
higher than your previous guesses. Analyze your previous guesses to
decide what word could reach a target score of {target_score:.4f}.
If you guess an invalid word (i.e., not in the dictionary or a repeat
guess), you will get no score, so stick to proper, single-word English
words, and do not repeat your previous guesses!
Now, guess exactly n=%s new word(s) that could give you a target score of
{target_score:.4f}.
(Note: give only the word in the provided JSON format)
```

Dockstring

```
Your task is to find the optimal drug molecule that has both a high
druglikeness (QED) as well as a strong binding affinity (vina) to the
target protein. While both properties are important, binding affinity is
twice as important as the druglikeness. Here is the target protein: {
target_protein}. Use the below series of your previous guesses in
increasing order of their scalarized scores (between 0 and 1) to make a
new guess to maximize the score. Your new guess should not have been made
before. Analyze your previous guesses to decide what molecule to guess
next. If you propose an invalid molecule or make a repeat guess, you will
get no score, so stick to valid SMILES strings, and do not repeat your
previous guesses!
Now, guess exactly n=%s new molecule(s) that could be the optimal molecule..
Be creative!
Now, guess exactly n=%s new molecule(s) that could give you a target score of
{target_score:.4f}.
(Note: give only a list of SMILES strings in the provided JSON format, e.
g. \
{{response: [SMILES1, SMILES2, ...]}})
```

1D-ARC

```
Given the following {len(task_demos)} examples of input-output grids of
integers from 0-9, your task is to determine the transformation logic
common to all examples and provide a python function that converts each
input grid into its corresponding output grid. Here are the examples:

INPUT #1:
[[1,1,1,1.....]]
OUTPUT #1:
[[1,1,1,1.....]]
...

Each number in a grid may be treated as a unique color and a 0 may be
treated as an empty or black cell. Continuous sequences of numbers row-
wise, column-wise, or diagonally may represent objects forming patterns.
The transformation involves figuring out how these object patterns change
from the input to the output. For e.g., object patterns may contain
shapes like rectangles, triangles, or crosses which can then be mirrored,
rotated, translated, deformed, combined, repeated, etc.

Your functions should have the following signature:

python
def transform(input: np.ndarray) -> np.ndarray:
# Your code here

The following packages are already imported so do not repeat them: `numpy
as np` and `itertools`

Here are your top previous guesses and scores (in ascending order of
accuracy compared to the target solution)

GUESS (score: ...):
<docstring> ... </docstring>

GUESS (score: ...):
<docstring> ... </docstring>

GUESS (score: ...):
<docstring> ... </docstring>
```

```
....

Analyze your previous guesses to make exactly {n} new guess(es) likely to
give you a target score of {target_score:.4f}. Be creative in your logic
!

Do not repeat any of the following guesses:

REPEAT #1:
<docstring>This function...</docstring>

python
def transform(input: np.ndarray) -> np.ndarray:
...

REPEAT #2:
...

(Note: directly start your response using the specified output format)
```

Symbolic Regression

```
Given the following observed input-output pairs, your task is to find a symbolic expression in terms
     of t that best fits the data. Use only valid expressions over t with numbers, the operators +, -,
      *, /, and the functions sin, cos, and exp.

t = {t_vals}
y = {y_vals}

Use the below series of your previous guesses, in {increasing/decreasing} order of their scores, to
     make a new guess. Your new guess should not have been made before and should score higher than
     your previous guesses. Analyze your previous guesses to decide what expression could reach the
     target score of {target_score}.

If you propose an invalid expression or repeat a previous guess, you will get no score, so stick to
     valid expressions over t only and do not repeat your previous guesses.

In particular, DO NOT REPEAT the following expressions: {list of expressions repeated by the LLM in
     the previous iteration}.

Here are some of your low-scoring previous guesses:
{low_scoring_expressions}

Here are your top previous guesses, from {worst/best} to {best/worst}:

Expression: {expression_1}
Score: {score_1}

Expression: {expression_2}
Score: {score_2}

...

Your last {num_prev_trials} guess(es) were:

Expression: {previous_expression_1}
Score: {previous_score_1}

Expression: {previous_expression_2}
Score: {previous_score_2}
```

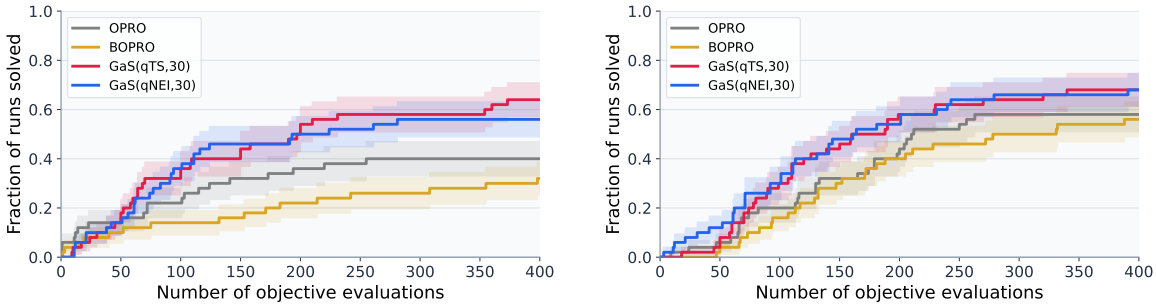
```
...

Now, guess exactly n={n_cands} new expression(s) that could give you a target score of {target_score}.

Note: give only a list of expressions in the provided JSON format, e.g. {response: [expr1, expr2,
     ...]}.
```

D.2. Ablation studies

We investigate how the performance of GaS emerges from the interaction between generation and selection. Generation constructs a candidate pool that evolves over time, while selection governs how effectively this pool is exploited. The ablations below isolate these components to provide a clearer, causal understanding of the method. Due to computational cost, all ablation experiments are conducted on the Semantle task.

D.2.1. Effect of generator choice

Across generator choices, GaS remains competitive or stronger than the baselines, suggesting that the gains come from the generate-and-select structure rather than a specific model. With `Gemma-3-27B`, both GaS variants solve a larger fraction of runs than OPRO, BOPRO; with qNEI slightly ahead than qTS during much of the optimization trajectory.



(a) `Mistral-large-3` (b) `Gemma-3-27B`

Figure 3: Effect of generator choice on Semantle. Curves show the fraction of solved runs versus objective evaluations for adaptive baselines and GaS variants using 30 generated candidates per iteration.

D.2.2. Effect of generation budget

Increasing the number of generated candidates per iteration improves performance, confirming that GaS benefits from a larger adaptive candidate pool. Gains are substantial when increasing from 20 to 30 candidates, with diminishing returns thereafter, consistent with improved discovery probability.

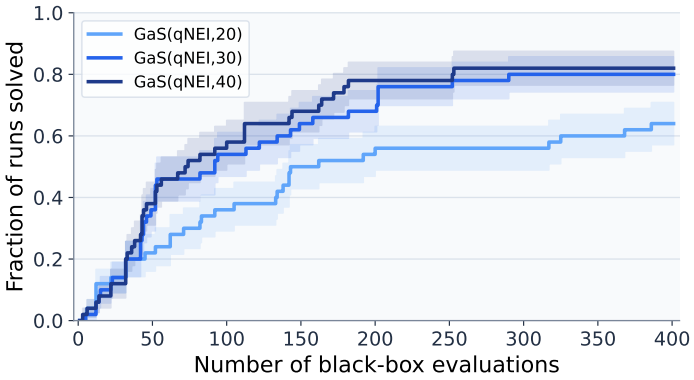


Figure 4: Ablation on the candidate pool size per iteration in GaS. All runs use the same black-box evaluation budget of 400; we vary only the number of candidates generated per iteration, comparing GaS with qNEI and 20, 30, or 40 generated candidates.

D.2.3. Effect of number of in-context exemplars

Performance depends sensitively on the number of in-context exemplars, but does not improve monotonically. Fewer exemplars perform best in this setting, suggesting that overly long prompts may introduce noise or reduce diversity in generation. This highlights the role of prompt design in shaping the generator-induced search space.

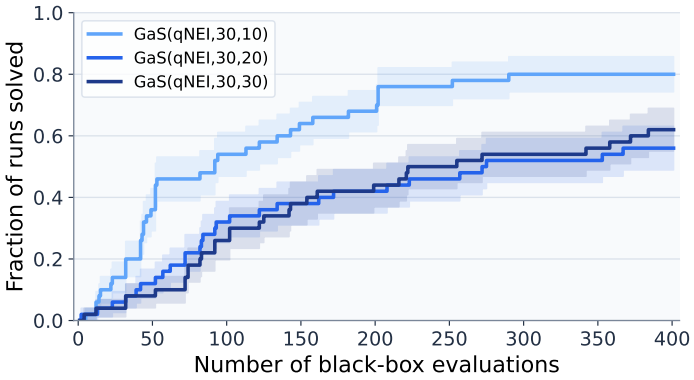


Figure 5: Effect of in-context example count in GaS. Curves show the fraction of solved runs versus black-box evaluations for GaS with qNEI, 30 generated candidates per iteration, and 10, 20, or 30 prompt exemplars.

D.2.4. Top-k versus random exemplars

We compare the default top-k exemplar selection strategy against randomly selected in-context exemplars. Both variants use GaS with qNEI and generate 30 candidates per iteration. Top-k improves over random exemplar selection, showing that the quality of the prompt context matters. This suggests that GaS benefits not only from generating an adaptive candidate pool, but also from conditioning the generator on informative prior evaluations.

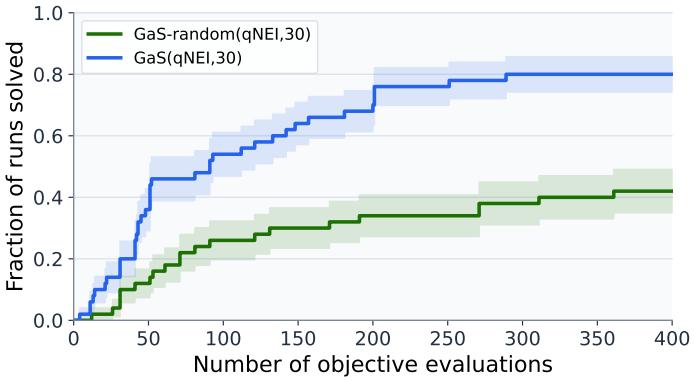


Figure 6: Effect of exemplar selection strategy in GaS. Curves show the fraction of solved runs versus objective evaluations for GaS with qNEI and 30 generated candidates per iteration, comparing top-k exemplar selection against random exemplar selection.

D.2.5. Adaptive vs. fixed candidate pools

Adaptive candidate generation substantially outperforms selection from a fixed pool, even when the pool is large. This shows that GaS benefits from iteratively refreshing the candidate set rather than simply filtering a large static pool, and that conditioning generation on the evolving dataset is critical for discovering high-quality solutions.

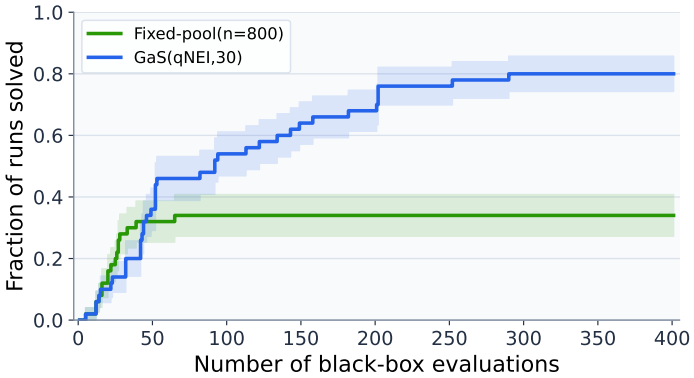


Figure 7: Comparison of fixed-pool and adaptive-pool optimization. The fixed-pool baseline selects from a static pool of 800 candidates, while GaS with qNEI adaptively generates 30 candidates per iteration.

D.2.6. Effect of acquisition function

Random selection already performs competitively, indicating that GaS constructs a high-quality candidate pool through generation alone. Model-based acquisition further improves performance, with qNEI achieving the strongest results, demonstrating that both generation and selection contribute to overall gains.

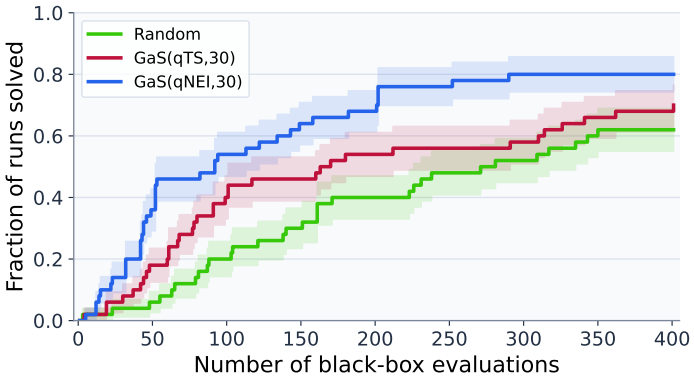
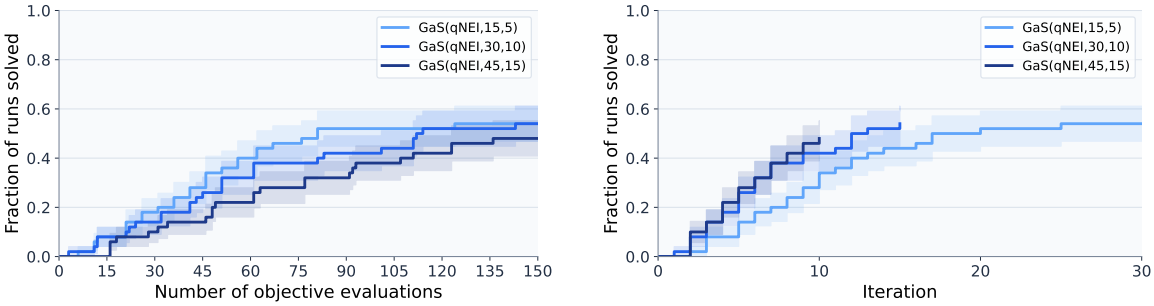


Figure 8: Ablation on acquisition functions in GaS. Curves show the fraction of solved runs versus evaluations for random selection, qTS, and qNEI.

D.2.7. Effect of evaluation batch size

Smaller per-iteration settings achieve better performance when measured against total objective evaluations, indicating that more frequent feedback improves sample efficiency. Larger settings make more progress per optimization iteration, highlighting a trade-off between sample efficiency and iteration-level efficiency when generation, context size, and evaluation batch size are scaled together.



(a) Performance vs. evaluations.

(b) Performance vs. iterations.

Figure 9: Scaling per-iteration parameters in GaS.