Adversarial Query Synthesis via Bayesian Optimization

Anonymous Author(s)
Affiliation
Address
email

Abstract

Benchmark workloads are extremely important to the database management research community, especially as more machine learning components are integrated into database systems. Here, we propose a Bayesian optimization technique to automatically search for difficult benchmark queries, significantly reducing the amount of manual effort usually required. In preliminary experiments, we show that our approach can generate queries with more than double the optimization headroom compared to existing benchmarks.

1 Introduction

Database researchers strive to build high-performance systems for processing SQL queries. To evaluate their work, the database community depends on benchmarks to approximate challenging queries. Traditionally, synthetic benchmarks (i.e., data drawn from random distributions) like the TPC family [14, 2] have been dominant, but some benchmarks over "real world" data, like the Join Order Benchmark (JOB) over the Internet Movie database (IMDb) have also been used [8]. Work applying machine learning techniques to database systems (e.g., [5]) has recently underscored the importance of "real world" benchmarks, as synthetic data distributions are normally either overly simplistic (e.g., a uniform key column) or entirely unlearnable (e.g., noise) [10]. As a concrete example, consider a recent report from AWS Redshift, which highlights the relative "easiness" of traditional benchmarks compared to the workloads they observe [23].

This gap has led to a scramble (or perhaps an arms race) for larger and more challenging benchmarks, with solutions including "matching" synthetic workloads to real traces [7], manually constructing new querysets over public data [11], synthesizing queries with LLMs [19], or adding additional queries to existing datasets [25]. With the exception of [19], all of these approaches require significant manual effort. And, with the exception of [25], all of these approaches do not optimize for the *difficulty* of the benchmark (i.e., the potential room for improvement over existing systems, which we call *headroom*).

In this work, we present a preliminary system that automatically synthesizes simple queries with significant optimization headroom (i.e., high difficulty). Our approach uses Bayesian optimization to discover query-and-plan pairs for which the discovered plan performs significantly better than the baseline system. In addition to being fully automated, this approach has three key advantages: first, by directly optimizing for query headroom, we show that we can find queries that have *more than double* the headroom of existing benchmarks. Second, by searching over plans in addition to queries, we generate *witness plans* with significantly better performance than the baseline system, which is potentially useful for debugging performance bugs. Third, by restricting our search space to simple queries (conjunctive queries with no aliased joins), our benchmark queries can run on every relational database system currently known to the authors.

Submitted to Machine Learning for Systems Workshop at (NeurIPS 2025). Do not distribute.

At a high level, our approach formulates the benchmark building problem as a optimization problem, and uses Bayesian optimization enhanced with a latent-space representation of both SQL queries and plans. To handle the structured, discrete nature of SQL queries, we use a composite variational autoencoder that encodes queries and plans into a shared continuous latent space, enabling efficient search. Queries are embedded with a large text embedding model and decoded with a fine-tuned compact LLM using grammar-constrained decoding [3], ensuring syntactic validity, while plans benefit from a closed vocabulary representation that guarantees well-formedness. This combination allows the system to efficiently propose and evaluate candidate pairs, iteratively refining its model to uncover challenging, high-headroom queries.

2 Methods

The core problem we address is the automated discovery of adversarial pairs (Q, P), where Q is a SQL query and P is an execution plan for Q. An adversarial pair is one where the latency of plan P, denoted $L(P)$, is significantly lower than the latency of the plan generated by the database's default query optimizer, $L(P_{\text{default}})$. We formulate this as a black-box optimization problem aimed at maximizing either the **relative speedup**, $L(P_{\text{default}})/L(P)$, or the **absolute speedup**, $L(P_{\text{default}}) - L(P)$.

Black-box and Bayesian optimization. In black-box optimization, we aim to optimize an *oracle* objective function $f(\mathbf{x})$ over a space of candidates $\mathbf{x}^* = \text{argmax}_{\mathbf{x}\in\mathcal{X}} f(\mathbf{x})$. Examples of such problems include molecule activity maximization for drug discovery [22, 12], and binding affinity of DNA sequences or proteins [1, 4]. Commonly, $f(x)$ is assumed to be expensive to evaluate or even completely unknown.

Bayesian optimization is a sample-efficient framework to solve these expensive model-based optimization problems [15, 13, 20]. At iteration t of BO, one has access to observations $\mathcal{D}_t = \{(\mathbf{x}_i, y_i)\}_{i=1}^t$, where y_i denotes the objective value of the input x_i. Typically, a Gaussian process [18] is employed as the surrogate model to approximate the objective function using these inputs and values. This surrogate model aids the optimization by employing an acquisition function, which strategically proposes the next candidates for evaluation. After querying these candidates through the true oracle, the surrogate model is updated with the new observations. This process gradually builds a more comprehensive dataset and refines the surrogate model, thereby improving the quality of the proposed samples in future iterations.

Bayesian optimization over SQL queries. Bayesian optimization is a sample-efficient technique for optimizing expensive black-box functions. However, standard BO operates on continuous vector spaces, whereas our search space consists of discrete, structured objects (SQL queries and query plans). We bridge this gap using Latent Space BO (LS-BO), a technique proven effective in domains like molecule design [12] and offline query planning [21].

The LS-BO framework requires two key components: (1) an encoder $\mathcal{E}$ that maps structured inputs into a continuous latent space, $\mathcal{Z}$, and (2) a decoder $\mathcal{D}$ that maps points from $\mathcal{Z}$ back into the structured input space. In this paper, we introduce a novel, composite Variational Autoencoder (VAE) architecture that creates a joint latent space for both queries and plans together.

We construct a continuous latent space $\mathcal{Z} = [z_q; z_p] \in \mathbb{R}^{320}$ that concatenates query VAE latents ($z_q \in \mathbb{R}^{256}$) with plan VAE latents ($z_p \in \mathbb{R}^{64}$):

$$\begin{aligned} &\mathcal{E}_q : \text{SQL string} \to z_q, && \mathcal{D}_q : z_q \to \text{SQL string}, \\ &\mathcal{E}_p : \text{plan string} \to z_p, && \mathcal{D}_p : z_p \to \text{plan string}. \end{aligned}$$

At inference time, decoding z yields an executable pair $(\widehat{Q}, \widehat{P})$; the plan string is turned into database-specific instructions (such as an optimizer hint string [16]) and combined with $\widehat{Q}$ for execution.

Plan representation and VAE. Our plan representation is identical to prior work using Bayesian optimization for query planning [21]. At a high level, we encode plans as strings of integers which are then interpreted as operator selections and join orders. Like SELFIES [6], every string represents a valid plan, and every plan is represented by at least one string, although (unlike SELFIES) some plans are represented by more than one string. This means that any sequence decoded from $\mathcal{E}_p$ can be interpreted as a valid plan.

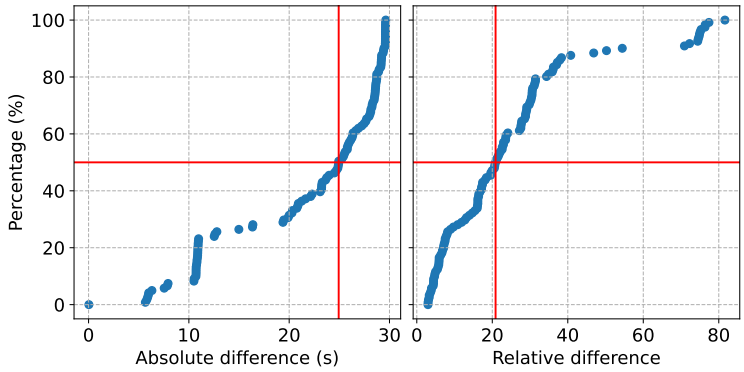

Figure 1: CDFs of absolute and relative differences between baseline system (DuckDB) and witness plan for our generated benchmark. Median values at red cross.

Query VAE. Unlike query plans, SQL lacks a SELFIES-style [6] closed vocabulary where any token sequence is valid. To sidestep data scarcity (∼20k queries) and grammar complexity, we do not train a full encoder–decoder VAE for queries. Instead, we (i) use a text embedding model as the query encoder $\mathcal{E}_q$ (OpenAI `text-embedding-3-large`), then (ii) fine-tune a small LLM (`Qwen-2.5-0.5B`) as a compact query decoder $\mathcal{D}_q$ to map the first 256 embedding dimensions back to SQL queries. This is similar to prompt tuning [9], where the embedding is passed to the LLM as a soft-prompt, and the LLM is trained to decode the embedding back to the original query string. We use grammar-constrained decoding during inference to guarantee syntactic correctness of the reconstructed query. This yields ∼67% string reconstruction accuracy for queries and ∼99% for plans.

A Simple SQL Subset To generate simple queries that work across a wide variety of databases, we focus on conjunctive queries [24]. Essentially, queries are restricted to: (1) joining together relations with equijoins (e.g., between foreign and primary keys), and (2) conjunctive predicates on table columns. Each query performs an ungrouped `count(*)` aggregation. Our formulation thus excludes SQL features like subqueries, window functions, recursion, etc. While simple, conjunctive queries are still difficult for modern query optimizers [8].

Grammar-constrained decoding for SQL. To make sure the query decoder $\mathcal{D}_q$ always decodes to a valid query, we compiled a concise EBNF for our SQL subset (by enumerating the powerset of joinable tables and filterable columns) and enforced it during $\mathcal{D}_q$ decoding via grammar constrained generation using vLLM. This addresses the main challenges associated with unconstrained decoding: (i) syntactic invalidity and (ii) semantically impossible references (e.g., undefined aliases). The grammar constraint only applies to $\mathcal{D}_q$, since $\mathcal{D}_p$ is designed to have a set of closed vocabulary where any decoded sequence is always valid.

3 Preliminary results

To test the feasibility of our approach, we generated a workload against the IMDb dataset (chosen to facilitate comparison with prior work using the same dataset). To match the original join order benchmark [8], we generated 122 query plans. These plans were collected by selecting the best candidates (in terms of relative and absolute headroom) from a weeklong BO run. Queries were executed on DuckDB [17] on a node equipped with an AMD Ryzen 5 3600 and 64GB of RAM. The entire dataset was cached in memory.

We plot the CDFs of the absolute and relative headrooms in Figure 1. The median query in our generated workload had an absolute headroom of 25 seconds, meaning that the witness plan was 25 seconds faster than the default plan chosen by DuckDB. In terms of relative headroom, the median query in our generated workload had a witness plan that was 20x faster than the default plan chosen by DuckDB. The largest differences found approach 30 seconds in absolute terms, and nearly 80x in relative terms. The least difficult query in our workload has less than a second of absolute improvement and only a 1.5x relative improvement (potentially attributable to noise). We hope that

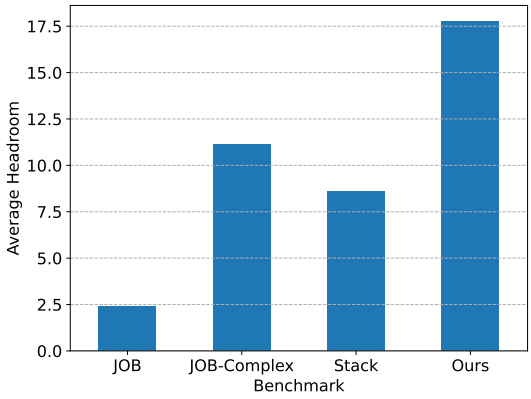

Figure 2: Comparison of geometric mean relative headroom for four benchmarks. Values for JOB and JOB-Complex are from [25]. Values for Stack are approximated from [11]. Values for "Ours" are computed with respect to the witness plan.

longer BO runs in the future will continue to shift both distributions right; the comparative "easiness" of the weakest generated queries is potentially attributable to a lack of search time.

To compare the quality of our generated benchmark against prior work, Figure 2 shows the mean (geometric) relative headroom of different benchmark. Since computing the headroom of a benchmark is expensive (i.e., one must execute millions of query plans per query), we rely on results reported in prior work, and thus compare only the mean relative headroom (as absolute headroom is not consistently reported).

4 Conclusion and Future Work

In this extended abstract, we presented a fully automated approach for creating challenging database benchmarks by viewing benchmark creation as an optimization problem and using Bayesian optimization in a joint latent space over both SQL queries and execution plans. Our combination of VAE decoders enables joint embeddings for queries and plans, while grammar-constrained decoding ensures syntactic validity of generated SQL within a simple, widely portable subset (conjunctive queries with equijoins and predicates). Together, these choices let the system efficiently find adversarial (query, plan) pairs with substantial optimization headroom.

Preliminary experiments on IMDb with DuckDB show promising evidence: from a week-long BO run we selected 122 candidates, and the resulting workload exhibits a median absolute headroom of 25 s and a median relative headroom of 20× between the default plan and the discovered witness plan (with the largest gaps approaching 30 s and 80×, respectively). These results suggest our automated process can find queries that are a lot harder than those in commonly used benchmarks.

For future work, with a much larger set of hard (query, plan) pairs from extended runs, we could fine-tune a schema-conditioned LLM to directly generate pairs in bulk, skipping Bayesian optimization for most cases. The model would learn from (SQL, plan, headroom) triples, use grammar-constrained decoding to preserve validity, and be paired with a fast verifier (parse + quick execution/cost check) to filter outputs. A simple reward model or DPO-style objective that favors higher headroom can bias generation toward adversarial regions without explicit search. In practice, we could batch-generate candidates, keep the top-k under the verifier, and fall back to optimization only when confidence is low, therefore cutting wall-clock time while maintaining quality through periodic active-learning sessions.

Overall, our method reduces manual effort, directly targets "difficulty" via headroom, and yields concrete witness plans that can help with debugging and system improvement, while keeping queries broadly executable across systems.

References

[1] Luis A. Barrera, Anastasia Vedenko, Jesse V. Kurland, Julia M. Rogers, Stephen S. Gisselbrecht, Elizabeth J. Rossin, Jaie Woodard, Luca Mariani, Kian Hong Kock, Sachi Inukai, Trevor Siggers, Leila Shokri, Raluca Gordân, Nidhi Sahni, Chris Cotsapas, Tong Hao, Song Yi, Manolis Kellis, Mark J. Daly, Marc Vidal, David E. Hill, and Martha L. Bulyk. Survey of variation in human transcription factors reveals prevalent dna binding changes. *Science*, 351(6280):1450–1454, 2016. doi: 10.1126/science.aad2257.

[2] Peter Boncz, Thomas Neumann, and Orri Erling. TPC-H Analyzed: Hidden Messages and Lessons Learned from an Influential Benchmark. In *Revised Selected Papers of the 5th TPC Technology Conference on Performance Characterization and Benchmarking - Volume 8391*, TPC '14, pages 61–76, Berlin, Heidelberg, 2014. Springer-Verlag. ISBN 978-3-319-04935-9. doi: 10.1007/978-3-319-04936-6_5. URL https://doi.org/10.1007/978-3-319-04936-6_5.

[3] Yixin Dong, Charlie F. Ruan, Yaxing Cai, Ruihang Lai, Ziyi Xu, Yilong Zhao, and Tianqi Chen. XGrammar: Flexible and Efficient Structured Generation Engine for Large Language Models, May 2025. URL http://arxiv.org/abs/2411.15100. arXiv:2411.15100 [cs].

[4] Nate Gruver, Samuel Stanton, Nathan C. Frey, Tim G. J. Rudner, Isidro Hotzel, Julien Lafrance-Vanasse, Arvind Rajpal, Kyunghyun Cho, and Andrew Gordon Wilson. Protein design with guided discrete diffusion. *arXiv Preprint*, 2023. doi: 10.48550/arXiv.2305.20009.

[5] Tim Kraska, Alex Beutel, Ed H. Chi, Jeffrey Dean, and Neoklis Polyzotis. The Case for Learned Index Structures. In *Proceedings of the 2018 International Conference on Management of Data*, SIGMOD '18, New York, NY, USA, 2018. ACM. ISBN 978-1-4503-4703-7. doi: 10.1145/3183713.3196909. URL http://doi.acm.org/10.1145/3183713.3196909.

[6] Mario Krenn, Florian Häse, AkshatKumar Nigam, Pascal Friederich, and Alán Aspuru-Guzik. Self-Referencing Embedded Strings (SELFIES): A 100% robust molecular string representation, March 2020. URL http://arxiv.org/abs/1905.13741. arXiv:1905.13741.

[7] Skander Krid, Mihail Stoian, and Andreas Kipf. Redbench: A Benchmark Reflecting Real Workloads, June 2025. URL http://arxiv.org/abs/2506.12488. arXiv:2506.12488 [cs].

[8] Viktor Leis, Andrey Gubichev, Atanas Mirchev, Peter Boncz, Alfons Kemper, and Thomas Neumann. How Good Are Query Optimizers, Really? *PVLDB*, 9(3):204–215, 2015. ISSN 2150-8097. doi: 10.14778/2850583.2850594. URL http://dx.doi.org/10.14778/2850583.2850594.

[9] Brian Lester, Rami Al-Rfou, and Noah Constant. The power of scale for parameter-efficient prompt tuning, 2021. URL https://arxiv.org/abs/2104.08691.

[10] Ryan Marcus, Andreas Kipf, Alexander Van Renen, Mihail Stoian, Sanchit Misra, Alfons Kemper, Thomas Neumann, and Tim Kraska. Benchmarking Learned Indexes. *PVLDB*, 14(1): 1–13, September 2020. doi: 10.14778/3421424.3421425. Citation Key: benchmark_lis.

[11] Ryan Marcus, Parimarjan Negi, Hongzi Mao, Nesime Tatbul, Mohammad Alizadeh, and Tim Kraska. Bao: Making Learned Query Optimization Practical. In *Proceedings of the 2021 International Conference on Management of Data*, SIGMOD '21, China, June 2021. ISBN 978-1-4503-8343-1. doi: 10.1145/3448016.3452838. Award: 'best paper award'.

[12] Natalie Maus, Haydn Jones, Juston Moore, Matt J Kusner, John Bradshaw, and Jacob Gardner. Local latent space bayesian optimization over structured inputs. In *Advances in Neural Information Processing Systems*, volume 35, pages 34505–34518. Curran Associates, Inc., 2022. doi: 10.48550/arXiv.2201.11872.

[13] Jonas Mockus. The Bayesian approach to global optimization. In *System Modeling and Optimization*, pages 473–481. Springer, 1982.

[14] Raghunath Othayoth Nambiar and Meikel Poess. The Making of TPC-DS. In *VLDB*, VLDB '06, pages 1049–1058, Seoul, Korea, 2006. VLDB Endowment. URL http://dl.acm.org/citation.cfm?id=1182635.1164217.

[15] M. A. Osborne, R. Garnett, and S. J. Roberts. Gaussian processes for global optimization. In *3rd International Conference on Learning and Intelligent Optimization (LION3)*, pages 1–15, 2009.

[16] PostgreSQL Developers. PostgreSQL hints, https://www.postgresql.org/docs/current/runtime-config-query.html, 2024. URL https://www.postgresql.org/docs/current/runtime-config-query.html. tex.key= 1.

[17] Mark Raasveldt and Hannes Mühleisen. DuckDB: an Embeddable Analytical Database. In *Proceedings of the 2019 International Conference on Management of Data*, SIGMOD '19, pages 1981–1984, New York, NY, USA, June 2019. Association for Computing Machinery. ISBN 978-1-4503-5643-5. doi: 10.1145/3299869.3320212. URL https://dl.acm.org/doi/10.1145/3299869.3320212.

[18] Carl Edward Rasmussen. Gaussian processes in machine learning. In *Summer School on Machine Learning*, pages 63–71. Springer, 2003.

[19] Tobias Schmidt, Viktor Leis, Peter Boncz, and Thomas Neumann. SQLStorm: Taking Database Benchmarking into the LLM Era. *PVLDB*, 18(11):4144–4157, 2025. doi: 10.14778/3749646.3749683.

[20] Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. Practical Bayesian optimization of machine learning algorithms. In *Proc NeurIPS*, volume 25, pages 2951–9, 2012. doi: 10.48550/arXiv.1206.2944.

[21] Jeffrey Tao, Natalie Maus, Haydn Jones, Yimeng Zeng, Jacob R. Gardner, and Ryan Marcus. Learned offline query planning via bayesian optimization. *Proc. ACM Manag. Data*, 3(3), June 2025. doi: 10.1145/3725316. URL https://doi.org/10.1145/3725316.

[22] Brandon Trabucco, Xinyang Geng, Aviral Kumar, and Sergey Levine. Design-bench: Benchmarks for data-driven offline model-based optimization. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *PMLR*, pages 21658–21676. PMLR, 17–23 Jul 2022. doi: 10.48550/arXiv.2202.08450.

[23] Alexander van Renen, Dominik Horn, Pascal Pfeil, Kapil Eknath Vaidya, Wenjian Dong, Murali Narayanaswamy, Zhengchun Liu, Gaurav Saxena, Andreas Kipf, and Tim Kraska. Why TPC is not enough: An analysis of the Amazon Redshift fleet. *Proceedings of the VLDB Endowment*, 2024. URL https://www.amazon.science/publications/why-tpc-is-not-enough-an-analysis-of-the-amazon-redshift-fleet.

[24] Qichen Wang, Bingnan Chen, Binyang Dai, Ke Yi, Feifei Li, and Liang Lin. Yannakakis+: Practical Acyclic Query Evaluation with Theoretical Guarantees. *Proc. ACM Manag. Data*, 3 (3):235:1–235:28, June 2025. doi: 10.1145/3725423. URL https://dl.acm.org/doi/10.1145/3725423.

[25] Johannes Wehrstein, Timo Eckmann, Roman Heinrich, and Carsten Binnig. JOB-Complex: A Challenging Benchmark for Traditional & Learned Query Optimization, July 2025. URL http://arxiv.org/abs/2507.07471. arXiv:2507.07471 [cs].