Towards Zero-Shot Pretrained Models for Efficient Black-Box Optimization

by

Jamison Chivvis Meindl

Bachelor of Science in Computer Science and Engineering and in Mathematics
Massachusetts Institute of Technology, 2024

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

MASTER OF ENGINEERING IN ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2025

Authored by: Jamison Chivvis Meindl
Department of Electrical Engineering and Computer Science
August 1, 2025

Certified by: Wojciech Matusik
Professor of Electrical Engineering and Computer Science, Thesis Supervisor

Accepted by: Katrina LaCurts
Chair, Master of Engineering Thesis Committee

Towards Zero-Shot Pretrained Models for Efficient Black-Box Optimization

by

Jamison Chivvis Meindl

Submitted to the Department of Electrical Engineering and Computer Science
on August 1, 2025 in partial fulfillment of the requirements for the degree of

MASTER OF ENGINEERING IN ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

ABSTRACT

Global optimization of expensive, derivative-free black-box functions requires extreme sample efficiency. While Bayesian optimization (BO) is the current state-of-the-art, its performance hinges on surrogate and acquisition function hyperparameters that are often hand-tuned and fail to generalize across problem landscapes. We present ZeroShotOpt, the first general-purpose, pretrained model for continuous black-box optimization tasks ranging from 2 D to 20 D. Our approach leverages offline reinforcement learning on large-scale optimization trajectories collected from 12 BO variants. To scale pretraining, we generate millions of synthetic Gaussian process-based functions with diverse landscapes, enabling the model to learn transferable optimization policies. As a result, ZeroShotOpt achieves robust zero-shot generalization on a wide array of unseen synthetic and real-world benchmarks, matching or surpassing the sample efficiency of leading global optimizers, including BO, while also offering a reusable foundation for future extensions.

Thesis supervisor: Wojciech Matusik
Title: Professor of Electrical Engineering and Computer Science

Acknowledgments

Thank you to Wojciech Matusik for advising me during my MEng. I have appreciated his feedback and thoughtful ideas and look forward to continuing to work together in the future. Additionally, thank you to Yunsheng Tian for his mentorship, starting as an undergraduate and moving on to my MEng. Thank you to my coauthors on this work, including Tony Cui, Veronika Thost, Zhang-Wei Hong, Johannes Dürholt, Jie Chen, and Mina Konaković Luković, who all provided valuable assistance with research and advice for this work. Lastly, thank you to my family and friends who have supported me throughout my academic journey.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Black-box optimization under tight evaluation budgets is pivotal in many scientific and engineering settings. Since derivatives are often unavailable, classical gradient-based solvers such as Newton's method or conjugate-gradient are not applicable. Practitioners therefore turn to gradient-free heuristics such as genetic algorithms, simulated annealing, and evolutionary strategies, which often require thousands of evaluations to converge [2–4]. When each experiment or simulation is slow or costly, such sample counts become prohibitive. Therefore, we focus on improving continuous black-box optimization under strict evaluation constraints.

Bayesian optimization (BO) [5] reduces this burden by fitting a probabilistic surrogate (typically a Gaussian process) and selecting queries via a heuristic acquisition function that balances exploration and exploitation. BO has driven progress in materials discovery [6], molecular design [7], clinical prognosis [8], and hyper-parameter tuning [9]. However, BO's success hinges on hand-chosen kernels, acquisition functions, and hyperparameters whose optimal settings vary across landscapes and are difficult to tune without expert insight or extra evaluations.

Consider a researcher maximizing the efficacy of a drug formulation. Each candidate formulation must pass a clinical assay, capping the budget at *fifty* trials. The formulation parameters seen in Table 1.1 span dosage, delivery rate, and co-administered compounds, which are continuous parameters with unknown interactions. Standard heuristics exhaust the budget before converging, while BO would succeed only if its kernel and acquisition hyperparameters were tuned. Therefore, there is a need for a low-budget general-purpose global optimizer that can perform well without tuning.

Table 1.1: Illustrative formulation parameters for a 50-trial budget.

Parameter	**Description**
Dosage Level	Active ingredient, 10–100 mg
Delivery Rate	Slow- vs. immediate-release (time constant)
Co-administered Compounds	Enzyme inhibitors / buffers (concentration)

The large amounts of data and increasing computing power available today have led to the development of pretrained transformer-based models, used in a variety of data-driven decision making scenarios, such as machine translation [10], regression [11], and robotic control [12].

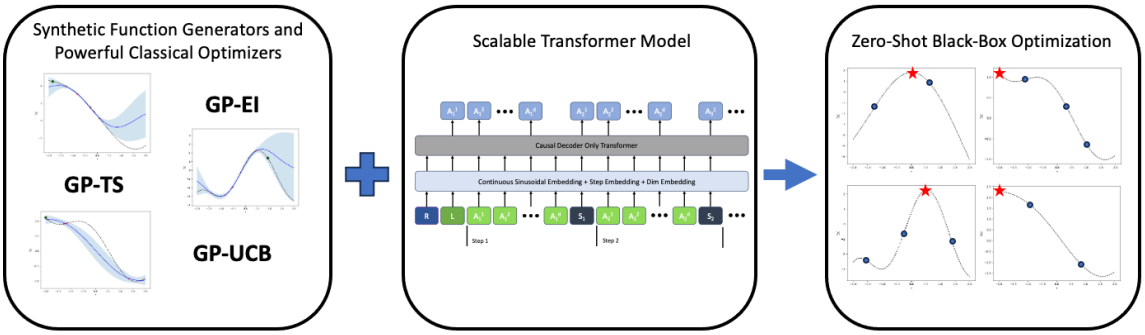


Figure 1.1: Overall Diagram of ZEROSHOTOPT. We use a combination of diverse synthetic functions and classical optimizers to train a pretrained transformer model for efficient black-box optimization.

Recent approaches have begun exploring using pretrained models for optimization, including both transformer architectures and offline pretraining [13–15]. However, the full potential of pretrained models in optimization remains untapped, in part due to the limited availability of massive high-quality optimization datasets [16]. Furthermore, demonstrating robust zero-shot generalization on completely unseen problems within low evaluation budgets remains an open challenge, as most pretrained approaches require training on similar data to which the model is tested. Consequently, traditional optimization, such as Bayesian optimization, continues to be the preferred choice in practical applications.

In this paper, we introduce ZEROSHOTOPT, a *general-purpose pretrained optimizer* for continuous black-box problems up to 20 D. We train a 200 M parameter transformer-based model using offline reinforcement learning on a diverse dataset of trajectories generated using various BO variants on synthetic GP functions. As a result, ZEROSHOTOPT demonstrates a robust understanding of optimization dynamics, allowing it to generalize to unseen problems without domain-specific tuning.

In summary, our contributions are as follows.

1. We develop a *synthetic function generator* based on Gaussian processes and collect *large-scale optimization trajectories* from 12 BO variants as pretraining data. We provide this as an open-source dataset, totaling $\sim$ 20 million synthetic trajectories, providing a valuable large dataset of expert optimization trajectories.

2. We design and train a single scalable transformer-based model that operates across dimensions from 2 D to 20 D and up to 50 total evaluations.

3. Our extensive evaluation demonstrates that ZEROSHOTOPT achieves *strong zero-shot generalization* performance across a range of unseen global optimization benchmarks, matching the performance of existing state-of-the-art BO methods, while providing the opportunity for further extensions to our method.

Chapter 2

Related work

Bayesian optimization (BO) BO is a powerful global optimization technique due to its ability to efficiently handle expensive black-box functions by balancing exploration and exploitation with carefully designed acquisition functions, such as Expected Improvement (EI), LogEI [17], Upper Confidence Bound (UCB) and Vizier [18, 19]. Designed with different principles of trading-off exploration and exploitation, they suit different types of optimization problems. BO has been made accessible by a collection of open-source libraries and software including Spearmint [20], BoTorch [21], AutoOED [22], SMAC3 [23], HEBO [24], Openbox [25], etc. However, it remains a challenge for users to determine the most appropriate configuration for their specific problem setting, as the selection is often heuristic and depends on the problem landscape.

Offline reinforcement learning Decision Transformer [12], which models offline RL as sequence learning, is a primary inspiration for our model architecture. We utilize their idea of conditional reward within transformer-based sequence model as the foundation of our model. Follow-up works on Decision Transformer [26, 27] are possible future avenues for improvement. Other insights from offline reinforcement learning [28] could also be used to build upon our methodology.

Pretrained optimization methods The idea of learning to optimize has been studied relatively early in continuous gradient-based optimization [29–31]. Many early works reformulate optimization as a sequence prediction problem, training recurrent neural network (RNN) to predict the next point to evaluate. Yet, gradients need to be provided as additional information and they mostly assume in-domain settings [31]. One notable exception are Chen et al. [32], who show that trained RNNs are able to generalize from simple objective functions to a variety of other unseen test functions without gradient information. With the increase in computation and the rise of advanced deep learning models, more recent studies are able to scale up datasets considerably and approach optimization in new ways.

Chaybouti et al. [33] train a transformer optimizer using online meta RL, but still limited to learning task-specific solvers without generalization to a wide variety of unseen tasks. Similarly, neural acquisition process (NAP) [15] also considers a similar online RL-based formulation of the problem. They propose an additional training objective to explicitly predicting acquisition function values to guide RL exploration more effectively. They also

focus on evaluating transfer learning but, different from us, still train individually for different tasks on selected, task-related data.

Pretrained Optimization Model (POM) [34] introduces a population-based model for zero-shot optimization by formulating optimization as an evolutionary algorithm and using meta-learning for training the model. However, they focus on population-based, high-dimensional problems (>100 dimensions), which are beyond the scope we are considering.

Given the close connection with reinforcement learning, there are numerous other works that separately study the problem of learning the surrogate [35, 36] or the acquisition function [37, 38]. Large language models have also been applied out of the box for global optimization; for instance, by using in-context learning [39, 40].

BONET [14] applies a causal transformer and RL-inspired pretraining to specific optimization tasks. They focus on synthesizing optimization trajectories by reordering samples in a given dataset. The study only evaluates domain-specific extrapolation based on a relatively large-scale dataset, instead of tackling unseen optimization problems with just a few dozens of samples.

Next, OptFormer [13] is a text-based transformer framework designed for hyperparameter optimization for general problem space beyond continuous. The model is trained on a proprietary hyperparameter optimization database and exploits the textual nature by leveraging hyperparameter names and descriptions. The method shows promise for transformer-based models trained on large datasets by outperforming solutions such as BO on the given test cases, which are drawn from similar distributions as training data. While the model shows significant performance degradation when tested on distributions different from its training data, its strong performance on a large dataset marks an important step towards general-purpose transformer-based optimization.

Lastly, RIBBO [41] is a transformer-based framework trained using offline reinforcement learning. While similarly inspired by decision transformer, RIBBO is trained on the benchmark datasets taken from existing optimization tasks and uses fixed dimensionality. Therefore, RIBBO can only evaluate results for small training and evaluation datasets from similar distributions. However, the ideas of conditioned transformer-based models and the improved performance RIBBO shows over baselines on individual datasets provides a step towards outperforming BO in transformer-based optimization.

Altogether, ZeroShotOpt is the first general-purpose, zero-shot, transformer-based optimizer verified on continuous black-box optimization problems ranging from 2 D–20 D. Unlike BONET, OptFormer and RIBBO, which each excel only when the test distribution resembles the data they were trained on, ZeroShotOpt is pretrained once on ~ 20 million synthetic trajectories and then deployed *unchanged* on out-of-distribution tests. It retains competitive performance against strong BO baselines on *unseen* synthetic and real-world test suites. In short, ZeroShotOpt closes the outstanding gap left by prior transformer-based optimizers: robust out-of-distribution optimization on a wide range of test cases with a single, reusable model.

Chapter 3

Methodology

3.1 Problem statement

For generic black-box global optimization problems, the goal is to solve $\mathbf{x}^* = \arg\min_{\mathbf{x}\in\mathcal{X}} f(\mathbf{x})$, where $\mathbf{x} \in \mathbb{R}^d$ is a vector of decision variables, $\mathcal{X} \subseteq \mathbb{R}^d$ represents the feasible search space, $f : \mathcal{X} \to \mathbb{R}$ is the black-box objective function, which is typically expensive to evaluate and lacks gradient information, and $\mathbf{x}^*$ is the global minimum of $f(\mathbf{x})$.

In practice, the optimization process begins with an initial set of points $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_m\}$ randomly sampled from the search space $\subset \mathcal{X}$, along with their corresponding objective values $\{f(\mathbf{x}_1), f(\mathbf{x}_2), \ldots, f(\mathbf{x}_m)\}$. The optimizer iteratively proposes new candidate points $\mathbf{x}_{t+1}$ based on previous points and their objective values, aiming to minimize $f(\mathbf{x})$ by exploring the search space. For example, in BO, given a dataset of points and their objective values $\mathcal{D}_t$ evaluated up to iteration t, $\pi(\mathcal{D}_t) = \arg\max_{\mathbf{x}\in\mathcal{X}} \alpha(\mathbf{x}; \mathcal{D}_t)$ where α represents a particular acquisition function. However, this process is sensitive to hyperparameter choices and choices of the GP and acquisition function.

3.2 Approach: ZeroShotOpt

Goal: We aim to develop a pretrained model that serves as a "plug-and-play" optimizer, capable of outperforming traditional black-box optimizers without hyperparameter tuning.

Challenge: Typically, training a model of this nature requires a large dataset of examples as most approaches are based on supervised learning [42–44]. Therefore, we need expert demonstrations illustrating how to choose evaluation points that minimize functions in low-budget scenarios. Unfortunately, no publicly available dataset provides such demonstrations.

Key idea: Offline RL from synthetic data: Collecting demonstrations for real-world black-box optimization is costly. Instead, we can train our model on large-scale synthetic data. By defining a design space $\mathcal{X}$ and sampling random black-box functions f, we generate cheap synthetic datasets. However, we still need demonstrations of how to select points to evaluate for optimization. When expert demonstrations are unavailable, reinforcement

learning (RL) [45] offers an alternative by optimizing purely from rewards (i.e., function evaluations). However, directly applying RL can be challenging because low-value points are rare, leading to sparse learning signals. To overcome this, we adopt offline RL [28], which learns from fixed demonstration datasets and can improve upon them. Since offline RL can exceed the performance of its training data, it allows us to leverage diverse demonstrations, not just expert ones. We generate demonstrations by using conventional black-box optimizers (e.g., BO) on synthetic problems and training our model with offline RL on this dataset.

3.2.1 Learning to optimize via offline RL

Optimization as sequential decision-making: We frame black-box optimization as a sequential decision-making problem, where the optimizer acts as an agent interacting with the black-box function f. The optimizer observes a set of initial samples:

$$\{(\mathbf{x}_1, f(\mathbf{x}_1)), \ldots, (\mathbf{x}_m, f(\mathbf{x}_m))\},$$

where each $\mathbf{x}_i \in \mathcal{X}$ is randomly sampled from the search space. At each step t, the optimizer observes all previously evaluated points and their function values,

$$\{\mathbf{x}_1, f(\mathbf{x}_1), \ldots, \mathbf{x}_{m+t}, f(\mathbf{x}_{m+t})\},$$

selects the next evaluation point $\mathbf{x}_{m+t+1}$ based on its policy π, receives the function value $f(\mathbf{x}_{m+t+1})$, and repeats this process until the number of steps t hits the user-defined limit. The agent's objective is to find a point $\mathbf{x}$ that minimizes the objective value $f(\mathbf{x})$ over its interactions with the environment.

Offline RL mitigates exploration challenge: By formulating optimization as sequential decision-making, we can apply RL to learn a policy π that minimizes $f(\mathbf{x})$. RL updates its policy based on interactions—sequences of proposed points and their objective values: $\{\mathbf{x}_1, f(\mathbf{x}_1), \ldots, \mathbf{x}_{m+T}, f(\mathbf{x}_{m+T})\}$, where T is the number of additional evaluations after initialization. A key challenge in applying RL to optimization is *exploration*. Early in training, most sampled points may have high objective values, resulting in sparse learning signals for RL. Since RL updates its policy based on rewards, if all points have similarly high objective values, there is little useful information to learn from. To address this challenge, we use *offline RL*, which learns from a static dataset of demonstrations. Because the dataset may already include expert samples, offline RL avoids the cold-start problem. It can then learn a policy that proposes solutions even better than those in the dataset [28], making it a simpler and more efficient alternative to purely online RL.

Implementation: We adapt Decision Transformer (DT) [12], a scalable, transformer-based, offline RL algorithm, to learn an optimizer policy from static datasets. DT is a transformer-based model that takes the history of states, actions, and trajectory quality as input and predicts actions to achieve trajectories of the specified quality. We define trajectory quality based on normalized regret R relative to the set of optimization methods run on the same

function, where f_L^* is the minimum achieved by a set of methods within L steps on a black-box function f and f_m is the median of inital random states:

$$R = \sqrt{(\min(\{f(\mathbf{x}_1), \ldots, f(\mathbf{x}_L)\}) - f_L^*)/(f_m - f_L^*)}.$$

Our adapted DT takes as input past query points $\mathbf{x}_i$ and their function values $f(\mathbf{x}_i)$, along with trajectory regret and length, and predicts the next query point required to generate a trajectory with the given regret and length. DT is trained using the maximum likelihood objective:

$$\max_{\pi} \mathbb{E}_{\mathcal{D}} \left[\log \pi(\mathbf{x}_{t+1} | R, L, \mathbf{x}_1, f(\mathbf{x}_1), \ldots, \mathbf{x}_L, f(\mathbf{x}_L))\right],$$

where $\mathcal{D}$ is a dataset of optimization trajectories.

Like supervised learning, DT learns from demonstrations in static datasets but differs by conditioning on trajectory regret and length. This is crucial for allowing the model to distinguish between effective and ineffective optimization trajectories. At inference time, we specify zero as the desired regret R to initiate generation, encouraging the model to act as the best methods do. We also specify parameter L as the evaluation budget for the method.

3.2.2 Synthetic data generation

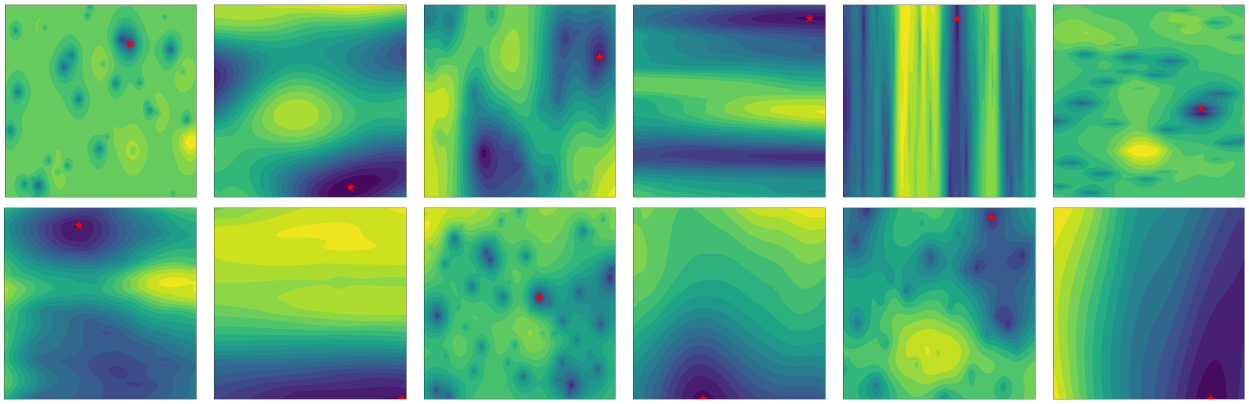


Figure 3.1: 2D synthetic functions generated by our GP-based function generator with various kernels and parameters. The red star is the global minimum and darker color signifies lower function value.

Gaussian process as function generators: Inspired by Chen et al. [32], we use Gaussian Processes (GPs) as flexible function generators. The intuition here is that GP-based BO methods perform well over a wide variety of function spaces. By basing our training data on GP functions as well, we hope to learn a policy that also adapts well to other function spaces. While GPs may not perfectly represent real-world function spaces, our empirical results show they provide a good basis for learning a policy that works on a wide range of function spaces. To introduce variability, we randomize the kernel type, length scale, and other initialization parameters. Given randomly sampled points and objective values, we fit a

GP to obtain the posterior mean, which serves as the objective function. We use the posterior mean to ensure deterministic, smooth training functions with high complexity. This setup efficiently generates diverse and complex training functions. Our implementation, based on GPy [46], randomly selects kernels from RBF, Matern (3/2 and 5/2), Exponential, Cosine, and Quadratic. Figure 3.1 illustrates example generated functions.

Trajectory generation: For each of the 1,600,000 synthetic functions generated ranging from 2 D to 20 D, we run BO with 12 kernel–acquisition variants. These include the Expected Improvement (EI), LogEI [17], Upper Confidence Bound (UCB), Joint Entropy Search (JES), Max-value Entropy Search (MES), and Thompson Sampling (TS) acquisition functions with RBF and Matern kernels. Each model is initialized with 10 random points before iteratively fitting the GP and selecting new points using the acquisition function. We generate 12 trajectories per environment, each consisting of 10 initial samples followed by 40 optimization steps, for a total of 50 evaluations.

3.2.3 Model architecture

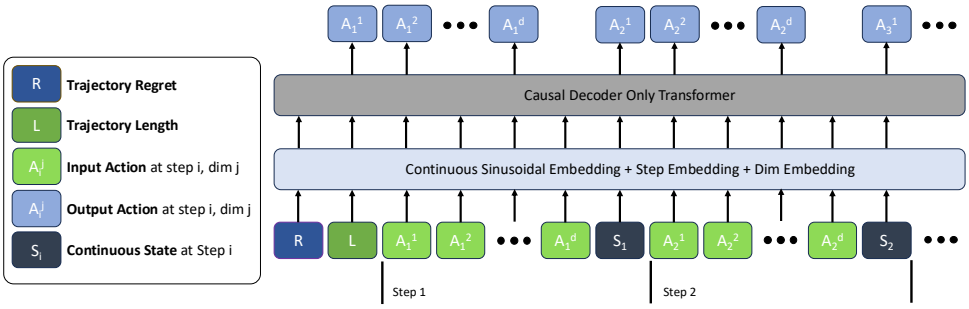


Figure 3.2: Illustration of the ZeroShotOpt architecture. The model embeds continuous inputs using a fixed sinusoidal embedding and 2 learned positional embeddings. The main model is a causal decoder only transformer, trained only on loss on the action space.

Our model takes 4 different types of token inputs: regret, length, action (proposed points), and state (function values). The actions are split into individual tokens for each dimension. These are provided as normalized continuous values to a fixed sinusoidal embedding. We add 2 learned positional embeddings, an embedding of dimension and an embedding of step. These embeddings are passed on to a causal decoder-only transformer model. This is trained using cross-entropy loss solely on binned action values, where actions are split into linearly spaced bins across a single dimension. We train a single model with 200 million parameters on data ranging from 2 D to 20 D, with step counts of 10, 20, 30, or 40 model steps. The main architecture of ZeroShotOpt is illustrated in Figure 3.2.

Chapter 4

Experiments

We conduct extensive experiments aiming to answer the following questions:

1. Does ZeroShotOpt match or surpass state-of-the-art black-box optimizers?
2. Does the pretrained policy *generalize* to unseen out-of-distribution tasks?
3. How competitive is ZeroShotOpt on real-world optimization benchmarks?

4.1 Benchmarks and experimental setup

We test our methodology on a set of in-distribution and out-of-distribution test functions. We utilize our GP function generator to create a test set of diverse GP functions to test our models in-distribution performance. To test out-of-distribution performance, we use the Virtual Library of Simulated Experiments (VLSE) [47] and Black-Box Optimization Benchmark (BBOB) [48]. These both contain synthetic functions that are traditionally used as standard benchmarks for global black-box optimization. We test our method on 2 D, 5 D, 10 D and 20 D function spaces.

To evaluate our model on real-world scenarios, we use the Hyperparameter Optimization Benchmark (HPO-B) [49]. HPO-B contains optimization benchmarks built on real-world experiments from OpenML [50] that have been cleaned up for benchmarking. We show results over 12 of the search spaces, ranging from 2 D to 10 D. We slightly adapt our inference strategy to deal with larger uncertainty for these real-world tests, changing the length input to always be the shortest input.

We utilize the same global optimizers used to generate training data as our baselines. This includes the following acquisition functions: EI, LogEI, UCB, JES, MES, and TS, all with both RBF and Matern kernels. These baselines were implemented in BoTorch [21]. We also compare to other gradient-free optimizers, including Covariance matrix adaptation evolution strategy (CMA-ES), particle swarm optimization (PSO), and differential evolution (DE). We show the performance of each method on these benchmarks using normalized performance. Normalized performance evaluates the performance of each model in relation to the overall regret of the optimization while factoring in the scale of the function. We measure this with $P = (f^* - \min(\{f(\mathbf{x}_1), \ldots, f(\mathbf{x}_L)\}))/(f_m - f^*)$, where f^* is the global minimum and f_m is

the median of the initial randomly sampled points. We report the mean performance over tested functions.

4.2 Performance on synthetic benchmarks

We show the mean normalized performance after 50 total evaluations on three synthetic datasets over various dimensions in Table 4.1. ZeroShotOpt consistently achieves top-tier performance across the GP test suite and out-of-distribution test suites (BBOB and VLSE). This is important because ZeroShotOpt was trained on the GP data, so the BBOB and VLSE tests are completely zero-shot. While GP-LogEI with Matern kernel and GP-EI with Matern kernel perform better than ZeroShotOpt on average, the performance gaps between these methods are minimal across a wide range of functions. Notably, ZeroShotOpt outperforms several other BO variants, including those using UCB, MES, JES, and TS acquisition strategies. Additionally, we see that the BO methods significantly outperform other classes of gradient-free methods, which is expected based on the low evaluation budget. Our evaluation includes a wide array of BO variants, ensuring that the baselines represent a strong state-of-the-art comparison.

We show the performance over different evaluation budgets on the out-of-distribution synthetic tests functions in Table 4.2. We find that performance of ZeroShotOpt remains consistent across different step counts and always aligns closely to the performance of best-performing BO methods. This conclusion generalizes over multiple different dimensions and types of synthetic functions, demonstrating ZeroShotOpt's ability to adapt without any task-specific tuning.

4.3 Performance on real-world benchmarks

We show the mean normalized performance on real-world test functions on 12 of the search spaces from HPO-B ranging from 2 D to 10 D in Table 4.3. ZeroShotOpt remains a consistent top performing method across our range of steps. Notably, ZeroShotOpt *outperforms* the strongest synthetic baselines (GP-LogEI / GP-EI with Matern kernel) at 20, 30, and 40 steps. While the best BO methods for each evaluation continue to show strong results, the ability of ZeroShotOpt to match or closely trail the state-of-the-art BO baselines across a wide array of conditions is evidence of its learned flexibility and robustness, despite not being trained on domain-specific data.

In contrast, learned models in prior works have not demonstrated the same level of robustness across evaluation distributions. Moreover, while the performance of traditional BO baselines is likely to plateau without manual tuning, models built upon ZeroShotOpt offer clear potential for further gains through iterative refinement or domain adaptation. These results demonstrate that ZeroShotOpt effectively transfers to real-world optimization settings, offering strong sample efficiency, robustness across dimensionalities, and practical ease-of-use as a plug-in optimizer.

Table 4.1: Mean normalized performance after 50 total evaluations (10 initial + 40 model steps), averaged over 2 D, 5 D, 10 D and 20 D synthetic functions. We test over 500 functions from each dimension and evaluate mean and standard deviation across 5 independent seed splits.

	GP		**BBOB**		**VLSE**	
Method	Mean ± SD	Rank	Mean ± SD	Rank	Mean ± SD	Rank
GP-LogEI (Matern)	0.633 ± 0.008	**1**	0.915 ± 0.001	**1**	0.809 ± 0.002	**1**
GP-MES (Matern)	0.625 ± 0.010	2	0.906 ± 0.002	8	0.796 ± 0.002	5
ZeroShotOpt	0.624 ± 0.007	3	0.910 ± 0.002	4	0.802 ± 0.002	3
GP-UCB (Matern)	0.615 ± 0.010	4	0.909 ± 0.002	6	0.786 ± 0.003	7
GP-EI (Matern)	0.610 ± 0.009	5	0.912 ± 0.001	2	0.809 ± 0.002	**1**
GP-UCB (RBF)	0.610 ± 0.010	5	0.911 ± 0.002	3	0.785 ± 0.001	8
GP-LogEI (RBF)	0.603 ± 0.009	7	0.910 ± 0.002	4	0.798 ± 0.002	4
GP-EI (RBF)	0.589 ± 0.009	8	0.908 ± 0.002	7	0.795 ± 0.002	6
GP-MES (RBF)	0.594 ± 0.008	9	0.900 ± 0.002	9	0.784 ± 0.002	9
GP-JES (Matern)	0.524 ± 0.006	10	0.843 ± 0.002	13	0.689 ± 0.002	11
GP-JES (RBF)	0.519 ± 0.009	11	0.841 ± 0.003	14	0.680 ± 0.003	12
GP-TS (Matern)	0.516 ± 0.008	12	0.841 ± 0.002	14	0.676 ± 0.002	13
GP-TS (RBF)	0.514 ± 0.008	13	0.844 ± 0.002	12	0.675 ± 0.002	14
PSO	0.471 ± 0.009	14	0.848 ± 0.003	10	0.670 ± 0.001	15
CMA-ES	0.445 ± 0.006	15	0.848 ± 0.001	10	0.749 ± 0.003	10
DE	0.429 ± 0.007	16	0.779 ± 0.002	16	0.567 ± 0.002	16

4.4 Runtime results

We compare the runtime of the best BO method on the synthetic data, GP-LogEI (Matern), to the runtime of ZeroShotOpt across various dimensions in Table 4.4. We report the average runtime per evaluation, measured on an NVIDIA H100 GPU using our GP benchmark suite. The runtime of both ZeroShotOpt and GP-LogEI (Matern) increases with higher dimensions, with the exception of GP-LogEI (Matern) at 20D (due to inherent variability in GP fitting and acquisition optimization, smoother acquisition landscapes, or reduced convergence that occurs in higher dimensions). However, ZeroShotOpt remains substantially faster than GP-LogEI (Matern) across all dimensions.

4.4.1 Inference scaling ablation

We show the performance of different inference strategies using the same model in Figure 4.1. We scale our states between 0 and 1 during training using the highest and lowest values achieved by any method on the function space, but we do not know these values at inference time. Therefore, we need to develop a scaling strategy for inference. We do this by changing the scaling of the input states at step $t + 1$, using parameters C_u and C_l in the following

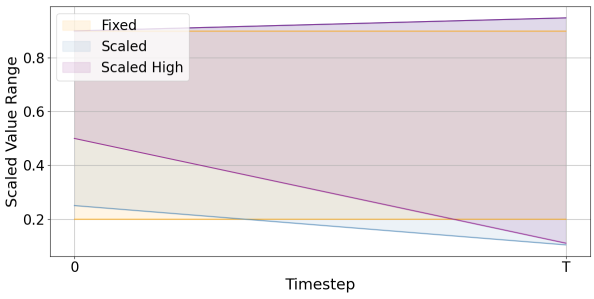
Table 4.2: Mean normalized performance after specified step counts (after 10 initial steps), averaged over 2 D, 5 D, 10 D and 20 D BBOB and VLSE functions. We test over 500 functions from each dimension and evaluate mean and standard deviation across 5 independent seed splits.

	Step 10		**Step 20**		**Step 30**		**Step 40**	
Method	Mean ± SD	Rank	Mean ± SD	Rank	Mean ± SD	Rank	Mean ± SD	Rank
GP-LogEI (Matern)	0.760 ± 0.001	**1**	0.811 ± 0.001	**1**	0.842 ± 0.001	**1**	0.862 ± 0.001	**1**
GP-EI (Matern)	0.759 ± 0.001	2	0.810 ± 0.001	2	0.841 ± 0.001	2	0.860 ± 0.001	2
ZeroShotOpt	0.757 ± 0.001	5	0.808 ± 0.001	3	0.837 ± 0.001	3	0.856 ± 0.001	3
GP-LogEI (RBF)	0.759 ± 0.001	2	0.806 ± 0.001	5	0.835 ± 0.001	4	0.854 ± 0.001	4
GP-EI (RBF)	0.759 ± 0.001	2	0.807 ± 0.001	4	0.834 ± 0.001	5	0.851 ± 0.001	5
GP-MES (Matern)	0.749 ± 0.003	8	0.798 ± 0.001	8	0.830 ± 0.001	6	0.851 ± 0.001	5
GP-UCB (RBF)	0.756 ± 0.001	6	0.804 ± 0.001	6	0.830 ± 0.001	6	0.848 ± 0.001	7
GP-UCB (Matern)	0.753 ± 0.002	7	0.804 ± 0.001	6	0.830 ± 0.002	6	0.848 ± 0.002	7
GP-MES (RBF)	0.744 ± 0.001	9	0.791 ± 0.001	9	0.821 ± 0.001	9	0.842 ± 0.001	9
CMA-ES	0.722 ± 0.001	10	0.757 ± 0.001	10	0.781 ± 0.002	10	0.799 ± 0.002	10
GP-JES (Matern)	0.682 ± 0.002	11	0.720 ± 0.002	11	0.746 ± 0.002	11	0.766 ± 0.002	11
GP-JES (RBF)	0.673 ± 0.001	12	0.712 ± 0.002	12	0.741 ± 0.002	12	0.760 ± 0.002	12
GP-TS (RBF)	0.660 ± 0.001	14	0.708 ± 0.001	13	0.738 ± 0.002	13	0.759 ± 0.001	13
PSO	0.663 ± 0.001	13	0.700 ± 0.002	15	0.731 ± 0.001	15	0.759 ± 0.001	13
GP-TS (Matern)	0.660 ± 0.002	14	0.707 ± 0.002	14	0.738 ± 0.002	13	0.758 ± 0.001	15
DE	0.611 ± 0.002	16	0.634 ± 0.002	16	0.654 ± 0.002	16	0.673 ± 0.002	16

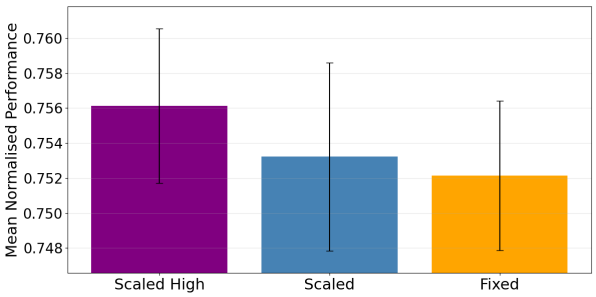
equation:

$$S_i' = (S_i - min(S_0, ..., S_t))/(max(S_0, ..., S_t) - min(S_0, ..., S_t)) * (1 - C_u - C_l) + C_l.$$

We test various methodologies, from fixed values to scaled values that decrease as the trajectory progresses. We show the ranges of scaled values inputed over T timesteps with 3 methodologies in Figure 4.1a and the results of these 3 methodologies in Figure 4.1b, which shows a high C_l that decreases with progression performs the best. However, the model is fairly robust to different initializations. Exact formulations of these inference methods are available in the Appendix.



(a) Scaled range of input over steps for each method.



(b) Results on 2D, 5D, and 10D GP functions.

Figure 4.1: Ablation on scaling strategies during inference.

Table 4.3: Mean normalized performance after specified step counts (after 10 initial steps), averaged over HPO-B functions ranging from 2 D to 10 D. We test over 250 functions from each dataset and evaluate standard deviation across 5 independent seed splits.

	Step 10		**Step 20**		**Step 30**		**Step 40**	
Method	Mean ± SD	Rank	Mean ± SD	Rank	Mean ± SD	Rank	Mean ± SD	Rank
GP-JES (Matern)	0.887 ± 0.003	7	0.924 ± 0.002	2	0.940 ± 0.001	**1**	0.947 ± 0.001	**1**
GP-JES (RBF)	0.892 ± 0.002	4	0.927 ± 0.002	**1**	0.939 ± 0.002	2	0.947 ± 0.002	**1**
GP-MES (RBF)	0.897 ± 0.003	**1**	0.924 ± 0.003	2	0.937 ± 0.002	3	0.943 ± 0.002	3
ZeroShotOpt	0.888 ± 0.002	6	0.920 ± 0.002	4	0.932 ± 0.002	5	0.942 ± 0.002	4
GP-LogEI (RBF)	0.896 ± 0.002	2	0.920 ± 0.003	4	0.933 ± 0.002	4	0.941 ± 0.002	5
GP-LogEI (Matern)	0.890 ± 0.003	5	0.917 ± 0.003	7	0.931 ± 0.003	7	0.940 ± 0.004	6
GP-MES (Matern)	0.893 ± 0.004	3	0.919 ± 0.004	6	0.932 ± 0.004	5	0.939 ± 0.003	7
GP-EI (RBF)	0.886 ± 0.003	8	0.915 ± 0.003	8	0.929 ± 0.003	8	0.937 ± 0.003	8
GP-EI (Matern)	0.885 ± 0.004	9	0.914 ± 0.004	9	0.929 ± 0.004	8	0.936 ± 0.004	9
DE	0.863 ± 0.003	11	0.896 ± 0.004	10	0.915 ± 0.003	10	0.927 ± 0.003	10
GP-UCB (RBF)	0.865 ± 0.005	10	0.895 ± 0.004	11	0.912 ± 0.004	11	0.922 ± 0.003	11
PSO	0.852 ± 0.005	13	0.882 ± 0.003	13	0.901 ± 0.003	12	0.915 ± 0.001	12
GP-TS (RBF)	0.842 ± 0.002	14	0.880 ± 0.003	14	0.898 ± 0.003	14	0.911 ± 0.004	13
GP-UCB (Matern)	0.855 ± 0.004	12	0.883 ± 0.003	12	0.899 ± 0.003	13	0.910 ± 0.002	14
GP-TS (Matern)	0.839 ± 0.002	15	0.879 ± 0.003	15	0.898 ± 0.002	14	0.910 ± 0.003	14
CMA-ES	0.767 ± 0.003	16	0.793 ± 0.003	16	0.815 ± 0.004	16	0.831 ± 0.004	16

Table 4.4: Average runtime (seconds) per evaluation over different dimensions.

Method	**2D**	**5D**	**10D**	**20D**
ZeroShotOpt	0.351	0.930	2.602	8.479
GP-LogEI (Matern)	30.597	50.030	92.865	87.752
Speed-Up Ratio	87.2×	53.8×	35.7×	10.3×

Chapter 5

Conclusion, limitations, and future work

In this work, we introduced ZeroShotOpt, a novel approach for black-box global optimization using a pretrained transformer model trained with offline RL. We formulated the optimization task as a sequential decision-making problem and utilized GPs to create a wide variety of synthetic functions for training. Learning from high-quality trajectories from multiple expert optimizers, our model shows strong performance on unseen optimization problems without the need for task-specific tuning. This work highlights the potential of data-driven pretrained models in advancing global optimization and naturally paves the way for promising future extensions.

Our approach is currently limited to continuous, single-objective optimization, which restricts its applicability to problems involving combinatorial or mixed-integer decision spaces, as well as multi-objective scenarios. Extending the model to tackle these broader categories could significantly enhance its versatility. Additionally, reducing model size through parameter-efficient techniques or pruning could improve deployment efficiency. Finally, although ZeroShotOpt is competitive with *all* baseline BO variants and generally matches the best performing BO variants, it is occasionally outperformed by the methods with the best variants for individual evaluation distributions.

However, there are many advantages of our transformer-based approach in comparison to BO and opportunities for future work. First, if training data is available, ZeroShotOpt can be finetuned to meet the needs of individual circumstances. This could result in further improved performance, even with limited training data. Additionally, ZeroShotOpt could be augmented to include semantic information that is difficult to incorporate with BO-based methods. Including information such as historical data from previous similar tests or parameter names/definitions could provide valuable information that models based on ZeroShotOpt could take advantage of. Lastly, scaling up with additional training data from a broader range of functions, as well as utilizing larger networks, is expected to improve performance. Therefore, ZeroShotOpt provides a base that can be utilized to extend to different contexts or quantities of information in a way that BO cannot. Making these extensions possible is a valuable part of our contributions.

Appendix A

Implementation details

A.1 Data generation

The ability to generate diverse functions, while scaling up in terms of complexity and dimensionality, is crucial to our model's success. Inspired by Chen et al. [32], we use the expressive power of Gaussian Processes (GPs) as a general function generator. Specifically, we randomize the kernel type ($k(\mathbf{x}, \mathbf{x}')$) and its length scale to create variability. We randomly sample n input points $\{\mathbf{x}_i\}_{i=1}^n \subset \mathcal{X}$ within the search space and assign random objective values $\{y_i\}_{i=1}^n$. Using these samples, we fit a GP to obtain the posterior mean $\mu(\mathbf{x})$ and covariance $k(\mathbf{x}, \mathbf{x}')$, resulting in a posterior function that serves as the training function.

To evaluate the function at a specific point $\mathbf{x}$, we perform inference on the GP posterior, yielding a sample from the predictive distribution: $f(\mathbf{x}) \sim \mathcal{N}(\mu(\mathbf{x}), \sigma^2(\mathbf{x}))$, where $\mu(\mathbf{x})$ is the mean of the posterior and $\sigma^2(\mathbf{x})$ is the variance derived from the kernel. We sample a single variance input used over the whole environment to generate a continuous function. This approach allows us to efficiently generate training functions with varied and complex behavior.

In our implementation based on GPy [46], we randomly select the kernel type from the set of RBF, Matern 3/2, Matern 5/2, Exponential, Cosine and Quadratic kernels. We also further randomize the kernel by combining up to two kernels together, by adding or multiplying randomly selected kernels. In total, we include 78 different kernel types by combining kernels. We also randomize the length scale between 0.1 and 10, and the number of input points for fitting the GP between $10 * d$ and $30 * d$, where d is the dimension of the action space. This diverse setup ensures variability and richness in the generated functions. A few examples of a generated functions are shown in Figure 3.1.

We use a variety of global optimization algorithms in each environment to generate expert trajectories. We vary the acquisition function and the kernel used to fit each model. We utilize both RBF and matern kernels for each process, with Expected Improvement (EI), Log Expected Improvement (LogEI), Upper Confidence Bound (UCB), Joint Entropy Search (JES), Max-value Entropy Search (MES), and Thompson Sampling (TS) as acquisition functions. We begin by sampling 10 initial points randomly to initialize each model. We then fit the GP model to the set of points and use the specified acquisition function to sample a prospective point. After evaluating that point, we refit the model and continue iteratively. In

total, we generate 12 trajectories for each environment. These trajectories contain the 10 initial samples and then a series of points generated by each method. We generate trajectories of length 40 steps, not including the initial samples, for a total of 50 evaluations. We use these trajectories as a baseline for our input regret value and include all as training data for our model.

We generate trajectories using CPU machines, primarily on an Intel Xeon Platinum 8260 system. We find that, although BoTorch supports GPU acceleration, the most cost-efficient manner to generate trajectories is to use parallel CPU processes. However, even with optimizations, running our baseline global optimizers is slow, especially for higher dimensions. Therefore, we total $\sim 150,000$ vCPU hours for our data generation. Additionally, we were limited by the amount of trajectories we could generate and more data may help improve the model.

A.2 Model architecture and training

Trajectory quality: Regret is defined as the trajectory quality in our paper. We process the regret by the following transformation for m initial samples and L overall steps:

$$R(\mathbf{x}_1, f(\mathbf{x}_1), \ldots, \mathbf{x}_L, f(\mathbf{x}_L)) = \sqrt{\frac{\min(\{f(\mathbf{x}_1), \ldots, f(\mathbf{x}_L)\}) - f_L^*}{\text{Median}(\{f(\mathbf{x}_1), \ldots, f(\mathbf{x}_m)\}) - f_L^*}} \tag{A.1}$$

Note that this trajectory quality definition differs from that used in Decision Transformer [12], which employs a return-to-go $R_t = \sum_{t'=t}^{T} r_{t'}$ at each step of the trajectory where r_t denotes the reward at times step t. In the context of our sequence, unlike in robotic control or Atari games, the state itself reflects the function value, providing direct information about the return. Thus, there is no need for an additional reward signal that differs from the function value. In other words, the state sufficiently conveys the outcomes of prior actions, giving the agent enough context to effectively propose new actions.

Training: Our full training dataset contains $\sim 20,000,000$ total trajectories. We use data augmentation to expand this and enable further generalization. We augment by swapping axes and flipping the action space to provide additional training data. This augmentation greatly expands the trajectory space, particularly for higher dimensions. We also include shortened trajectories (i.e., the first 20 steps of a 40 step trajectory) to expand the dataset. Overall, Table A.1 contains the number of environments and therefore trajectories of each type before augmentation. We input trajectories of length 10, 20, 30, and 40 to the model, where length does not include the 10 random initial steps.

Our overall training framework is based on the NanoGPT repository [1], which provides a simple and fast GPT implementation, though additional engineering optimizations could further improve efficiency. We adapted this model with our embedding strategy and data pipeline. We train a model with 16 layers, 16 heads, and an embedding dimension of 1024, totaling ~ 200 million parameters. The action space is split into 2,000 evenly spaced bins for our loss calculation. We use trajectories ranging from 2D to 20D and from 10 steps to 40 steps, not including the 10 initial random steps, to train this model. Due to computational

Table A.1: Number of functions and trajectories for each dimension.

Dimension	Functions	Trajectories
2D	500,000	6,000,000
3D	200,000	2,400,000
4D	100,000	1,200,000
5D–20D	50,000 each	600,000 each
Total	**1,600,000**	**19,200,000**

constraints and slow generation of data, this dataset contains more data from lower dimensions and fewer steps.

We use the hyperparameters shown in Table A.2 to train the model. The model is trained using the AdamW optimizer using 4 Nvidia H100 GPUs. In total, training takes ~ 3 days on this system.

Table A.2: Hyperparameters for model training. The training implementation is largely based on NanoGPT [1], with adaptations for our specific architecture and data input.

Hyperparameter	Value
Total Parameters	200 million
Number of Transformer Heads	16
Transformer Embedding Dimension	1024
Transformer Layers	16
Learning Rate	6×10^{-4} with cosine scaling
Weight Decay	1×10^{-1}
Batch Size	1024 trajectories
Total Iterations	250,000
Precision	BF16

A.3 Inference

A.3.1 Overall Methodology

At inference time, we specify zero as the desired regret R to initiate generation, encouraging the agent to act as the best baseline methods do. We provide parameter L as the length of our sequence for synthetic functions, but we find that our model performs best on real-world functions when we input shorter length L during inference time. We then provide the 10 initial samples, as provided to the global optimization methods to initialize the model. To perform a step, the model generates a distribution across bins for the first dimension. We sample from the distribution using top-p sampling with $p = 0.9$ and convert the bin to a continuous action using the center of the range of the selected bin. We iteratively select

actions for each dimension before evaluating the overall action on the environment. We continue to sample and select actions for the specified number of steps.

A.3.2 Inference Scaling

We show the performance of different inference strategies using the same model in Figure 4.1. We scale our states between 0 and 1 during training using the highest and lowest values achieved by any method on the function space, but we do not know these values at inference time. Therefore, we need to develop a scaling strategy for inference. We adjust the scaling of the input states at step $t+1$, using parameters C_u and C_l in the following equation:

$$S_i' = (S_i - min(S_0, ..., S_t))/(max(S_0, ..., S_t) - min(S_0, ..., S_t)) * (1 - C_u - C_l) + C_l.$$

We define C_u and C_l differently for each method, where t is the current step and L is the number of overall steps. We trial multiple different methodologies for scaling states during inference. These methods vary the way we normalize the state values, given we have less information about the sequence at inference time. We detail the methodology used to generate results in Section 4.4.1 here.

Fixed: $C_u : 0.1$, $C_l : 0.2$
Scaled: $C_u : 0.05 + 0.05 * \frac{L-t}{L}$, $C_l : 0.1 + 0.15 * \frac{L-t}{L}$
Scaled High: $C_u : 0.05 + 0.05 * \frac{L-t}{L}$, $C_l : 0.1 + 0.4 * \frac{L-t}{L}$

As shown in Figure 4.1, we find that **Scaled High** methodology produces the best results. Therefore, we use this scaling for all presented results. Keeping our scaling configuration consistent across tests allows for flexible model usage and improved utility of our model.

A.3.3 Inference Runtime

We show complete results comparing the runtime of ZeroShotOpt to our baseline methods in Table A.3. We report the average runtime per evaluation, measured on an NVIDIA H100 GPU using our GP benchmark suite. We see that there is some variance across BO methods due to the nature of different acquisition functions, but ZeroShotOpt is generally faster than most BO methods and can be made more efficient by improved techniques in speeding up transformers. CMA-ES, DE, and PSO are all much faster due to their evolutionary nature, but this comes with much worse performance, which is not preferred for optimizations with low evaluation budgets.

A.4 Model Scaling

We complete an ablation on model size, suggesting that further increasing the size of ZeroShotOpt could further improve results. We compare the performance of ZeroShotOpt with different model sizes in Figure A.1. In particular, we compare our full-size ZeroShotOpt model with 200M parameters with a smaller model with 90M parameters. All training is completed with the same number of iterations and training data. We find that the larger model performs better due to improved capacities, suggesting that further scaling may continue to improve results.

Table A.3: Average runtime (seconds) per evaluation over different dimensions, sorted by 20D runtime.

Method	**2D**	**5D**	**10D**	**20D**
CMA-ES	0.017	0.022	0.040	0.083
PSO	0.032	0.043	0.090	0.208
DE	0.033	0.044	0.091	0.209
GP-JES (RBF)	2.012	1.972	1.888	1.905
GP-JES (Matern)	2.018	2.067	2.061	2.116
ZeroShotOpt	0.351	0.930	2.602	8.479
GP-TS (Matern)	13.041	20.094	24.262	24.326
GP-TS (RBF)	12.460	18.486	26.779	24.905
GP-UCB (Matern)	20.088	31.973	43.356	36.811
GP-UCB (RBF)	19.016	32.738	44.413	37.279
GP-EI (RBF)	15.804	37.344	57.042	46.994
GP-EI (Matern)	17.093	37.621	67.082	54.672
GP-LogEI (Matern)	30.597	50.030	92.865	87.752
GP-MES (RBF)	21.625	54.192	85.005	89.352
GP-LogEI (RBF)	29.871	54.489	85.576	94.265
GP-MES (Matern)	23.166	57.048	86.132	95.136

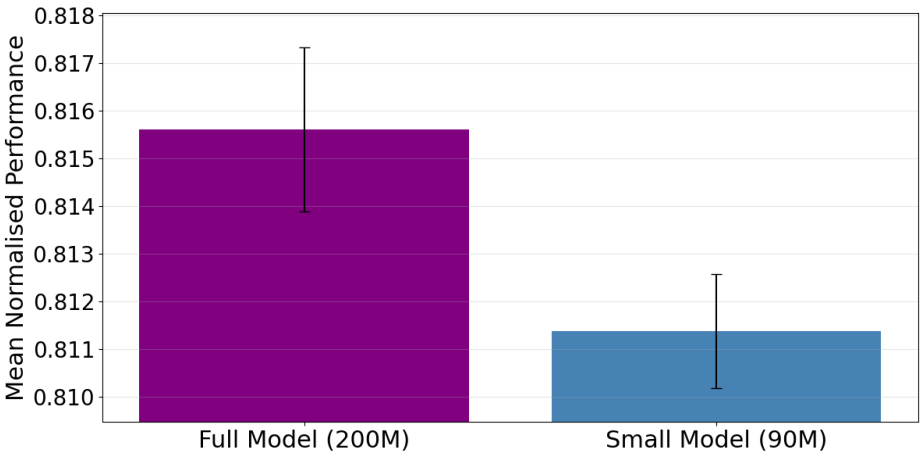


Figure A.1: Ablation on Model Size. Mean normalized performance over steps 10, 20, 30 and 40 on 2 D, 5 D, 10 D and 20 D BBOB and VLSE functions. We test over 500 functions from each dataset and evaluate standard deviation across 5 independent seed splits.

Appendix B

Model Comparison and Evaluation

B.1 Comparison to other learned methods

We include more detailed comparison between ZeroShotOpt and other pretrained optimizers (BONET, RIBBO, and OptFormer) below to further emphasize the differences and impact of ZeroShotOpt.

BONET [14]: BONET applies a causal transformer and RL-inspired pretraining to specific optimization tasks. They introduce a novel approach by synthesizing optimization trajectories by reordering samples in a given dataset, which creates high quality synthetic trajectories. They show promising results on domain-specific extrapolation, outperforming classical methods in their test domain. In comparison to ZeroShotOpt, they focus on domains with higher budgets (in the hundreds) and show results on a small set of high-dimensional problems. Additionally, the study only evaluates domain-specific extrapolation based on a relatively large-scale dataset, instead of tackling unseen optimization problems with just a few dozens of samples. This is an important distinction, as BONET uses more initial points to initialize the model and trains and tests on similar distributions. Therefore, the general problem being solved by BONET has different limitations. Overall, they provide a methodology that advances ideas in transformer-based optimization, but work towards different problems than ones ZeroShotOpt is designed for.

OptFormer [13]: OptFormer is a text-based transformer framework designed for hyperparameter optimization for general problem space beyond continuous. The model is trained on a proprietary hyperparameter optimization database and exploits the textual nature by leveraging hyperparameter names and descriptions. The method shows promise for transformer-based models trained on large datasets by outperforming solutions such as BO on the given test cases, which are drawn from similar distributions as training data. They train a model on a proprietary dataset, BBOB, and HPO-B data. This shows strong results on the test sets of each respective dataset. As their dataset contains a diverse set of functions and their model shows the ability to learn a variety of different algorithms, their contributions show promise towards the idea of a general-purpose transformer-based optimizer. However, the model shows significant performance degradation when tested on distributions different

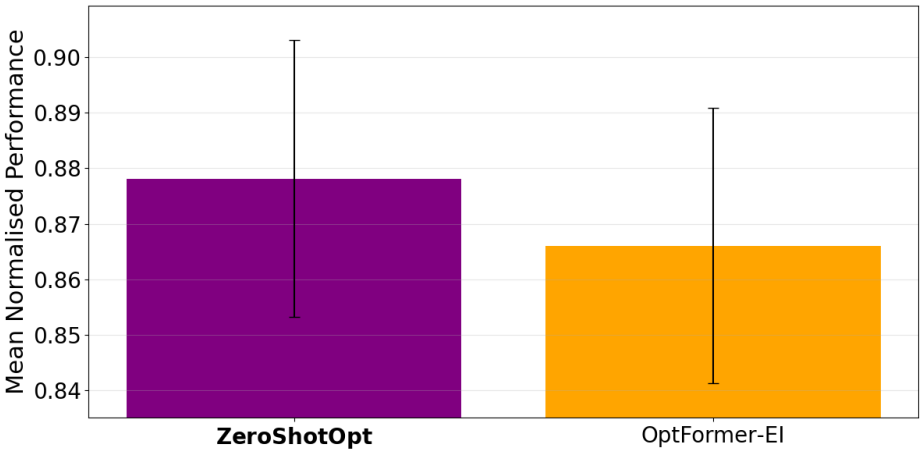


Figure B.1: Evaluation Comparison with OptFormer. Mean normalized performance over steps 10, 20, 30 and 40 on 2 D, 5 D, 10 D and 20 D BBOB functions. We test over 24 functions from each dataset and evaluate standard deviation across 4 independent seed splits.

from its training data. Overall, its strong performance on a large dataset marks an important step towards general-purpose transformer-based optimization, but does not truly work as a general purpose optimizer.

We utilize the pretrained model and code provided by OptFormer to compare to our results on BBOB data. Although we used the default parameters and setup provided in the original documentation, we find the OptFormer model very slow on our hardware, which limited the scope of our experiments. Therefore, we test one environment for each of the 24 BBOB categories for each dimension in 2 D, 5 D, 10 D and 20 D. Access to TPU machines may negate these issues. We choose the BBOB tests because although they are contained within the OptFormer training data, we can create general synthetic tests. Therefore, we can limit the impact of including exactly the same data in our test set as OptFormer was trained on. We show the results of our experiments in Figure B.1. We find that ZeroShotOpt outperforms OptFormer, although there is a high standard deviation due to the limited number of tests. Additionally, it is important to note that OptFormer includes BBOB tests within its training data, whereas ZeroShotOpt does not. OptFormer trains on 16 out of the 24 BBOB classes, while specifying 5 examples for out-of-distribution tests. Therefore, this shows the effectiveness of ZeroShotOpt as a general-purpose optimizer that can surpass previous learned optimizers, even when they are trained on task-specific data.

RIBBO [41]: RIBBO is a transformer-based framework trained using offline reinforcement learning. While similarly inspired by decision transformer, RIBBO is trained on the benchmark datasets taken from existing optimization tasks. RIBBO shows strong performance and is able to outperform the algorithms it was trained on on many of its test datasets, which is an important contribution. This shows the promising aspect of conditioned transformed-based models. However, RIBBO uses fixed dimensionality, an inflexible tokenization scheme, and a relatively small scale training scheme. Therefore, RIBBO can only evaluate results for small training and evaluation datasets from similar distributions. This differs from ZeroShotOpt,

which is designed to be scalable and adapt to different dimension counts and function environments. This scalable nature and use of competitive data generation methods on synthetic functions allows ZeroShotOpt to generalize to different distributions, a feature which is minimal within RIBBO. However, the ideas of conditioned transformer-based models and the improved performance RIBBO shows over baselines on individual datasets provides a step towards outperforming BO in transformer-based optimization.

B.2 Benchmarks

We test ZeroShotOpt on various synthetic and real-world benchmarks to understand the model's capabilities and performance over a wide range of functions. We detail these benchmarks below.

GP: We use the function generator used for generating training data as the initial baseline for our method. This contains diverse GP functions over a flexible dimension.

BBOB [48]: The Black-Box Optimization Benchmarking suite was developed as part of the COCO (Comparing Continuous Optimizers) platform to provide a rigorous and standardized environment for evaluating continuous, unconstrained optimization algorithms. BBOB includes 24 benchmark functions that represent a wide range of challenges encountered in real-world optimization, such as separability, multimodality, ill-conditioning, and non-convexity. Each function is parameterized with randomized shifts, scalings, and rotations to prevent algorithms from overfitting to specific patterns. The suite was carefully designed through mathematical constructions and transformations of base functions to create controlled yet diverse test cases. It supports varying dimensions and is widely used in the black-box optimization community.

VLSE [47]: The Virtual Library of Simulation Experiments is a benchmark suite aimed at simulating real-world optimization problems where the objective function is defined by computational simulations rather than closed-form expressions. We use the optimization test problems from this set.

HPO-B [49]: The Hyperparameter Optimization Benchmark was created to support fast and reproducible evaluation of hyperparameter optimization methods by providing surrogate models that approximate the behavior of real machine learning training processes. Built from large-scale logging of real hyperparameter tuning runs on algorithms like XGBoost, SVMs, feedforward neural networks, and others across multiple datasets, HPO-B allows researchers to benchmark optimization strategies without incurring the computational cost of retraining models for each evaluation. It enables fair comparisons across different optimizers under controlled experimental conditions. There are a few different evaluation sets provided by HPO-B. We utilize the HPO-B v2 set designed for non-transfer learning methods.

B.3 Baselines

We utilize the same global optimizers used to generate training data as our baselines. This includes the following acquisition functions for BO:

- **Expected Improvement (EI)**: Selects point that is expected to improve upon the current best observation, balancing exploration and exploitation.
- **Log Expected Improvement (LogEI)**: A variant of EI that operates in log space, making it more suitable for objectives with large dynamic ranges or multiplicative noise.
- **Upper Confidence Bound (UCB)**: Prioritizes points with high predicted mean and uncertainty, controlled by a trade-off parameter.
- **Joint Entropy Search (JES)**: Reduces uncertainty about both the location and value of the global minimum by selecting the point expected to maximally reduce the joint entropy of the posterior over the minimum.
- **Max-value Entropy Search (MES)**: Reduces uncertainty about the value of the global minimum by selecting query points that are expected to most reduce the entropy of its posterior distribution.
- **Thompson Sampling (TS)**: Samples functions from the posterior and optimizes them directly, encouraging diverse sampling over time.

Each acquisition function is used with both RBF and Matern kernels, providing a good baseline across different smoothness assumptions. We use the default parameterizations of these methods from BoTorch [21], covering a broad range of BO variants.

We also compare to other gradient-free global optimizers:

- **Covariance Matrix Adaptation Evolution Strategy (CMA-ES)**: An evolutionary algorithm that adapts the sampling distribution using covariance information for efficient search.
- **Particle Swarm Optimization (PSO)**: A population-based stochastic optimizer inspired by the social behavior of birds and fish, adjusting candidate solutions based on personal and global bests.
- **Differential Evolution (DE)**: A simple yet powerful method that perturbs candidate solutions using scaled differences between population members.

These classical methods typically require thousands of iterations to converge but provide a strong point of comparison to highlight the performance of our model.

Bibliography

[1] Andrej Karpathy. NanoGPT. https://github.com/karpathy/nanoGPT, 2022.

[2] John H Holland. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press, 1992.

[3] Scott Kirkpatrick, C Daniel Gelatt Jr, and Mario P Vecchi. Optimization by simulated annealing. *science*, 220(4598):671–680, 1983.

[4] Nikolaus Hansen and Andreas Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary computation*, 9(2):159–195, 2001.

[5] Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P Adams, and Nando De Freitas. Taking the human out of the loop: A review of bayesian optimization. *Proceedings of the IEEE*, 104(1):148–175, 2015.

[6] Timothy Erps, Michael Foshey, Mina Konaković Luković, Wan Shou, Hanns Hagen Goetzke, Herve Dietsch, Klaus Stoll, Bernhard von Vacano, and Wojciech Matusik. Accelerated discovery of 3d printing materials using data-driven multiobjective optimization. *Science advances*, 7(42):eabf7435, 2021.

[7] Ryan-Rhys Griffiths and José Miguel Hernández-Lobato. Constrained bayesian optimization for automatic chemical design using variational autoencoders. *Chemical science*, 11 (2):577–586, 2020.

[8] Ahmed Alaa and Mihaela Schaar. Autoprognosis: Automated clinical prognostic modeling via bayesian optimization with structured kernel learning. In *International conference on machine learning*, pages 139–148. PMLR, 2018.

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

[10] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 5998–6008, 2017.

[11] Xingyou Song, Oscar Li, Chansoo Lee, Daiyi Peng, Sagi Perel, Yutian Chen, et al. Omnipred: Language models as universal regressors. *arXiv preprint arXiv:2402.14547*, 2024.

[12] Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Misha Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. Decision transformer: Reinforcement learning via sequence modeling. *Advances in neural information processing systems*, 34: 15084–15097, 2021.

[13] Yutian Chen, Xingyou Song, Chansoo Lee, Zi Wang, Richard Zhang, David Dohan, Kazuya Kawakami, Greg Kochanski, Arnaud Doucet, Marc'aurelio Ranzato, et al. Towards learning universal hyperparameter optimizers with transformers. *Advances in Neural Information Processing Systems*, 35:32053–32068, 2022.

[14] Siddarth Krishnamoorthy, Satvik Mehul Mashkaria, and Aditya Grover. Generative pretraining for black-box optimization. *arXiv preprint arXiv:2206.10786*, 2022.

[15] Alexandre Maraval, Matthieu Zimmer, Antoine Grosnit, and Haitham Bou Ammar. End-to-end meta-bayesian optimisation with transformer neural processes. *Advances in Neural Information Processing Systems*, 36, 2024.

[16] Xingyou Song, Yingtao Tian, Robert Tjarko Lange, Chansoo Lee, Yujin Tang, and Yutian Chen. Position paper: Leveraging foundational models for black-box optimization: Benefits, challenges, and future directions. *arXiv preprint arXiv:2405.03547*, 2024.

[17] Sebastian Ament, Samuel Daulton, David Eriksson, Maximilian Balandat, and Eytan Bakshy. Unexpected improvements to expected improvement for bayesian optimization. *Advances in Neural Information Processing Systems*, 36:20577–20612, 2023.

[18] Daniel Golovin, Benjamin Solnik, Subhodeep Moitra, Greg Kochanski, John Karro, and David Sculley. Google vizier: A service for black-box optimization. In *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1487–1495, 2017.

[19] Xingyou Song, Qiuyi Zhang, Chansoo Lee, Emily Fertig, Tzu-Kuo Huang, Lior Belenki, Greg Kochanski, Setareh Ariafar, Srinivas Vasudevan, Sagi Perel, et al. The vizier gaussian process bandit algorithm. *arXiv preprint arXiv:2408.11527*, 2024.

[20] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. *Advances in neural information processing systems*, 25, 2012.

[21] Maximilian Balandat, Brian Karrer, Daniel R. Jiang, Samuel Daulton, Benjamin Letham, Andrew Gordon Wilson, and Eytan Bakshy. BoTorch: A Framework for Efficient Monte-Carlo Bayesian Optimization. In *Advances in Neural Information Processing Systems 33*, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/f5b1b89d98b7286673128a5fb112cb9a-Abstract.html.

[22] Yunsheng Tian, Mina Konaković Luković, Timothy Erps, Michael Foshey, and Wojciech Matusik. Autooed: Automated optimal experiment design platform. *arXiv preprint arXiv:2104.05959*, 2021.

[23] Marius Lindauer, Katharina Eggensperger, Matthias Feurer, André Biedenkapp, Difan Deng, Carolin Benjamins, Tim Ruhkopf, René Sass, and Frank Hutter. Smac3: A versatile bayesian optimization package for hyperparameter optimization. *Journal of Machine Learning Research*, 23(54):1–9, 2022. URL http://jmlr.org/papers/v23/21-0888.html.

[24] Alexander Cowen-Rivers, Wenlong Lyu, Rasul Tutunov, Zhi Wang, Antoine Grosnit, Ryan-Rhys Griffiths, Alexandre Maravel, Jianye Hao, Jun Wang, Jan Peters, and Haitham Bou Ammar. Hebo: Pushing the limits of sample-efficient hyperparameter optimisation. *Journal of Artificial Intelligence Research*, 74, 07 2022.

[25] Huaijun Jiang, Yu Shen, Yang Li, Beicheng Xu, Sixian Du, Wentao Zhang, Ce Zhang, and Bin Cui. Openbox: A python toolkit for generalized black-box optimization. *Journal of Machine Learning Research*, 25(120):1–11, 2024.

[26] Taku Yamagata, Ahmed Khalil, and Raul Santos-Rodriguez. Q-learning decision transformer: Leveraging dynamic programming for conditional sequence modelling in offline rl. In *International Conference on Machine Learning*, pages 38989–39007. PMLR, 2023.

[27] Yueh-Hua Wu, Xiaolong Wang, and Masashi Hamaya. Elastic decision transformer. *Advances in Neural Information Processing Systems*, 36, 2024.

[28] Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems, 2020. URL https://arxiv.org/abs/2005.01643.

[29] Ke Li and Jitendra Malik. Learning to optimize. *arXiv preprint arXiv:1606.01885*, 2016.

[30] Marcin Andrychowicz, Misha Denil, Sergio Gomez, Matthew W Hoffman, David Pfau, Tom Schaul, Brendan Shillingford, and Nando De Freitas. Learning to learn by gradient descent by gradient descent. *Advances in neural information processing systems*, 29, 2016.

[31] Tianlong Chen, Xiaohan Chen, Wuyang Chen, Howard Heaton, Jialin Liu, Zhangyang Wang, and Wotao Yin. Learning to optimize: A primer and a benchmark. *Journal of Machine Learning Research*, 23(189):1–59, 2022.

[32] Yutian Chen, Matthew W Hoffman, Sergio Gómez Colmenarejo, Misha Denil, Timothy P Lillicrap, Matt Botvinick, and Nando Freitas. Learning to learn without gradient descent by gradient descent. In *International Conference on Machine Learning*, pages 748–756. PMLR, 2017.

[33] Sofian Chaybouti, Ludovic Dos Santos, Cedric Malherbe, and Aladin Virmaux. Meta-learning of black-box solvers using deep reinforcement learning. In *NeurIPS 2022, MetaLearn Workshop*, 2022.

[34] Xiaobin Li, Kai Wu, Yujian Betterest Li, Xiaoyu Zhang, Handing Wang, and Jing Liu. Pretrained optimization model for zero-shot black box optimization. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

[35] Samuel Müller, Matthias Feurer, Noah Hollmann, and Frank Hutter. Pfns4bo: In-context learning for bayesian optimization. In *International Conference on Machine Learning*, pages 25444–25470. PMLR, 2023.

[36] Zi Wang, George E Dahl, Kevin Swersky, Chansoo Lee, Zachary Nado, Justin Gilmer, Jasper Snoek, and Zoubin Ghahramani. Pre-trained gaussian processes for bayesian optimization. *Journal of Machine Learning Research*, 25(212):1–83, 2024.

[37] Michael Volpp, Lukas P Fröhlich, Kirsten Fischer, Andreas Doerr, Stefan Falkner, Frank Hutter, and Christian Daniel. Meta-learning acquisition functions for transfer learning in bayesian optimization. *arXiv preprint arXiv:1904.02642*, 2019.

[38] Bing-Jing Hsieh, Ping-Chun Hsieh, and Xi Liu. Reinforced few-shot acquisition function learning for bayesian optimization. *Advances in Neural Information Processing Systems*, 34:7718–7731, 2021.

[39] 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.

[40] Tennison Liu, Nicolás Astorga, Nabeel Seedat, and Mihaela van der Schaar. Large language models to enhance bayesian optimization. *arXiv preprint arXiv:2402.03921*, 2024.

[41] Lei Song, Chenxiao Gao, Ke Xue, Chenyang Wu, Dong Li, Jianye Hao, Zongzhang Zhang, and Chao Qian. Reinforced in-context black-box optimization. *arXiv preprint arXiv:2402.17423*, 2024.

[42] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[43] Florian Bordes, Richard Yuanzhe Pang, Anurag Ajay, Alexander C Li, Adrien Bardes, Suzanne Petryk, Oscar Mañas, Zhiqiu Lin, Anas Mahmoud, Bargav Jayaraman, et al. An introduction to vision-language modeling. *arXiv preprint arXiv:2405.17247*, 2024.

[44] Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan Foster, Grace Lam, Pannag Sanketi, et al. Openvla: An open-source vision-language-action model. *arXiv preprint arXiv:2406.09246*, 2024.

[45] Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4:237–285, 1996.

[46] GPy. GPy: A gaussian process framework in python. http://github.com/SheffieldML/GPy, 2012.

[47] S. Surjanovic and D. Bingham. Virtual library of simulation experiments: Test functions and datasets. Retrieved September 20, 2024, from http://www.sfu.ca/~ssurjano, 2013.

[48] Ouassim Elhara, Konstantinos Varelas, Duc Nguyen, Tea Tusar, Dimo Brockhoff, Nikolaus Hansen, and Anne Auger. Coco: the large scale black-box optimization benchmarking (bbob-largescale) test suite. *arXiv preprint arXiv:1903.06396*, 2019.

[49] Sebastian Pineda Arango, Hadi S Jomaa, Martin Wistuba, and Josif Grabocka. Hpo-b: A large-scale reproducible benchmark for black-box hpo based on openml. *arXiv preprint arXiv:2106.06257*, 2021.

[50] Joaquin Vanschoren, Jan N Van Rijn, Bernd Bischl, and Luis Torgo. Openml: networked science in machine learning. *ACM SIGKDD Explorations Newsletter*, 15(2):49–60, 2014.