POP: PRIOR-FITTED FIRST-ORDER OPTIMIZATION POLICIES

Anonymous authors
Paper under double-blind review

ABSTRACT

Adaptive first-order optimizers rely on fixed, hand-designed mechanisms for incorporating optimization history, limiting how flexibly they can adapt to a new objective. We introduce **POP**, a **p**rior-fitted first-order **op**timizer whose learned action is deliberately restricted to trajectory-conditioned coordinate-wise step sizes along the descent direction. POP is meta-trained via reinforcement learning on millions of synthetic optimization problems sampled from a task-independent compositional prior with Gaussian process components, using scale-invariant policy inputs to support transfer across objectives. Despite training only on low-dimensional synthetic problems, POP generalizes to longer horizons, higher-dimensional objectives, and unseen problem distributions. On a benchmark of 43 diverse optimization functions, POP significantly outperforms per-function tuned gradient-based baselines and transfers zero-shot to practical optimization tasks on real-world datasets. Our work provides a proof of concept that transferable optimization behavior can be learned through trajectory-conditioned step-size adaptation alone, and that such behavior can emerge from meta-training exclusively on objectives sampled from a synthetic generative prior.

1 INTRODUCTION

Gradient- and momentum-based methods are widely used for continuous optimization due to their simple update rules and strong empirical performance (Robbins & Monro, 1951; Kingma & Ba, 2015; Daoud et al., 2022; Liu et al., 2020). Despite their success, their performance can be highly sensitive to hyperparameter choices, particularly the learning rate schedule and momentum coefficients (Tian et al., 2023; Keskar & Socher, 2017; Hassan et al., 2022). Identifying suitable hyperparameters often requires extensive task-specific tuning, which can be computationally costly and difficult to transfer across tasks and domains (Mary et al., 2025; Liao et al., 2022; Schlotthauer et al., 2025).

More recently, learned optimizers have emerged as an alternative to manually designed optimization algorithms, with the goal of learning optimization strategies through meta-training (Tang & Yao, 2024; Metz et al., 2022; Cao et al., 2019; Vishnu et al., 2020; Goldie et al., 2024; Lan et al., 2023). Existing approaches range from directly learning parameter-update rules (Andrychowicz et al., 2016; Wichrowska et al., 2017; Metz et al., 2020) to augmenting hand-designed optimizers with learned components (Kristiansen et al., 2024). Many recent learned optimizers are meta-trained on neural-network task distributions related to their downstream use (Metz et al., 2020; 2022), while transfer from synthetic meta-training tasks to neural-network optimization has also been demonstrated (Wichrowska et al., 2017). However, rather than learning an unrestricted parameter-update rule, we deliberately restrict the learned action to positive step sizes along the negative-gradient direction, conditioned on the optimization trajectory. Our central question is: *Can an optimizer that learns only trajectory-conditioned step sizes along the negative gradient, meta-trained solely on samples from a synthetic generative prior, transfer zero-shot to unseen optimization problems and remain competitive with per-task-tuned optimizers?*

We investigate this question through **POP**, a ***p**rior-fitted* first-order **op**timizer that is meta-trained exclusively on millions of synthetic objective functions sampled from a task-independent prior and subsequently applied zero-shot to unseen optimization problems. We deliberately meta-train in a controlled proof-of-concept setting based on low-dimensional continuous non-convex objectives to test how far optimization behavior learned through this restricted step-size policy can transfer beyond

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

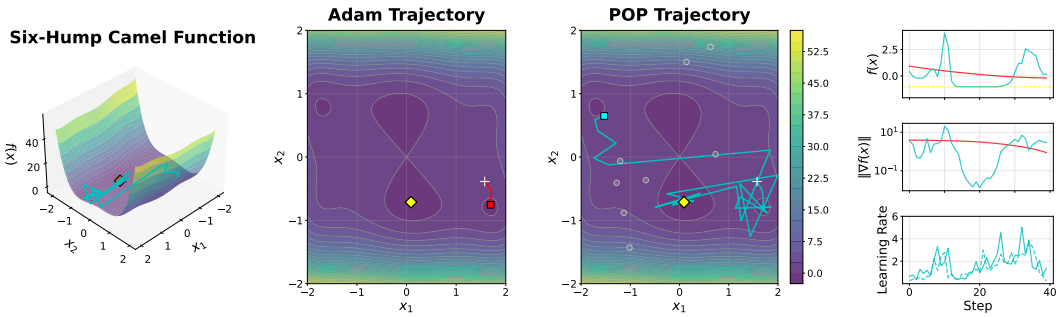


Figure 1: POP (blue) adapts its learning rate based on the optimization landscape, enabling rapid convergence, escape from local minima, and improved global optimization compared to Adam (red). Yellow diamond represents the global minimum, the white cross represents the start state, and the square represents the end state.

the distribution encountered during meta-training. POP represents the optimization as a trajectory-conditioned policy: given the optimization history, it predicts positive coordinate-wise step sizes that adapt the magnitude of the gradient update. We meta-train this policy using reinforcement learning (RL) with optimization-specific reward shaping, while a transformer-based backbone (Vaswani et al., 2017) encodes the optimization trajectory.

We illustrate the resulting trajectory-conditioned behavior exemplarily in Figure 1 on the Six-Hump Camel function (Surjanovic & Bingham). POP adapts its step sizes throughout optimization, using larger updates to move rapidly across the landscape and smaller updates to refine promising solutions. After reaching a local basin, the policy can increase its step size again to explore other regions of the objective landscape. Importantly, this behavior is not specified through a hand-designed learning rate schedule, but emerges from meta-training across the diverse optimization problems sampled from our synthetic prior.

We evaluate POP's generalization beyond synthetic optimization problems encountered during meta-training. We first test on unseen objectives sampled from the same prior, and then examine transfer to longer optimization horizons. We further evaluate POP zero-shot on 43 diverse functions from the *Virtual Library of Simulation Experiments* (Surjanovic & Bingham) benchmark, including 20 dimension-scalable functions evaluated at 32D, and motivate real-world transfer on Gaussian-process hyperparameter optimization and small neural-network training tasks using real-world datasets. Across these settings, POP shows strong performance compared to conventional optimization baselines, including baselines tuned specifically for the respective downstream tasks. Overall, these results provide evidence that a trajectory-conditioned policy restricted to adapting step sizes along the negative gradient can acquire transferable optimization behavior from a low-dimensional, task-independent synthetic prior.

The key contributions of this paper are:

- **Trajectory-conditioned first-order adaptation:** POP learns positive coordinate-wise step sizes along the negative-gradient direction from the optimization trajectory.
- **Prior-fitted first-order policy:** We meta-train POP exclusively on objectives sampled from a task-independent compositional prior and apply it zero-shot without downstream-specific adaptation.
- **Generalization beyond meta-training:** A single policy trained on low-dimensional synthetic objectives transfers to longer horizons, unseen benchmark distributions, higher dimensions, and practical optimization tasks.

2 Related Work

Optimizers. First-order optimization methods such as *Gradient Descent (GD)* (Cauchy, 1847), *Adam* (Kingma & Ba, 2015), and *Lion* (Chen et al., 2023) are widely used for continuous optimization, but rely on hand-designed update and adaptation mechanisms. Quasi-Newton methods such as *L-BFGS* (Liu & Nocedal, 1989) instead incorporate approximate curvature information to improve local optimization. Beyond gradient-based methods, derivative-free approaches such as evolutionary

strategies (Schwefel, 1995) and Bayesian optimization (Bergstra et al., 2011) provide alternative mechanisms for global search. POP remains a first-order gradient-based method, but replaces a hand-designed adaptive step-size mechanism with a meta-learned, trajectory-conditioned policy that predicts coordinate-wise step sizes from the optimization history.

Learned Optimizers. *Learning to Optimize* (L2O) learns components of optimization algorithms across distributions of downstream-specific optimization tasks (Chen et al., 2021; Tang & Yao, 2024). Early methods use recurrent networks to map per-coordinate gradient histories to parameter updates (Andrychowicz et al., 2016), while later work introduces hierarchical architectures that exchange information across parameters and tensors to improve scalability and transfer (Wichrowska et al., 2017; Metz et al., 2022; Moudgil et al., 2025). More constrained approaches instead learn an optimizer's hyperparameters, including learning rate schedules (Xu et al., 2019; Shu et al., 2020), or structured transformations such as learned preconditioners (Gärtner et al., 2023). Meta-training methods include differentiation through unrolled optimization (Wichrowska et al., 2017), evolution strategies (Metz et al., 2022; Moudgil et al., 2025), and RL based policy search (Li & Malik, 2017). Meta-training distributions likewise range from curated synthetic optimization problems (Wichrowska et al., 2017) to broad collections of neural network training tasks (Metz et al., 2022; Moudgil et al., 2025) and narrowly specialized downstream domains (Kristiansen et al., 2024). The closest synthetic meta-training setup learns a general parameter-update rule, whereas POP restricts the learned action to positive coordinate-wise step sizes along the descent direction (Wichrowska et al., 2017)[1]. Its training suite spans both convex and non-convex objectives, including a subset of functions drawn from the same source as our VLSE evaluation. In contrast, POP samples objectives from a compositional generative prior that includes GP-based landscape components. A key distinction is that the step length in the respective work is adapted through relative changes for stability, whereas POP directly predicts trajectory-conditioned step sizes while keeping the descent direction fixed.

Prior-fitted Models. Prior-Data Fitted Networks (PFNs) meta-learn models from synthetic data sampled from a specified generative prior (Müller et al., 2022; Hollmann et al., 2025). This idea has also been applied to Bayesian optimization, where the prior-fitted model acts as a learned surrogate (Müller et al., 2023). POP instead prior-fits the optimizer policy itself, directly predicting first-order optimization steps from trajectory context.

3 POP: Prior-fitted Optimizer

3.1 Problem Setting

In the following, we consider unconstrained optimization problems of the form $x^* \in \operatorname{argmin}_{x\in\mathbb{R}^d} f(x)$, where each task instance is an objective function f sampled from a prior $f \sim p(f)$ over the space of compact functions. We refer to an episode as optimizing a single sampled function f for a fixed budget of T time steps. We define an optimization trajectory as:

Definition 3.1 (Optimization trajectory)**.** For an objective function f, we define the optimization trajectory that includes the first t update steps of a maximum budget of T updates as:

$$\tau^{(t)} \coloneqq \tau^{(0)} \cup \left\{ \left(x_i, y_i, \nabla f(x_i), \frac{i}{T} \right) \right\}_{i=1}^{t}, \tag{1}$$

where $\tau^{(0)}$ denotes an initial context of function observations collected before the first optimization update, x_i denotes the parameters of the optimization problem in step i, while $y_i \coloneqq f(x_i)$ is the function value, $\nabla f(x_i)$ the gradient information, and $\frac{i}{T}$ the normalized timestep.

First-order optimizers proceed by applying an update rule based on local gradient information, e.g., GD applies $x_t = x_{t-1} - \eta_t \nabla f(x_{t-1})$, where η_t is the step size at iteration t. In this work, we replace a preset step size schedule by a meta-learned policy $\eta_t \sim \pi_\theta(\eta_t | \tau^{(t-1)})$, where at each iteration, our optimizer outputs the next step size η_t via a state-conditioned control signal $\tau^{(t-1)}$ based on the optimization trajectory (Definition 3.1) up to step $t-1$.

[1] No public checkpoint or implementation is available for comparison.

3.2 MDP FOR OPTIMIZATION

We formulate the optimization problem as a Markov Decision Process (MDP) defined by the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, r, p_0, \gamma)$, where $\mathcal{S}$ are the states, $\mathcal{A}$ the actions, $\mathcal{P}$ the transition dynamics, $r : \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is the reward signal, p_0 is the probability distribution of initial elements in the optimization trajectory, and γ denotes the discount factor. Our goal is to train a parameterized policy $\pi_\theta : \mathcal{S} \times \mathcal{A} \to \mathbb{R}^+$.

In our meta-learning setting, an episode corresponds to a full optimization trajectory τ^T, where we omit the superscript for brevity, on a sampled task instance from a prior $f \sim p(f)$. We formulate optimization as a sequential decision problem (SDP), in which, at each step t, the policy receives the optimization trajectory $\tau^{(t-1)}$ observed so far and outputs a continuous coordinate-wise step-size action $a_t \in \mathbb{R}^d_{>0}$. The state $s_t \in \mathcal{S}$ is simply the trajectory of updates so far $s_t \coloneqq \tau^{(t-1)}$, for $t \in [1, \dots, T]$.

Optimization Step. Given an optimization trajectory $\tau^{(t-1)}$, the policy outputs $\mu_\theta(\tau^{(t-1)}) \in \mathbb{R}^d$ and samples coordinate-wise step sizes η_t from a log-normal distribution with standard deviation $\sigma_\theta \in \mathbb{R}^d$. The log-normal parameterization ensures positive step sizes and thus updates along the negative gradient direction.

$$\pi_\theta\left(\eta_t \mid \tau^{(t-1)}\right) \sim \text{LogNormal}\left(\mu_\theta\left(\tau^{(t-1)}\right), \text{diag}\left(\sigma_\theta^2\right)\right) \tag{2}$$

POP performs optimization in three steps. First, it samples a coordinate-wise step size from the policy network (Equation 3). Then, POP conducts a gradient update (Equation 4), and finally, it updates the optimization trajectory (Equation 5).

$$\eta_t \sim \pi_\theta\left(\eta_t \mid \tau^{(t-1)}\right) \tag{3}$$

$$x_t \leftarrow x_{t-1} - \eta_t \nabla f(x_{t-1}) \tag{4}$$

$$\tau^{(t)} \leftarrow \tau^{(t-1)} \cup \left\{\left(x_t, y_t, \nabla f(x_t), \frac{t}{T}\right)\right\} \tag{5}$$

Reward Signal. The reward function for our optimization is defined as the improvement to the best observed function value so far in a trajectory. Let $y^*_{t-1} = \min_{(\cdot, y, \cdot, \cdot) \in \tau^{(t-1)}} y$, we define:

$$R\left(y_t, \tau^{(t-1)}\right) = \max\left(0, y^*_{t-1} - y_t\right). \tag{6}$$

This reward assigns a positive signal only when the optimization step yields a new observed minimum, directly linking policy optimization to objective improvement. Since non-improving steps receive no penalty, the agent can explore the function freely, and meaningful improvements guide learning.

3.3 META-LEARNING OBJECTIVE

We optimize the policy parameters θ to maximize the expected return over tasks sampled from our prior defined in Section 3.4. We formulate the objective as follows:

$$\max_\theta \; \mathbb{E}_{f \sim p(f)} \, \mathbb{E}_{\tau \sim \text{POP}(f;\theta)} \sum_{t=1}^{T} \gamma^t R\left(y_t, \tau^{(t-1)}\right), \tag{7}$$

where $\tau^T \sim \text{POP}(f; \theta)$ is computed by running Equations 3-5 for T steps on the function f sampled from the prior $p(f)$. We train the policy parameters using the Proximal Policy Optimization (PPO) algorithm (Schulman et al., 2017).

3.4 A PRIOR FOR OPTIMIZATION PROBLEMS

To meta-train our method, we sample optimization problems from a prior distribution, $f \sim p(f)$, designed to generate diverse objective landscapes and support generalization across tasks. The prior combines quadratic, saddle, and monkey-saddle terms with tilting and confinement operations, as well as an RBF Gaussian process component approximated using *Random Fourier Features* (RFF) for computational efficiency (Rahimi & Recht, 2007). Further details and visualizations are provided in Appendix A, with theoretical background on universal-kernel Gaussian process priors in Appendix G.

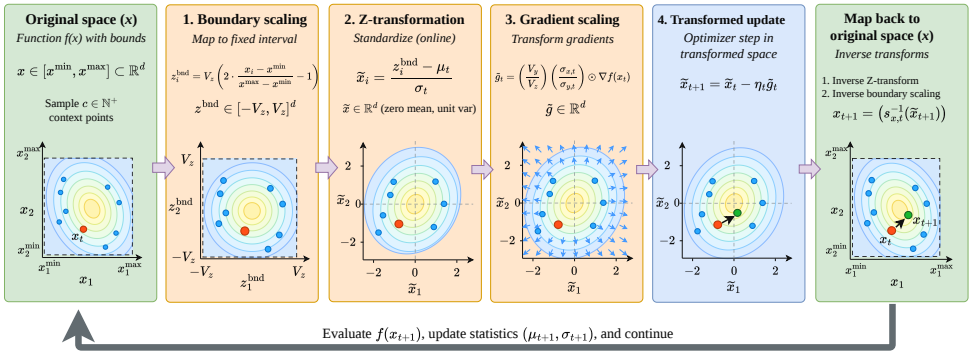


Figure 2: POP's workflow for scale-invariant policy inputs.

4 Scale-Invariant Policy Inputs

One of the major challenges when meta-learning an optimizer's policy is the mismatch of domains and ranges across functions, leading to out-of-distribution failures (Chen et al., 2021). Following an in-distribution assumption, we transform the domains and ranges of all training and test functions identically.

Initial Context. POP conditions its learning rate policy on the optimization trajectory and therefore requires an initial context before the first update. We sample $c \in \mathbb{N}^+$ observations from an initial distribution $p_0 \in \mathcal{P}(x)$, forming the initial trajectory $\tau^{(0)} := s_c$. The initial iterate x_0 is chosen as the context point with the lowest function value, while all c observations are retained in $\tau^{(0)}$. At each subsequent step, POP conditions on $\tau^{(t-1)}$ to select coordinate-wise step sizes, and appends the resulting observation to the trajectory according to Definition 3.1. The procedure is illustrated in the first step of Figure 2.

Boundary Scaling. For each function, we assume known domain coordinate-wise boundaries $x \in [x^{\min}, x^{\max}]$. This provides a common reference scale across objectives with different parameter ranges and is consistent with optimization settings in which a bounded or localized search region is specified a priori. We map each coordinate to a fixed symmetric interval via boundary scaling:

$$x_t^{\text{bnd}} = V_x \cdot \left(2 \cdot \frac{x_t - x^{\min}}{x^{\max} - x^{\min}} - 1 \right) \quad \text{with } V_x \in \mathbb{R}^+, \tag{8}$$

where V_x refers to the target scale. Analogously, we scale function values using running extrema $y_{t,\min} := \min_{j \le t} y_j$ and $y_{t,\max} := \max_{j \le t} y_j$, which summarize the observed function-value range up to optimization step t. The extrema are initialized from the random context s_c and updated online as additional points are evaluated:

$$y_t^{\text{bnd}} = V_y \cdot \left(2 \cdot \frac{y_t - y_t^{\min}}{y_t^{\max} - y_t^{\min}} - 1 \right) \quad \text{with } V_y \in \mathbb{R}^+, \tag{9}$$

where V_y refers to the target scale.

Z-transformation. Boundary-scaled x_t^{bnd} and y_t^{bnd} are then standardized by applying an online Z-transformation using the evaluation data of the current optimization trajectory, yielding $\tilde{x}_t$ and $\tilde{y}_t$.

Gradient Scaling. Because $\tilde{x}_t, \tilde{y}_t$ are transformed before being passed to the model, gradients defined in the original coordinates are no longer consistent with the transformed space. We therefore apply the corresponding Jacobian to obtain gradients that reflect the correct local sensitivities:

$$\nabla \tilde{f}(x_t) = \left(\frac{\sigma_{x,t}}{\sigma_{y,t}} \right) \left(\frac{V_y}{V_x} \right) \left(\frac{x^{\max} - x^{\min}}{y_t^{\max} - y_t^{\min}} \right) \odot \nabla f(x_t), \tag{10}$$

with $V_x, V_y \in \mathbb{R}^+$ denoting coordinate-wise scaling parameters, and $\sigma_{x,t}, \sigma_{y,t}$ denoting the standard deviation of the respective dimension up to time step t. The optimizer maintains one state per coordinate, defined exclusively in the transformed space.

Transformed Update. The policy conditions on the transformed optimization trajectory $\tilde{\tau}^{(t-1)}$, which contains the optimization history expressed in the normalized coordinate system. The corresponding gradient update is then performed in this transformed space $\tilde{x}_t = \tilde{x}_{t-1} - \eta_t \nabla \tilde{f}(x_{t-1})$. The updated iterate $\tilde{x}_t$ is then mapped back to the original domain by applying the inverse Z-transformation followed by inverse boundary scaling (we refer to both of these inverse transformations as $s_{x,t}^{-1}$), after which it is evaluated using the original objective function $f(x_t)$. Following the evaluation of the new x_t, y_t, and $\nabla f(x_t)$, the entire transformed set $\tilde{\tau}^{(t)}$ is re-transformed using the updated statistics.

5 Experimental Protocol

We meta-train POP on objective functions sampled from the synthetic prior described in Appendix A, using the parameter distributions reported in Table 1. Throughout meta-training, we restrict the prior to two-dimensional (D=2) objective functions. Representative samples from the prior are visualized in Appendix A.3. Rather than conditioning on the full optimization trajectory $\tau^{(t-1)}$, we decompose trajectories along dimensions and restrict the policy input to the history of the corresponding coordinate. The optimizer thus operates independently per dimension, enabling efficient batching and direct application to higher-dimensional problems. We open-source our implementation to foster future research[2].

Architecture and Training. We parameterize both the actor and critic using a shared transformer backbone with separate MLP heads, for a total of $333K$ trainable parameters. The policy processes coordinates independently and uses last-token pooling, allowing it to handle higher dimensions and longer trajectories without retraining. Full architectural and training hyperparameters are reported in Tables 2 and 3 in the appendix. The actor outputs the mean of a log-normal policy and samples actions using a learned, state-independent log standard deviation. For numerical stability, the log standard deviation is passed through a $\tanh$ nonlinearity and rescaled to lie in $[-3.0, 2.0]$. Algorithm 1 summarizes the complete POP meta-training procedure. At each training iteration, we sample $B = 256$ optimization problems from the function prior. For each sampled function, we collect 16 rollout trajectories. Each rollout begins with a newly generated random initial context of $c = 10$ points, after which the agent performs $T = 40$ optimization steps. After all rollouts are collected, the trajectories are combined and used to update the policy and value networks with PPO. We train POP for 8,000 iterations, yielding approximately 3.27×10^7 optimization problem instances. Training requires around 56 hours on a single NVIDIA H100 GPU.

Benchmarks and Downstream Tasks. We evaluate our method in the following settings:
• *In-distribution Performance.* We construct a validation set for ablations and a test set for zero-shot evaluation, each consisting of 1024 optimization problems in D=2. All problems use a fixed initial context of size c=10.

• *Out-of-distribution Performance.* We further evaluate the out-of-distribution performance of our method on 43 optimization problems from the widely used Virtual Library of Simulation Experiments (VLSE) benchmark (Surjanovic & Bingham). It covers a diverse range of optimization landscapes, including functions with many local minima, as well as bowl-shaped, plate-shaped, and valley-shaped geometries, in addition to steep ridges and drops. Each benchmark problem is evaluated at its native dimensionality, which ranges from 1D to 6D across the suite. We additionally evaluate POP on the 20 dimension-scalable functions at 32D.

• *Real-world Transfer.* We further evaluate whether POP transfers from synthetic optimization problems to objectives arising from real-world datasets. We consider Gaussian process (GP) hyperparameter optimization on the Diabetes, California Housing, Wine Quality, and Auto MPG regression datasets from the UCI Machine Learning Repository [3], where the kernel and noise hyperparameters are optimized by minimizing the negative log marginal likelihood. We additionally study transfer to neural-network training objectives across two tabular datasets. Full experimental details and additional results are provided in Appendix F.

Baselines. We compare our method against a diverse set of gradient-based optimization baselines. *Gradient Descent (GD)* (Cauchy, 1847), *Adam* (Kingma & Ba, 2015), and *Lion* (Chen et al., 2023)

[2] https://anonymous.4open.science/r/pop-F342/README.md
[3] https://archive.ics.uci.edu

represent first-order gradient-based optimizers, while *Sophia-G* (Liu et al., 2023) is a second-order-inspired optimizer that uses a lightweight diagonal Hessian approximation. *L-BFGS* (Liu & Nocedal, 1989) is a quasi-Newton method leveraging approximate second-order information. We add *Random Search (RS)* as a placebo method. All baselines are evaluated under the same optimization budget.

6 EVALUATION

Hypothesis 1: Learned optimizers trained on synthetic priors can generalize to unseen problems from the same distribution.
We test this hypothesis on unseen objective functions sampled independently from the same prior used during meta-training. We compare POP against GD, Adam, L-BFGS, Lion, Sophia-G, and random search. Since POP requires c initial context points, all methods are initialized using identical context sets, with the best-performing context point used as the starting point for the baselines. We tune the learning rate of GD, Adam, L-BFGS, Lion, and Sophia-G on the validation set and evaluate the selected configurations on the held-out test set; full hyperparameter settings are provided in Appendix C. We measure performance by normalized improvement (NI):

$$\mathrm{NI} = \frac{y_t^{\min} - y_t^*}{y_t^{\max} - y_t^{\min} + \epsilon},$$

where $y_t^{\min}$ and $y_t^{\max}$ are the minimum and maximum losses in the initial context, y_t^* is the best loss found so far, and ϵ ensures numerical stability.

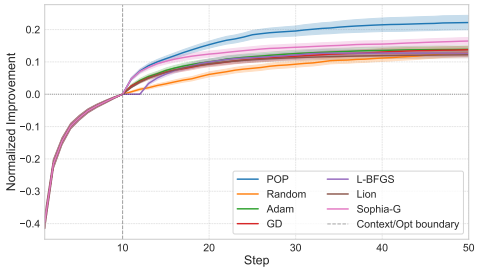


Figure 3: In-distribution performance. The dashed line ends the context phase.

As shown in Figure 3, POP achieves higher normalized improvement than all baselines throughout the optimization horizon, while the gradient-based baselines tend to plateau earlier and random search improves more slowly. We further compare methods by their ranks in Figure 4. Lower ranks indicate better performance, and connected methods are not significantly different (Friedman test with Nemenyi post-hoc test at $\alpha = 0.05$, computed with autorank (Herbold, 2020)). POP attains the best average rank. Beyond aggregate performance, qualitative trajectories in Appendix C.3 show that POP adapts its step sizes to alternate between exploration and local refinement, allowing it to escape suboptimal regions and revisit promising basins. Finally, we extend the comparison to non-convex solvers in Appendix E.1, where POP remains competitive despite relying only on first-order gradients.

Reward ablation. Our reward signal achieves the highest mean normalized improvement compared to four alternative reward formulations ablated in Appendix D.1, suggesting that allowing non-improving exploratory steps without penalty benefits optimization.

Ablating coordinate-wise independence. We compare POP with a matched Joint POP variant that models cross-coordinate interactions. Both perform similarly on the 2D test distribution, with final normalized improvements of 0.2217 (POP) and 0.2210 (Joint POP). All ablation results and a qualitative evaluation on a non-axis-aligned and ill-conditioned objective are provided in Appendix D.2.

Hypothesis 2: Learned coordinate optimizers trained for a fixed optimization horizon generalize to longer horizons.
To test POP's generalization capabilities beyond its training horizon, we evaluate performance under a longer optimization budget. Since our architecture is not restricted to the sequence length observed during training, we extend the evaluation budget from 50 to 100 iterations, setting c=10 and allowing the model to predict the subsequent T=90 updates, while retaining the hyperparameter settings tuned in the previous section. As shown in Figure 5, POP continues to improve beyond the training horizon and, under the 100-step budget, achieves the best overall performance, outperforming GD, Adam, L-BFGS, Lion, and Sophia-G. Additional results for the extended set of optimization methods, for the 100-iteration setting, are provided in Appendix E.2.

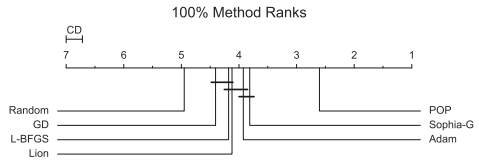


Figure 4: Full-budget ranks. Lower is better; bars connect methods without significant differences.

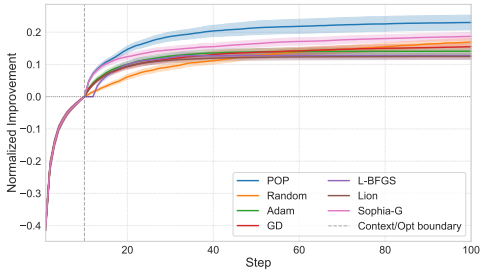


Figure 5: Performance at T=90 (mean and 95% CI). The dashed line ends the context phase.

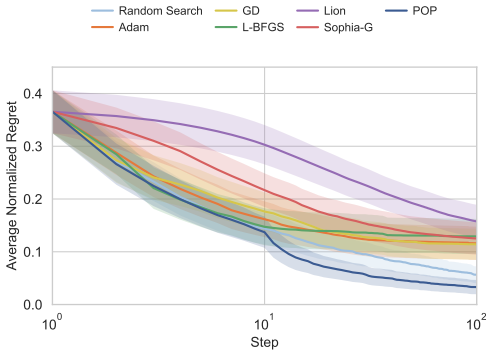


Figure 6: Average normalized regret on VLSE.

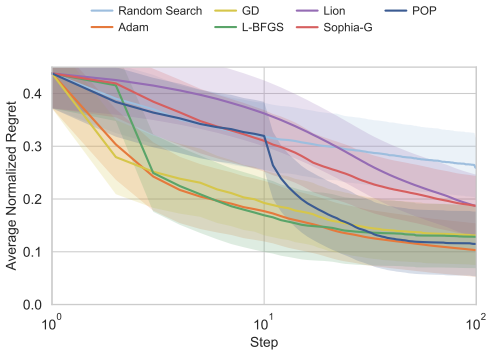
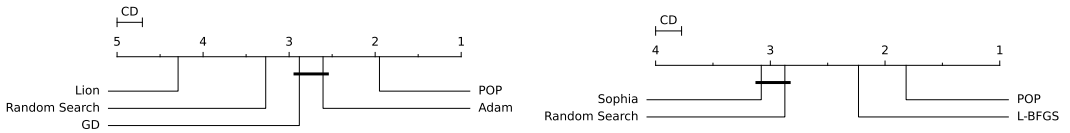


Figure 7: Average normalized regret on 20 VLSE functions at 32D.

Hypothesis 3: Learned coordinate optimizers trained on a low-dimensional (2D) synthetic prior can generalize to out-of-distribution problems and higher dimensions.
Since POP operates coordinate-wise, it can be applied to higher dimensions by batching over coordinates, without architectural changes or retraining. We further validate our method on the *Virtual Library of Simulation Experiments* (Surjanovic & Bingham) benchmark. We tune GD, Adam, Lion, Sophia-G, and L-BFGS separately for each VLSE function by sweeping seven learning rates over five random initializations for tuning, and evaluate the selected configuration for $T = 90$ steps on ten independent random initializations. Importantly, these auxiliary evaluations are performed directly on the test functions, while POP is applied entirely zero-shot. The comparison is therefore intentionally favorable to the baselines. We compare all methods in terms of per-task normalized regret $r_{\mathrm{opt},t} = \frac{y_t^* - y^{\min}}{y^{\max} - y^{\min}}$, where y_t^* is the best function value found up to step t, and $y^{\min}$ and $y^{\max}$ denote the minimum and maximum of the corresponding function. We average the regret across the ten seeds and all VLSE functions.

We assess method rankings and statistical significance using the same autorank procedure as above (Herbold, 2020). In Figure 6, POP achieves the lowest average normalized regret across the VLSE benchmark despite the function-specific tuning of the baselines. The CD diagrams in Figure 8 further show that POP achieves the lowest mean rank and significantly outperforms the respective tuned first- and second-order baselines. Comparisons to further model families are provided in Appendix E.3. Although POP incurs additional inference overhead from its Transformer policy, a wall-clock comparison shows that its per-decision runtime remains substantially below the best performing non-convex solvers. Detailed runtime measurements are reported in Appendix E.4.

Higher-dimensional VLSE. We also evaluate POP zero-shot on the 20 dimension-scalable VLSE functions at $D = 32$, using the same per-function learning rate sweeps and evaluation setup as before on the new functions. This subset contains a higher proportion of convex functions (7/20 vs. 9/43 full suite). As shown in Figure 7, POP achieves the second-lowest average normalized regret at $D = 32$, behind only tuned Adam over seven learning rates, despite being trained exclusively on $D = 2$ functions and applied zero-shot without task-specific tuning. We report the corresponding ranks in Appendix E.3.



(a) POP vs. First-order Methods

(b) POP vs. Second-order Methods

Figure 8: Critical difference diagrams on VLSE for first- and second-order baselines.

Ablating the Prior. We ablate the synthetic prior by masking each prior component individually, without renormalizing the remaining weights, and retraining each variant for 8K iterations. Evaluated on VLSE, the full prior achieves the best mean rank, while removing the RFF (GP) components causes the largest performance degradation, suggesting that these components contribute most strongly to out-of-distribution generalization. Full results are provided in Appendix D.3.

Hypothesis 4: Learned coordinate optimizers trained on a low-dimensional synthetic prior can transfer to real-world optimization tasks.
We evaluate POP on the real-world transfer tasks described in Section 5. For the Gaussian process (GP) hyperparameter optimization experiments, we fit a zero-mean GP with an automatic relevance determination (ARD) squared-exponential kernel and optimize its $d + 2$ kernel and noise hyperparameters by minimizing the negative log marginal likelihood (NLL). All methods are initialized from the same best of 10 random restarts and optimized for 200 steps. We tune Adam and SGD over learning rates $\{10^{-1}, 10^{-2}, 10^{-3}\}$, while POP is applied zero-shot without task-specific tuning. We report the MSE of the posterior predictive mean on held-out test data. Across the four datasets, POP remains competitive with the best-tuned baseline, obtaining the lowest test MSE on California Housing (0.332) and Auto MPG (0.096), and ranking second on Diabetes (0.564). On Wine Quality, POP performs weaker than the best Adam configuration (0.720 vs. 0.679). Interestingly, lower training NLL does not consistently translate into lower test error; e.g., on Auto MPG, Adam obtains a lower NLL while POP achieves the best test MSE. Overall, these results provide evidence that optimization behavior learned exclusively from our synthetic prior can transfer zero-shot to objectives arising from real-world data. Full results and an additional transfer study on neural-network training objectives are shown in Appendix F.

Limitations and Future Work. POP is designed as a proof-of-concept for prior-fitted first-order optimization and therefore adopts several simplifying assumptions. Its trajectory-conditioned policy stores per-coordinate histories, leading to linear memory growth and quadratic self-attention cost with the optimization horizon. The synthetic prior is designed primarily for deterministic, low-dimensional continuous functions, and robustness to noisy or stochastic objectives and large-scale neural-network optimization remains underexplored. The scaling procedure also assumes known coordinate-wise domain boundaries. Finally, coordinate-wise processing limits cross-coordinate interactions, though our joint-conditioning and rotated-objective experiments suggest little impact in the evaluated settings. Future work will explore compressed trajectory representations, priors better suited to stochastic ML loss landscapes, and scalable architectures.

7 Conclusion

Optimization algorithms are central to minimizing functions, yet conventional first-order methods rely on hand-designed mechanisms for adapting their updates across objectives. We propose POP, a meta-learned first-order optimizer that restricts the learned action to trajectory-conditioned coordinate-wise step sizes along the descent direction. POP is meta-trained on millions of synthetic optimization problems drawn from a task-independent generative prior. In a thorough evaluation, our model generalizes to longer horizons and higher-dimensional problems, achieves strong performance on a well-established benchmark suite encompassing 43 diverse optimization functions, and transfers zero-shot to practical optimization tasks. Overall, POP shows that a constrained first-order policy, meta-trained solely on a synthetic generative prior, can acquire transferable optimization behavior across substantially different optimization settings.

AI USE STATEMENT

In this work, generative AI tools were used to assist with code implementation, support the authors' understanding of relevant concepts, improve the clarity and readability of the manuscript, and refine the visual presentation of plots. Generative AI tools were not used to formulate the original research question. The authors identified the majority of the relevant literature independently, using AI tools only to supplement this process by suggesting additional potentially relevant work.

All AI-assisted outputs were reviewed and verified by the authors. The authors checked AI-assisted code for correctness, verified literature suggestions against the original sources, and reviewed all AI-assisted textual and visual revisions prior to inclusion. The authors take full responsibility for the final content, claims, results, and artifacts presented in this work.

REPRODUCIBILITY STATEMENT

We open source our implementation for meta-training and evaluation at `https://anonymous.4open.science/r/pop-F342/README.md`. POP is meta-trained entirely on synthetic optimization problems from the prior specified in Section 3.4 and Appendix A, while the real-world transfer experiments use public datasets including Diabetes, California Housing, Wine Quality, and Auto MPG. Section 3 fully specify the method, including the policy, reward, meta-learning objective, prior, and scale-invariant transformations. Section 5 describes the experimental protocol and compute budget, and the Appendicies provide the prior Appendix A, architecture and training hyperparameters Appendix B, baseline tuning Appendix C, ablations Appendix D, extended comparisons Appendix E, and real-world transfer details Appendix F.

REFERENCES

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.

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

Yue Cao, Tianlong Chen, Zhangyang Wang, and Yang Shen. Learning to optimize in swarms. *Advances in neural information processing systems*, 32:15044–15054, 2019.

Augustin-Louis Cauchy. Méthode générale pour la résolution des systèmes d'équations simultanées. In *Comptes Rendus Hebdomadaires des Séances de l'Académie des Sciences*, 1847.

Tianlong Chen, Xiaohan Chen, Wuyang Chen, Howard Heaton, Jialin Liu, Zhangyang Wang, and W. Yin. Learning to optimize: A primer and a benchmark. *J. Mach. Learn. Res.*, 23:189:1–189:59, 2021.

Xiangning Chen, Chen Liang, Da Huang, Esteban Real, Kaiyuan Wang, Hieu Pham, Xuanyi Dong, Thang Luong, Cho-Jui Hsieh, Yifeng Lu, et al. Symbolic discovery of optimization algorithms. *Advances in neural information processing systems*, 36:49205–49233, 2023.

Alexander I Cowen-Rivers, Wenlong Lyu, Rasul Tutunov, Zhi Wang, Antoine Grosnit, Ryan Rhys Griffiths, Alexandre Max Maraval, Hao Jianye, Jun Wang, Jan Peters, et al. Hebo: Pushing the limits of sample-efficient hyper-parameter optimisation. *Journal of Artificial Intelligence Research*, 74:1269–1349, 2022.

Mohammad Sh. Daoud, Mohammad Shehab, Hani M. Al-Mimi, Laith Abualigah, Raed Abu Zitar, and Mohd Khaled Yousef Shambour. Gradient-based optimizer (gbo): A review, theory, variants, and applications. *Archives of Computational Methods in Engineering*, 30(4):2431–2449, December 2022. ISSN 1886-1784. doi: 10.1007/s11831-022-09872-y.

Aaron Defazio, Xingyu Yang, Ahmed Khaled, Konstantin Mishchenko, Harsh Mehta, and Ashok Cutkosky. The road less scheduled. *Advances in Neural Information Processing Systems*, 37: 9974–10007, 2024.

Alexander D Goldie, Chris Lu, Matthew T Jackson, Shimon Whiteson, and Jakob N Foerster. Can learned optimization make reinforcement learning less difficult? *Advances in Neural Information Processing Systems*, 37:5454–5497, 2024.

Erik Gärtner, Luke Metz, Mykhaylo Andriluka, C. Daniel Freeman, and Cristian Sminchisescu. Transformer-based learned optimization, 2023.

Nikolaus Hansen. The cma evolution strategy: A tutorial. *arXiv preprint arXiv:1604.00772*, 2016.

Esraa Hassan, Mahmoud Y. Shams, Noha A. Hikal, and Samir Elmougy. The effect of choosing optimizer algorithms to improve computer vision tasks: a comparative study. *Multimedia Tools and Applications*, 82(11):16591–16633, September 2022. ISSN 1573-7721. doi: 10.1007/s11042-022-13820-0.

Steffen Herbold. Autorank: A python package for automated ranking of classifiers. *Journal of Open Source Software*, 5(48):2173, 2020. doi: 10.21105/joss.02173.

John H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, 1992.

Noah Hollmann, Samuel G. Müller, Lennart Purucker, Arjun Krishnakumar, Max Körfer, Shi Bin Hoo, R. Schirrmeister, and Frank Hutter. Accurate predictions on small data with a tabular foundation model. *Nature*, 637:319 – 326, 2025. doi: 10.1038/s41586-024-08328-6.

N. Keskar and R. Socher. Improving generalization performance by switching from adam to sgd. *ArXiv*, abs/1712.07628, 2017.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun (eds.), *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

Gus Kristiansen, Mark Sandler, A. Zhmoginov, N. Miller, Anirudh Goyal, Jihwan Lee, and Max Vladymyrov. Narrowing the focus: Learned optimizers for pretrained models. *ArXiv*, abs/2408.09310, 2024. doi: 10.48550/arxiv.2408.09310.

Qingfeng Lan, Rupam Mahmood, Shuicheng Yan, and Zhongwen Xu. Learning to optimize for reinforcement learning. *RLJ*, 2:481–497, 2023. doi: 10.48550/arxiv.2302.01470.

Ke Li and Jitendra Malik. Learning to optimize. In *International Conference on Learning Representations*, 2017.

Lizhi Liao, Heng Li, Weiyi Shang, and L. Ma. An empirical study of the impact of hyperparameter tuning and model optimization on the performance properties of deep neural networks. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 31:1 – 40, 2022. doi: 10.1145/3506695.

Dong C Liu and Jorge Nocedal. On the limited memory bfgs method for large scale optimization. *Mathematical Programming*, 45(1):503–528, 1989.

Hong Liu, Zhiyuan Li, David Leo Wright Hall, Percy Liang, and Tengyu Ma. Sophia: A scalable stochastic second-order optimizer for language model pre-training. In *The Twelfth International Conference on Learning Representations*, 2023.

Yanli Liu, Yuan Gao, and Wotao Yin. An improved analysis of stochastic gradient descent with momentum. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS '20, Red Hook, NY, USA, 2020. Curran Associates Inc. ISBN 9781713829546.

Amala Mary, Vincent, I. P. J. Student Member, and I. A. A. B. Senior Member. Optimizing hyperparameters in meta-learning for enhanced image classification. *IEEE Access*, 13:130816–130831, 2025. doi: 10.1109/access.2025.3591142.

Luke Metz, Niru Maheswaranathan, C. Freeman, Ben Poole, and Jascha Narain Sohl-Dickstein. Tasks, stability, architecture, and compute: Training more effective learned optimizers, and using them to train themselves. *ArXiv*, abs/2009.11243, 2020.

Luke Metz, James Harrison, C. Freeman, Amil Merchant, Lucas Beyer, James Bradbury, Naman Agrawal, Ben Poole, Igor Mordatch, Adam Roberts, and Jascha Narain Sohl-Dickstein. Velo: Training versatile learned optimizers by scaling up. *ArXiv*, abs/2211.09760, 2022. doi: 10.48550/arxiv.2211.09760.

A. Moudgil, Boris Knyazev, Guillaume Lajoie, and Eugene Belilovsky. Celo: Training versatile learned optimizers on a compute diet. *Trans. Mach. Learn. Res.*, 2025, 2025.

Samuel Müller, Noah Hollmann, Sebastian Pineda Arango, Josif Grabocka, and Frank Hutter. Transformers can do bayesian inference. In *International Conference on Learning Representations*, 2022.

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

Ali Rahimi and Benjamin Recht. Random features for large-scale kernel machines. In *Neural Information Processing Systems*, 2007.

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

H. Robbins and S. Monro. A stochastic approximation method. *Annals of Mathematical Statistics*, 22:400–407, 1951.

Joel Schlotthauer, Christian Kroos, Chris Hinze, Viktor Hangya, Luzian Hahn, and Fabian Küch. Pre-training llms on a budget: A comparison of three optimizers. *ArXiv*, abs/2507.08472, 2025. doi: 10.48550/arxiv.2507.08472.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017.

Hans-Paul Schwefel. Evolution and optimum seeking. In *Sixth-generation computer technology series*, 1995.

Jun Shu, Yanwen Zhu, Qian Zhao, Deyu Meng, and Zongben Xu. Meta-lr-schedule-net: Learned lr schedules that scale and generalize, 07 2020.

Ingo Steinwart and Andreas Christmann. *Support Vector Machines*. Information Science and Statistics. Springer, 2008. ISBN 9780387772424.

Rainer Storn and Kenneth Price. Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11(4):341–359, 1997.

S. Surjanovic and D. Bingham. Virtual library of simulation experiments: Test functions and datasets. Retrieved December 14, 2025, from `http://www.sfu.ca/~ssurjano`.

Ke Tang and Xin Yao. Learn to optimize—a brief overview. *National Science Review*, 11(8), April 2024. ISSN 2053-714X. doi: 10.1093/nsr/nwae132.

Yingjie Tian, Yuqi Zhang, and Haibin Zhang. Recent advances in stochastic gradient descent in deep learning. *Mathematics*, 2023. doi: 10.3390/math11030682.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

T. Vishnu, Pankaj Malhotra, Jyoti Narwariya, L. Vig, and Gautam M. Shroff. Meta-learning for black-box optimization. *Machine Learning and Knowledge Discovery in Databases*, 2020.

Holger Wendland. *Scattered Data Approximation*. Cambridge University Press, 2004.

Olga Wichrowska, Niru Maheswaranathan, Matthew W Hoffman, Sergio Gomez Colmenarejo, Misha Denil, Nando Freitas, and Jascha Sohl-Dickstein. Learned optimizers that scale and generalize. In *International conference on machine learning*, pp. 3751–3760. PMLR, 2017.

Zhen Xu, Andrew M Dai, Jonas Kemp, and Luke Metz. Learning an adaptive learning rate schedule. *arXiv preprint arXiv:1909.09712*, 2019.

A A SYNTHETIC PRIOR FOR OPTIMIZATION PROBLEMS

This appendix describes the synthetic prior over optimization landscapes used to meta-train POP. We define its functional form and constituent components in Appendix A.1, summarize the corresponding parameter distributions in Appendix A.2, and provide representative samples from the prior in Appendix A.3.

A.1 CONSTRUCTION OF THE SYNTHETIC PRIOR

We build each function from seven additive components. For a given number of dimensions D, the function takes the form:

$$f(x) = \alpha_1 f^{(1)}_{\mathrm{RFF}}(x) + \alpha_2 f^{(2)}_{\mathrm{RFF}}(x) + \alpha_3 f_Q(x) + \alpha_4 f_S(x) + \alpha_5 f_M(x) + f_{\mathrm{conf}}(x) + f_{\mathrm{tilt}}(x),$$

The weights $(\alpha_1, \ldots, \alpha_5)$ are initialized by drawing five independent positive uniform random variables and normalizing them to sum to one. We then adjust these weights according to the sampled landscape type: bowl-dominated instances suppress the saddle and monkey-saddle terms, saddle instances receive a non-negligible saddle weight, and instances without a monkey saddle set the corresponding weight to zero. The remaining active weights are re-normalized where necessary. The confinement, tilt, and vertical offset terms are added separately. In the following, we define each component in detail:

Quadratic component. This term adds a convex bowl-shaped surface:

$$f_Q(x) = \sum_{d=1}^{D} \beta^1_d \big(R_Q \tilde{x}\big)^2_d,$$

where $\tilde{x}$ denotes the shifted coordinates normalized by the domain half-width, β^1_d sets the curvature along each dimension, and R_Q is an optional rotation matrix. β^1_d sets the curvature along each dimension and may be sampled with a target condition number to produce ill-conditioned objectives.

Saddle component. This term acts on independently rotated coordinates $(u, v) = R_S \tilde{x}$:

$$f_S(x) = \tfrac{1}{2}\, \gamma^1 \big(u^2 - \gamma^2 v^2\big),$$

where γ^1 sets the saddle strength and γ^2 its asymmetry.

Monkey saddle component. A higher-order saddle point is defined as

$$f_M(x) = \tfrac{1}{3}\, \delta^1 \big(u^3 - 3uv^2\big),$$

where $(u, v) = R_S \tilde{x}$ and δ^1 sets its strength.

Confinement and tilt. To keep the function bounded, we add a quartic term

$$f_{\mathrm{conf}}(x) = \tfrac{1}{4}\, c \|\tilde{x}\|^4.$$

A linear tilt

$$f_{\mathrm{tilt}}(x) = t^\top \tilde{x}$$

is also added to break symmetry.

Random Fourier Feature (RFF) components. Each RFF component (Rahimi & Recht, 2007) approximates a Gaussian-process sample with an RBF kernel:

$$f_{\mathrm{RFF}}(x) = \sqrt{2} \sum_{m=1}^{M} \omega^1_m \cos\big((\omega^2_m)^\top x + \omega^3_m\big),$$

with phases $\omega^3_m \sim \mathcal{U}(0, 2\pi)$, weights $\omega^1_m \sim \mathcal{N}(0, \sigma^2/M)$, and frequencies $\omega^2_m \sim \mathcal{N}(\mathbf{0}, \ell^{-2}\mathbf{I}_D)$, where M is the number of *Random Fourier Features*. The lengthscale $\ell \sim \mathcal{U}(\ell_{\min}, \ell_{\max})$ controls

smoothness, while $\sigma \sim \mathcal{U}(\sigma_{\min}, \sigma_{\max})$ controls amplitude; the $1/M$ scaling keeps the variance bounded as M increases. We sample two independent RFF components of this form. To introduce multi-scale variation, each frequency is multiplied by a bounded bimodal log-normal jitter factor.

All deterministic geometric components, namely the quadratic, saddle, monkey-saddle, confinement, and tilt terms, are evaluated on shifted coordinates normalized by the domain half-width, while the RFF components are evaluated on the original input coordinates.

To sample a new function, we sample the mixing weights $(\alpha_1, \ldots, \alpha_5)$, function type, geometric parameters $(\beta, \gamma, \delta, R, R_S)$, and RFF parameters $(\ell_j, \sigma_j, \omega^j)$ for each GP components. The resulting objectives vary naturally in topology, from smooth bowls to saddle points and multi-modal landscapes, reflecting the diversity induced by the prior rather than any manual shaping of function landscapes. Samples of our prior are visually presented in Figure 9.

A.2 Prior Hyperparameters and Sampling Distributions

Table 1 summarizes the main hyperparameters and sampling distributions used to instantiate the synthetic prior. The five mixing weights $(\alpha_1, \ldots, \alpha_5)$ are initialized as normalized positive uniform draws and subsequently adjusted according to the sampled landscape type. In particular, saddle-dominated instances are assigned a non-negligible saddle weight, inactive monkey-saddle components are assigned zero weight, and the remaining active weights are renormalized accordingly. The table reports the principal hyperparameters of the prior; additional implementation details, including the exact landscape-type adjustment rules and frequency-jitter parameters, are provided in the published code.

Table 1: Main hyperparameter values of the proposed prior.

Symbol / Quantity	Distribution / Value
D	2
Domain	$[-50, 50]^D$
Input shift	$\mathcal{U}(-40, 40)^D$
Vertical offset b	0
M	1000
Quadratic mixing weight α_3	$\mathcal{U}(0.05, 0.4)$ for bowl-dominated instances
Saddle strength γ^1	log-$\mathcal{U}(1, 12)$
Saddle asymmetry γ^2	$\mathcal{U}(0.3, 2.5)$
Monkey-saddle strength δ^1	log-$\mathcal{U}(0.5, 6)$
Confinement coefficient c	log-$\mathcal{U}(0.5, 3)$
Tilt strength $\|t\|$	$\mathcal{U}(0, 0.5)$
RFF kernel	RBF
RFF lengthscale ℓ	$\mathcal{U}(4, 10)$
RFF amplitude σ	$\mathcal{U}(1, 5)$
Quadratic curvature scale β_d^1	$\mathcal{U}(0.5, 8.0)$
Rotation probability	0.7
Ill-conditioning probability	0.5 for bowl-dominated instances, 0.4 otherwise
Condition number κ	$\mathcal{U}(10^3, 10^6)$ for bowl-dominated instances, $\mathcal{U}(10^1, 10^4)$ otherwise

A.3 Visual Illustration of the Prior

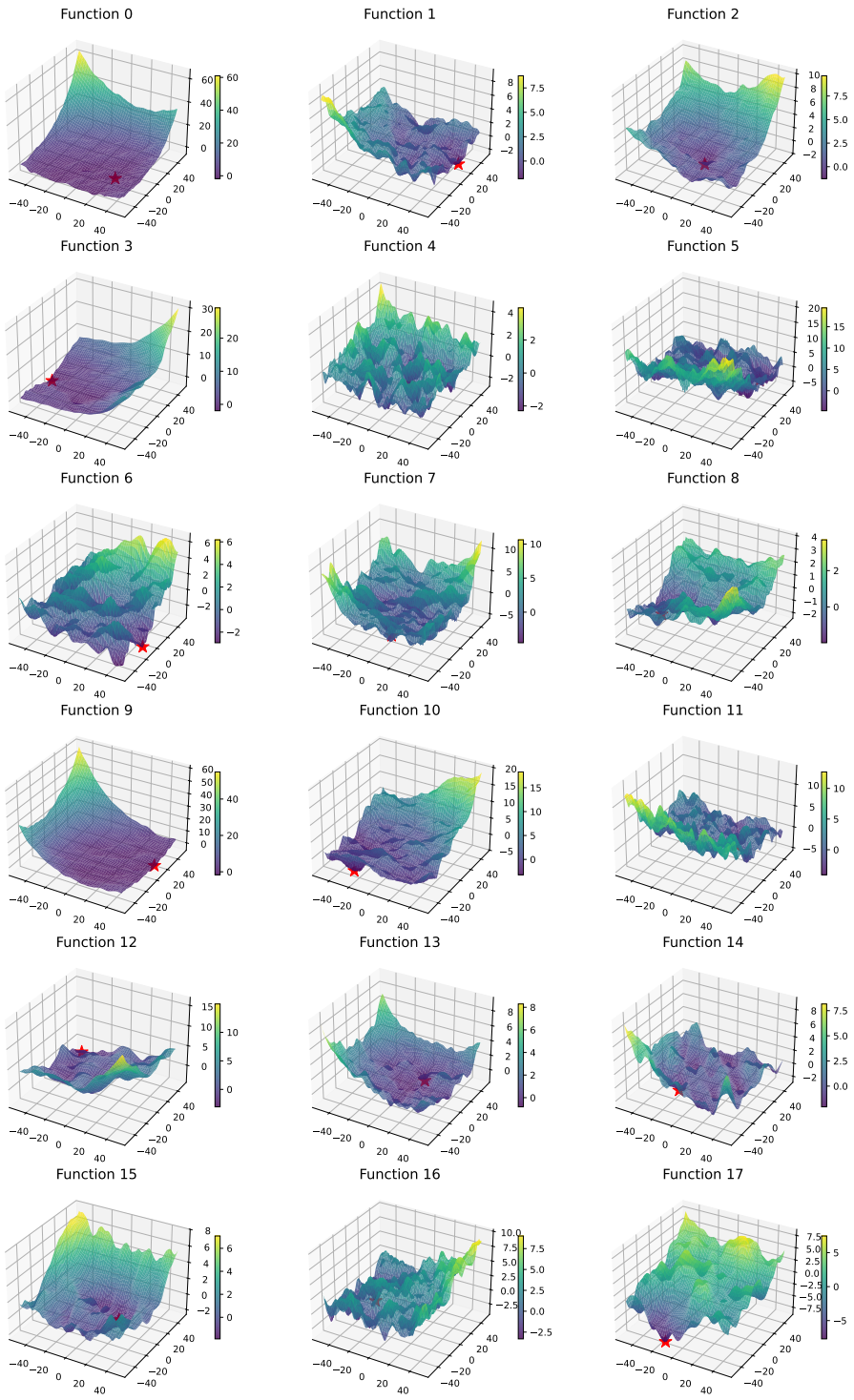


Figure 9: Exemplary functions sampled from the prior distribution where the red star (⋆) highlights the global minimum. (cf. Section 3.4)

B Architecture and Training Details

This appendix provides an algorithmic overview of POP together with the architectural and training details used in all experiments. Algorithm 1 summarizes the complete meta-training procedure, while Table 2 and Table 3 report the Transformer architecture and PPO training hyperparameters, respectively. We additionally report the meta-training reward dynamics in Figure 10, illustrating how the policy evolves and stabilizes over the course of training. Unless stated otherwise, a single configuration is used across all experiments. We set the boundary target scale to $V_x = V_y = 3$.

Algorithmic overview. At each training update, objective functions are sampled from the synthetic prior and multiple optimization trajectories are collected for each function. Starting from an initial random context, POP conditions on the transformed optimization history to predict coordinate-wise learning rates and applies the corresponding gradient updates in transformed space. The resulting observations are appended to the trajectory, after which GAE advantages and returns are computed and the policy and value function are updated using PPO.

Algorithm 1 POP training

Require: Prior $p(f)$, horizon T, context size c, rollouts per function K
Initialize policy π_θ and value function V_ϕ
for update $= 1, \dots, N$ **do**
  Sample $f_1, \dots, f_B \sim p(f)$
  for $f \in \{f_1, \dots, f_B\}$ **do**
    for $k = 1, \dots, K$ **do**
      Sample c context observations to initialize $\tau^{(0)}$
      Set x_0 to the best context point
      for $t = 1, \dots, T$ **do**
        Transform $\tau^{(t-1)}$ using boundary, Z-, and gradient scaling to obtain $\widetilde{\tau}^{(t-1)}$
        Sample $\eta_t \sim \pi_\theta(\cdot \mid \widetilde{\tau}^{(t-1)})$
        Update in transformed space: $\widetilde{x}_t \leftarrow \widetilde{x}_{t-1} - \eta_t \odot \nabla \widetilde{f}(x_{t-1})$
        Map $\widetilde{x}_t$ back and evaluate $y_t = f(x_t)$ and $\nabla f(x_t)$
        Append $(x_t, y_t, \nabla f(x_t), t/T)$ to the trajectory
        Set $R_t \leftarrow \max(0, y^*_{t-1} - y_t)$
      end for
    end for
  end for
  Compute GAE advantages $\widehat{A}$ and returns $\widehat{R}$
  Update θ and ϕ with clipped PPO for E epochs
end for

Table 2: Key architecture hyperparameters for the shared Transformer backbone and actor/critic heads.

Component	Value
Shared backbone (Transformer)	
Embedding dimension	64
# Transformer blocks	8
# Attention heads	8 ($d_{\text{head}} = 8$)
Dropout	0.05
Pooling	Last-token pooling
Shared output dim	32
Actor / Critic heads	
Actor head (MLP)	$32 \to 256 \to 256 \to 1$
Critic head (MLP)	$32 \to 256 \to 256 \to 1$

Table 3: PPO training hyperparameters and optimizer settings.

Component	Value
Actor lr	5×10^{-5}
Critic lr	5×10^{-5}
Optimizer	`AdamWScheduleFree (Defazio et al., 2024)`
Actor weight decay	1×10^{-4}
Critic weight decay	1×10^{-4}
Actor warmup	300
Critic warmup	10
PPO clip ratio	0.1
GAE γ	0.97
GAE λ	0.9
Epochs over collected trajectories	4
PPO Minibatch size	8192
Gradient clip norm	0.5
KL early stopping	0.01
Adaptive KL penalty	yes (initial coef 0.1, target KL 0.01)
Return normalization	Online Welford estimator

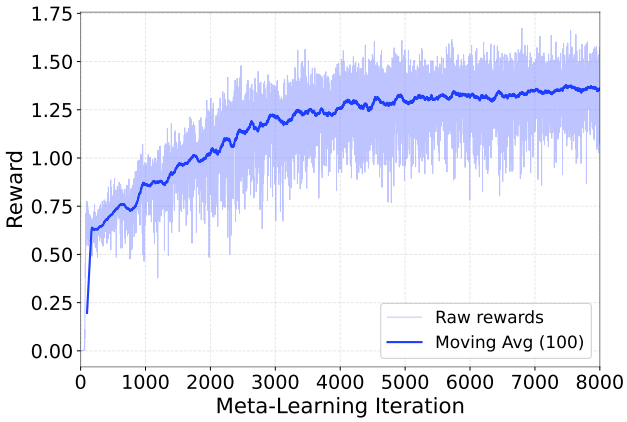


Figure 10: Episode rewards with moving average.

C Baseline Hyperparameter Optimization and Qualitative Analysis

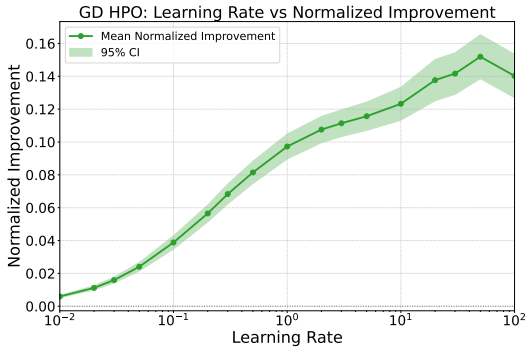
This appendix provides additional experimental details supporting the evaluation in Section 6. We first describe the hyperparameter optimization procedures used for the gradient-based, quasi-Newton, and evolutionary baselines, and report the selected configurations. We then provide qualitative examples of POP's optimization trajectories on objectives sampled from the synthetic prior.

C.1 Gradient-Based and Quasi-Newton Baseline Tuning

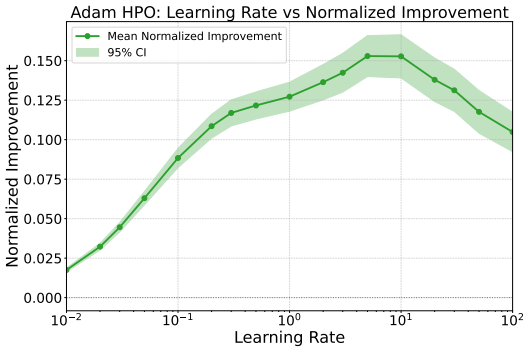
We sweep the initial learning rate for GD, Adam, Lion, Sophia-G and L-BFGS using a grid search over

$$\eta \in \{0.01, 0.02, 0.03, 0.05, 0.1, 0.2, 0.3, 0.5, 1.0, 2.0, 3.0, 5.0, 10.0, 20.0, 30.0, 50.0, 100.0\}.$$

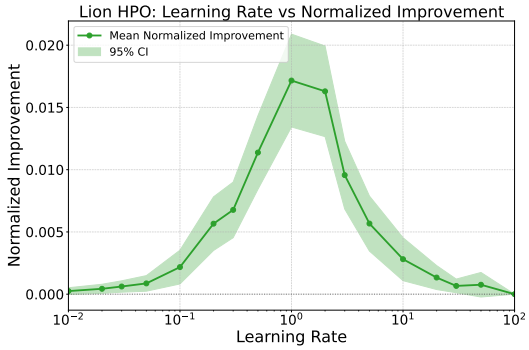
All other hyperparameters are kept at their default values. Figures 11a to 11e show the learning rate sweeps for GD, Adam, Lion, Sophia-G, and L-BFGS on the validation set, respectively.



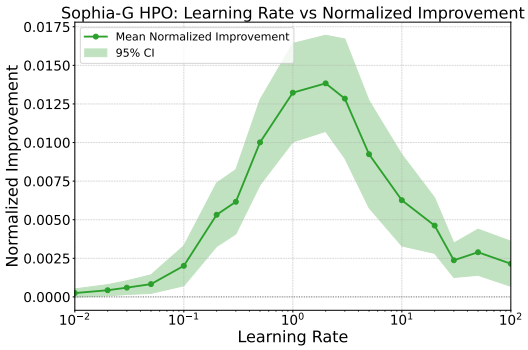
(a) GD peaks at a learning rate of 50.



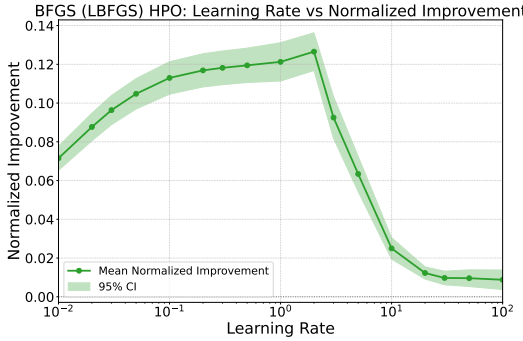
(b) Adam peaks at a learning rate of 10.



(c) Lion peaks at a learning rate of 1.



(d) Sophia peaks at a learning rate of 2.



(e) L-BFGS peaks at a learning rate of 2.

Figure 11: Learning rate sweep on the in-distribution validation set.

C.2 Evolutionary Baseline Tuning

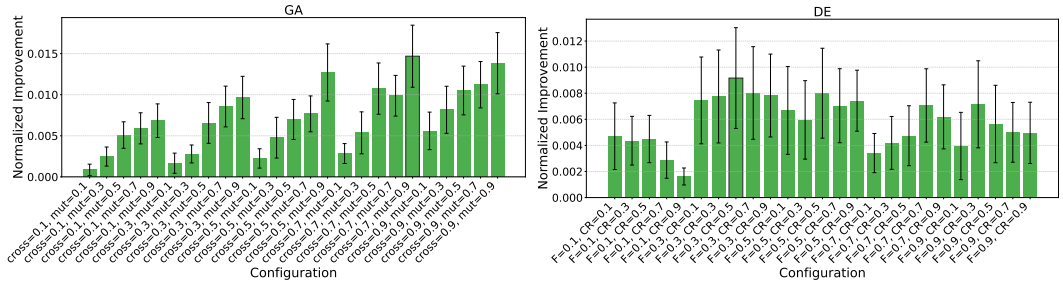
For GA and DE, we sweep the crossover rate and mutation factor using a grid search over

$$crossover_rate \in \{0.1, 0.3, 0.5, 0.7, 0.9\} \quad mutation_factor \in \{0.1, 0.3, 0.5, 0.7, 0.9\}.$$

For CMAES, we sweep the initial step size using a grid search over

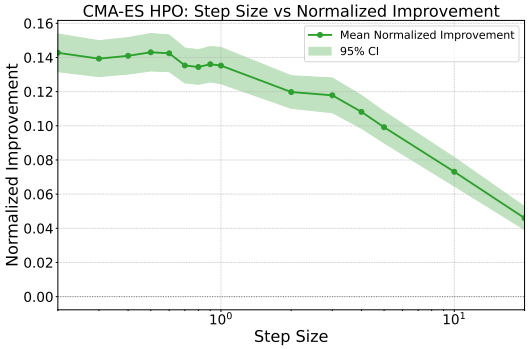
$$init_step_size \in \{0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0, 2.0, 3.0, 4.0, 5.0, 10.0, 20.0\}.$$

We keep the population size at 10 for all evolutionary methods. Figures 12a to 12c show the sweeps for GA , DE and CMA-ES on the validation set, respectively.



(a) GA peaks at a crossover rate of 0.7 and mutation factor of 0.9.

(b) DE peaks at a crossover rate of 0.5 and mutation factor of 0.3.



(c) CMAES peaks at an initial step size of 0.2.

Figure 12: Hyperparameter sweep for the evolutionary methods on the in-distribution validation set.

C.3 Qualitative Optimization Trajectories

We provide qualitative visualizations of optimization trajectories produced by POP on functions sampled from the synthetic prior. These examples complement the quantitative results in Section 6 by illustrating how POP adapts its step sizes and exploration behavior to the local geometry of the objective landscape.

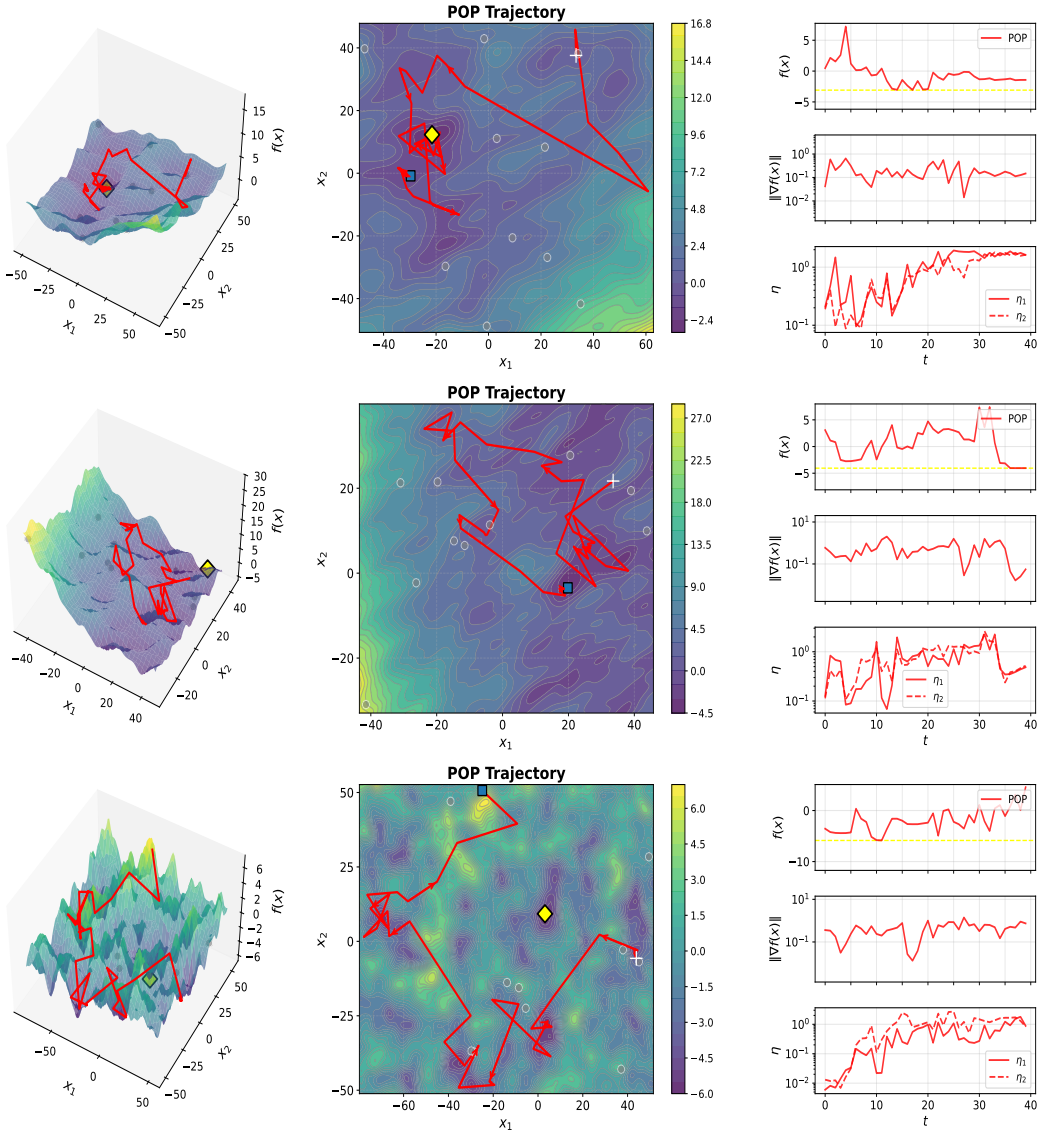


Figure 13: POP trajectories on in-distribution functions. The algorithm progresses from the initial state (white cross) terminating at the final state (square).

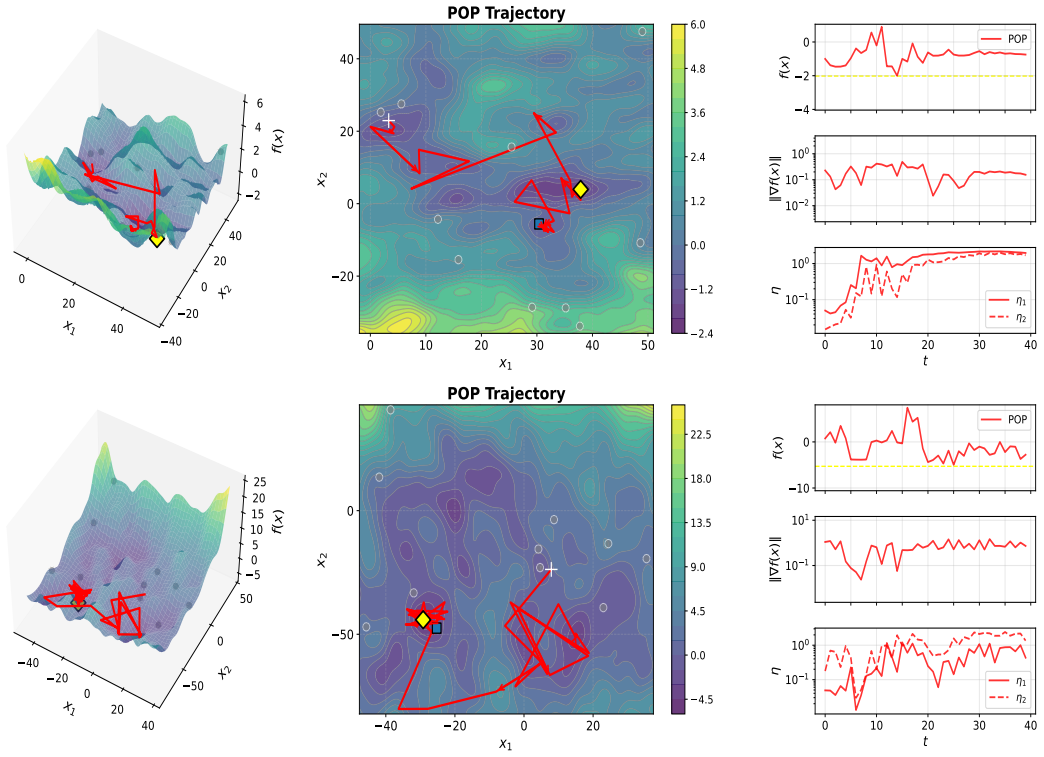


Figure 14: POP trajectories on in-distribution functions. The algorithm progresses from the initial state (white cross) terminating at the final state (square).

D Ablation Studies

This appendix presents additional ablation studies examining key design choices of POP. We first compare alternative reward formulations used for meta-training Appendix D.1, then evaluate the effect of processing optimization coordinates independently Appendix D.2, and finally assess the contribution of individual components of the synthetic prior Appendix D.3.

D.1 Reward Formulation Ablation

We compare five reward families using a lightweight 166K-parameter model. Let $\tilde{y}_t$ denote the boundary-scaled and Z-transformed objective value at step t, and let $\tilde{y}^*_{t-1}$ denote the best transformed value observed previously. The evaluated rewards are:

$$
\begin{aligned}
R_{\text{Current}} &= \tilde{y}_t, \\
R_{\text{Global Imp}} &= \tilde{y}^*_{t-1} - \tilde{y}_t, \\
R_{\text{Global Imp},c} &= \max\big(0, \tilde{y}^*_{t-1} - \tilde{y}_t\big), \\
R_{\text{SMAPE}} &= \max\left(0, \frac{2(\tilde{y}^*_{t-1} - \tilde{y}_t)}{|\tilde{y}^*_{t-1}| + |\tilde{y}_t|}\right), \\
R_{\text{Mix},\alpha} &= \alpha \max\big(0, \tilde{y}^*_{t-1} - \tilde{y}_t\big) + (1-\alpha)(\tilde{y}_{t-1} - \tilde{y}_t).
\end{aligned}
$$

For $R_{\text{Mix},\alpha}$, we evaluate $\alpha \in \{0.1, 0.2, 0.3\}$. Each model is trained for 2K iterations and evaluated on the in-distribution validation set using identical initial contexts. Table 4 reports the resulting mean normalized improvement and 95% confidence intervals.

Table 4: Reward formulation ablation on the in-distribution validation set. Higher mean normalized improvement is better.

Method	Mean	$\text{CI}_{95\%}$ low	$\text{CI}_{95\%}$ high
R_{Current}	0.1522	0.1403	0.1641
$R_{\text{Global Imp}}$	0.1120	0.1038	0.1202
$R_{\text{Global Imp},c}$	**0.1990**	0.1843	0.2138
R_{SMAPE}	0.1669	0.1562	0.1775
$R_{\text{Mix},\alpha=0.1}$	0.1354	0.1241	0.1467
$R_{\text{Mix},\alpha=0.2}$	0.1385	0.1268	0.1503
$R_{\text{Mix},\alpha=0.3}$	0.1401	0.1283	0.1519

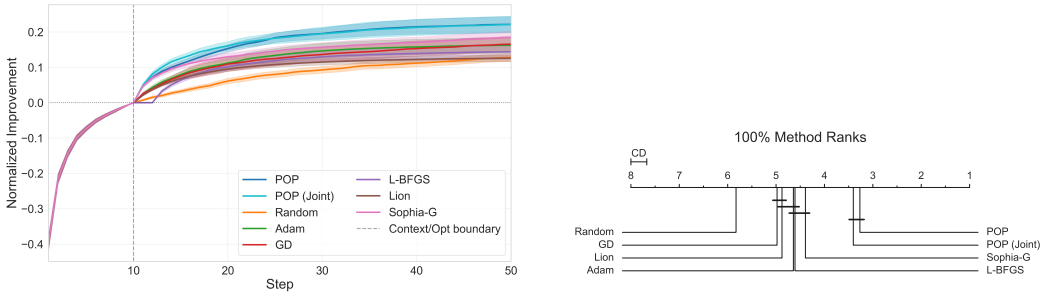
The clipped global-improvement reward, $R_{\text{Global Imp},c}$, achieves the highest mean normalized improvement. This suggests that clipping negative improvements is beneficial because exploratory actions that temporarily move away from the current best solution are not directly penalized.

D.2 Coordinate-wise Independence Ablation

We compare POP with a matched Joint POP variant that models cross-coordinate interactions. POP processes each coordinate independently, conditioning only on its transformed coordinate-wise history and predicting a single step size $\eta_{t,d}$. In contrast, Joint POP attends jointly across all D coordinates and predicts the full step-size vector $\eta_t \in \mathbb{R}^D_{>0}$, allowing each step size to depend on the positions and gradients of the other coordinates.

Figure 15a compares the two methods on the 2D test distribution, while Figure 15b shows their ranks relative to the remaining baselines. The two variants achieve nearly identical final normalized improvement (POP: 0.2217; Joint POP: 0.2210) and similar mean ranks (POP: 3.27; Joint POP: 3.40). These results suggest that, in this setting, explicitly modeling cross-coordinate interactions provides little additional benefit over POP's coordinate-wise formulation.

We further evaluate POP on a rotated Discus objective, which is strongly ill-conditioned and non-axis-aligned. As shown in Figure 16, POP reliably approaches the optimum despite the unfavorable geometry. The policy initially increases its learning rate to take larger steps and subsequently reduces it as optimization approaches the optimum.



(a) Mean normalized improvement of POP and Joint POP on the 2D test distribution.

(b) Critical difference diagram comparing POP and Joint POP with the optimization baselines.

Figure 15: Comparison of coordinate-wise POP and Joint POP on the 2D test distribution.

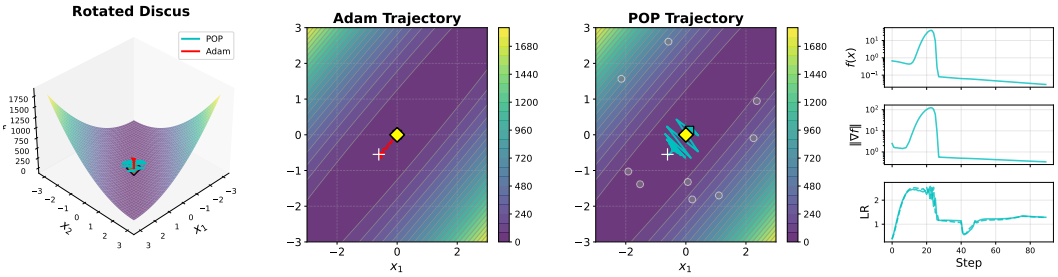


Figure 16: POP on the rotated Discus objective. The objective is strongly ill-conditioned and non-axis-aligned. Despite processing coordinates independently, POP approaches the optimum while adapting its learning rate throughout the optimization trajectory.

D.3 Prior Component Ablation

We evaluate the contribution of the individual components of the synthetic prior used to meta-train POP. For each ablation, we mask the sampled weight of one prior component while leaving the remaining sampled weights unchanged, i.e., without renormalization. Each resulting variant is meta-trained independently for 8K iterations and evaluated on the VLSE benchmark using the same evaluation protocol as the full model.

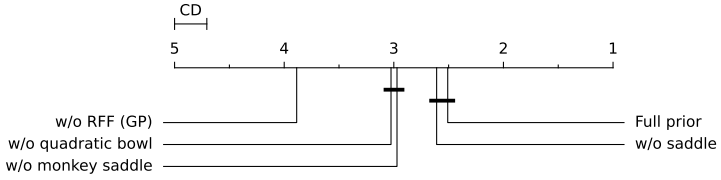


Figure 17: Critical difference diagram for the prior component ablation on VLSE.

As shown in Figure 17, the full prior achieves the best mean rank. Removing the RFF (GP) components leads to the largest degradation in performance, indicating that these components make the strongest individual contribution to out-of-distribution generalization among the ablated prior components.

E Extended Comparison Against Optimization Solvers

This appendix extends the evaluation in Section 6 to a broader set of optimization methods. We report extended comparisons for Hypotheses 1–3, covering in-distribution performance, generalization across optimization horizons, and out-of-distribution performance on VLSE. We additionally compare the computational cost of the evaluated methods through wall-clock latency measurements.

For the extended comparison we compare our method against a diverse set of optimization baselines. **Gradient Descent (GD)** (Cauchy, 1847), **Adam** (Kingma & Ba, 2015), and **Lion** (Chen et al., 2023) represent first-order gradient-based optimizers, while **Sophia-G** (Liu et al., 2023) is a second-order-inspired optimizer that uses a lightweight diagonal Hessian approximation. **L-BFGS** (Liu & Nocedal, 1989) is a quasi-Newton method leveraging approximate second-order information. **Random Search** provides a simple derivative-free baseline, while **Genetic Algorithms** (Holland, 1992), **Differential Evolution** (Storn & Price, 1997), and **CMA-ES** (Hansen, 2016) are population-based evolutionary methods for global optimization. **TPE** (Bergstra et al., 2011), **GP**-based Bayesian optimization (Rasmussen & Williams, 2006), and **HEBO** (Cowen-Rivers et al., 2022) are surrogate-based Bayesian optimization methods. Finally, **VeLo** (Metz et al., 2022) and **PFNs4BO** (Müller et al., 2023) are learned optimization baselines: VeLo learns an optimizer from a prior over neural network training tasks, while PFNs4BO uses a prior-data fitted network as a learned surrogate for Bayesian optimization. All baselines are evaluated under the same optimization budget.

E.1 Hypothesis 1: Extended comparison

We extend the in-distribution comparison from Section 6 to the broader set of optimization baselines described above.

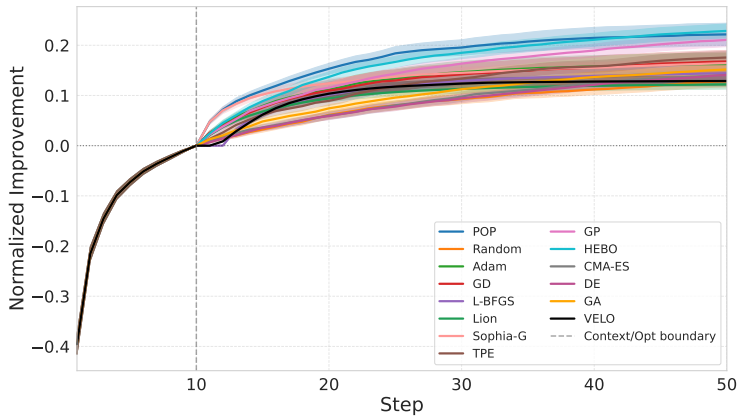


Figure 18: In-distribution test set performance vs. all baselines. Mean normalized improvement over steps; shading indicates 95% CIs. Dashed line marks the context/optimization boundary.

E.2 Hypothesis 2: Extended comparison

We next extend the horizon generalization experiment to the broader baseline set, evaluating all methods under the 100-step budget.

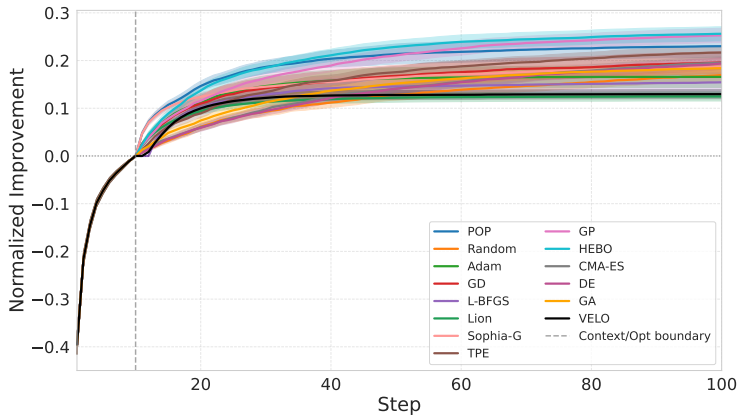


Figure 19: In-distribution test set performance vs. all baselines (with T=90). Mean normalized improvement over steps; shading indicates 95% CIs. Dashed line marks the context/optimization boundary.

E.3 Hypothesis 3: Extended comparison

Finally, we extend the out-of-distribution evaluation on VLSE to additional evolutionary, Bayesian optimization, and learned optimization methods.

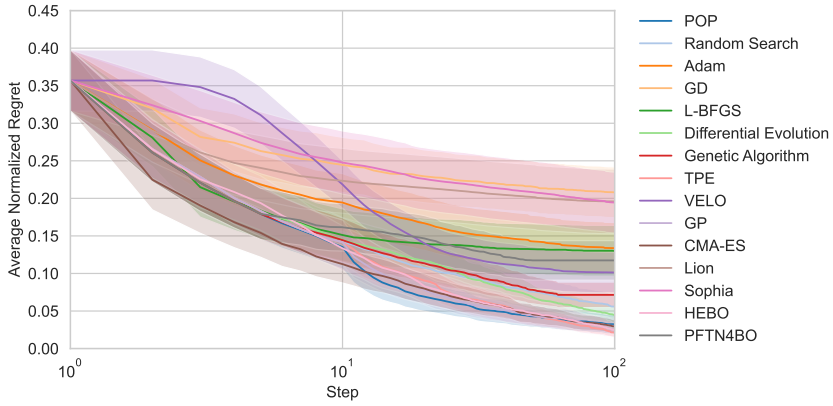
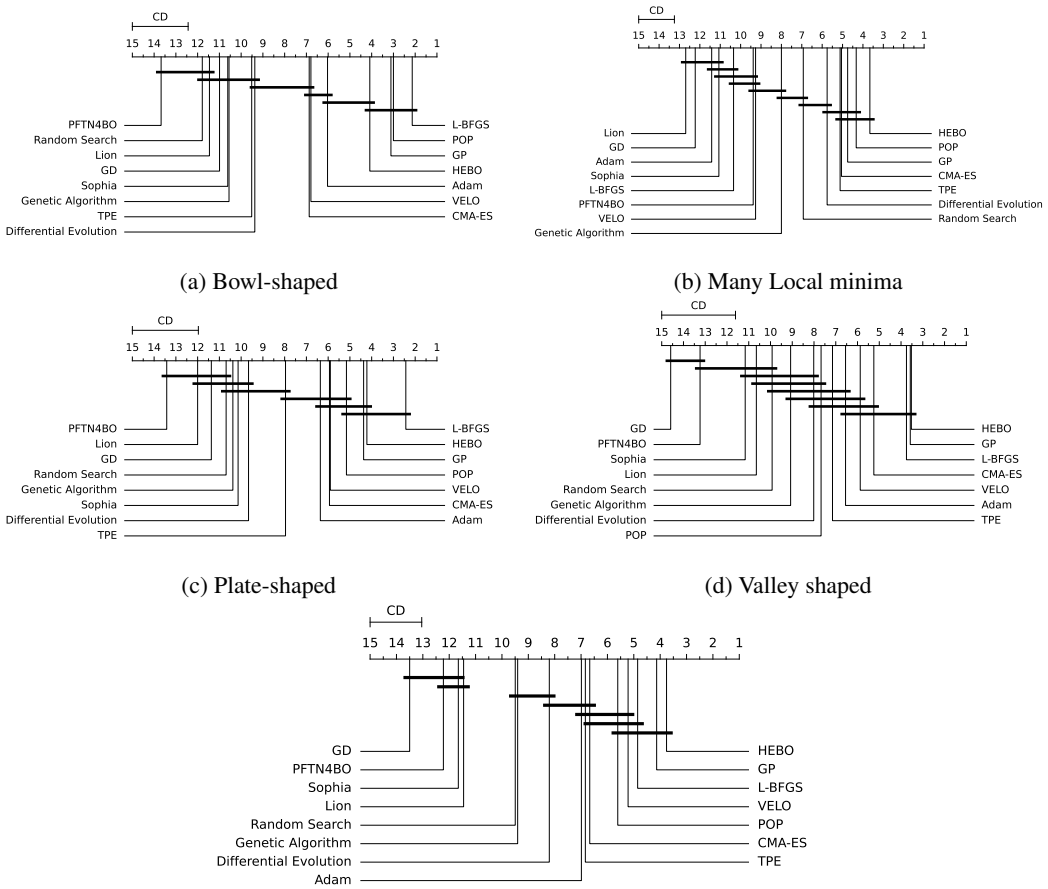


Figure 20: The average normalized regret for all methods on VLSE. Solid lines represent the mean value.

Figure 20 shows the average normalized regret of POP and all baselines, aggregated over the full VLSE benchmark.



(a) Bowl-shaped

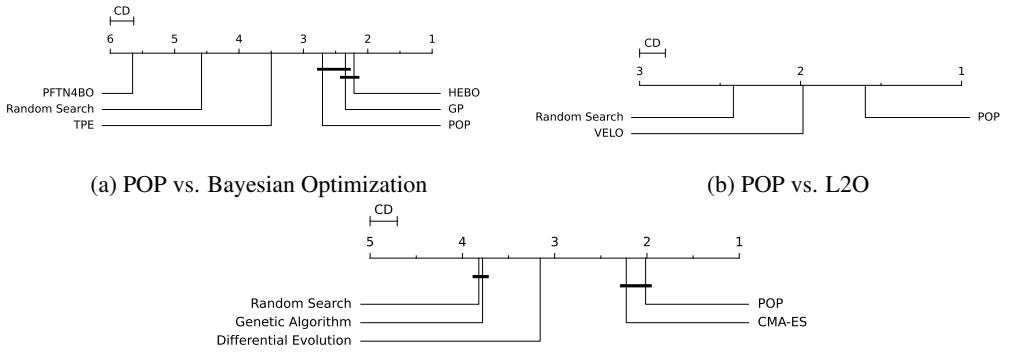
(b) Many Local minima

(c) Plate-shaped

(d) Valley shaped

(e) Irregular

Figure 21: Performance of POP compared to individual competitors from different model categories on diverse function classes



(a) POP vs. Bayesian Optimization

(b) POP vs. L2O

(c) POP vs. Evolutionary Methods

Figure 22: Performance of POP compared across different model categories and representative models.

Figure 21 shows a more detailed comparison of POP across different model families and their representative methods. In the out-of-distribution setting, POP performs comparably to the GP-based Bayesian optimization family, with no statistically significant difference (cf. Figure 22a). Notably, this

competitive performance comes at a substantially lower computational cost see Appendix E.4 POP further shows superior performance and achieves the lowest rank among the other four paradigms: first-order methods (cf. Figure 8a), second-order methods (cf. Figure 8b), learned optimizers (L2O; cf. Figure 22b), and evolutionary strategies (cf. Figure 22c).

Figure 23 shows the critical difference diagram for the 20 dimension-scalable VLSE functions at $d = 32$. L-BFGS and tuned Adam attain the lowest mean ranks, followed by POP, which ranks ahead of GD, Lion, Sophia-G, and random search, despite being applied zero-shot at $16\times$ the dimensionality seen during meta-training, while all gradient-based baselines were tuned per function.

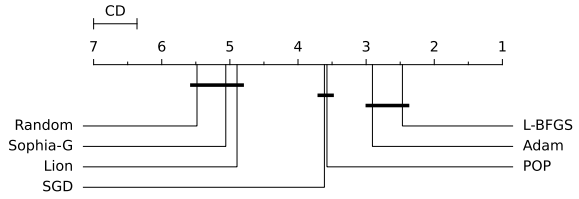


Figure 23: Critical difference diagram on the 20 dimension-scalable VLSE functions at $d = 32$. Lower rank is better; bars connect methods without significant differences.

E.4 WALL-CLOCK OPTIMIZATION LATENCY

While our main evaluation compares methods under matched optimization-step budgets, different optimizers incur substantially different computational costs when selecting the next point. We therefore additionally measure wall-clock decision latency on the 6D Hartmann function from the VLSE benchmark. All methods are evaluated on the same Apple M3 Pro machine using CPU-only execution. We report the average time required to produce one optimization decision over a 100-step optimization run. For POP, the reported latency includes the state preprocessing and normalization required before the policy evaluation and therefore does not correspond to a pure neural-network forward pass.

Table 5: Wall-clock latency per optimization decision on the 6D Hartmann function. All methods are executed on CPU on an Apple M3 Pro.

Method	**Per decision (ms)**
DE, GD, Sophia-G, Lion, Adam, L-BFGS, Random, GA	$< 0.213 \pm 0.019$
CMA-ES	0.459 ± 0.086
VeLO*	0.579 ± 0.212
TPE	7.35 ± 2.42
POP	18.3 ± 0.332
GP	36.1 ± 17.8
PFNs4BO	122 ± 10.5
HEBO	1353 ± 375

*VeLO additionally requires approximately 2.4 s per function for building and compilation.

POP introduces more overhead than lightweight first-order methods such as Adam, GD, and L-BFGS which have been heavily optimized by the community. Nevertheless, POP remains faster per decision than the GP, PFNs4BO, and HEBO baselines in this setting. Thus, although POP relies on a Transformer backbone, its policy-evaluation overhead does not dominate the computational cost relative to several competitive global optimization approaches.

Because POP conditions on an optimization trajectory whose length increases over time, we additionally measure its latency at different optimization horizons. Table 6 reports the mean per-decision latency for optimization budgets of 50, 100, and 200 steps.

Table 6: POP wall-clock latency across different optimization budgets.

Optimization budget	Per decision (ms)
50	18.32 ± 0.42
100	18.36 ± 0.24
200	18.83 ± 0.24

For the optimization horizons considered here, the increase in latency due to the growing trajectory context is modest: the average decision time increases from 18.32 ms at 50 steps to 18.83 ms at 200 steps. We emphasize, however, that computational efficiency was not an explicit objective of the current proof-of-concept implementation. For substantially longer trajectories, both the retained optimization history and the attention computation of the transformer are expected to introduce additional computational and memory costs.

F Real-World Transfer

F.1 Gaussian Process Regression

Setup. Let $\mathcal{D} = \{(x_n, y_n)\}_{n=1}^N$ with $x_n \in \mathbb{R}^d$ and $y_n \in \mathbb{R}$ denote the (standardized) training set, and write $X \in \mathbb{R}^{N\times d}$, $y \in \mathbb{R}^N$ for the stacked design matrix and targets. We place a zero-mean GP prior on the latent regression function f,

$$f \sim \mathcal{GP}\big(0,\ k(\cdot,\cdot;\theta)\big), \tag{11}$$

with observation model $y_n = f(x_n) + \varepsilon_n$, $\varepsilon_n \overset{\text{i.i.d.}}{\sim} \mathcal{N}(0, \sigma_n^2)$.

The covariance function is an automatic-relevance-determination (ARD) squared-exponential (RBF) kernel,

$$k(x, x'; \theta) = \sigma_f^2 \exp\left(-\frac{1}{2}\sum_{i=1}^{d}\frac{(x_i - x_i')^2}{\ell_i^2}\right), \tag{12}$$

with one lengthscale $\ell_i > 0$ per input dimension, a signal (output) variance $\sigma_f^2 > 0$, and observation noise variance $\sigma_n^2 > 0$. The optimization variable is the hyperparameter vector

$$\theta = (\ell_1, \ldots, \ell_d, \sigma_f^2, \sigma_n^2) \in \mathbb{R}^{d+2}_{>0}, \qquad P := d + 2. \tag{13}$$

To perform unconstrained gradient-based optimization while enforcing positivity, each entry of θ is stored as a *raw* (unconstrained) parameter $\tilde\theta \in \mathbb{R}^P$ and mapped through a shifted softplus,

$$\ell_i = \mathrm{softplus}(\tilde\ell_i) + c_\ell, \qquad \mathrm{softplus}(z) = \log(1 + e^z), \tag{14}$$

$$\sigma_f^2 = \mathrm{softplus}(\tilde\sigma_f) + c_f, \tag{15}$$

$$\sigma_n^2 = \mathrm{softplus}(\tilde\sigma_n) + c_n, \tag{16}$$

with small positive floors $c_\ell = 10^{-3}$, $c_f = 10^{-3}$, $c_n = 10^{-4}$ that keep the kernel matrix numerically well-conditioned. An optimizer acts on the raw vector $\tilde\theta \in \mathbb{R}^P$; $\theta = g(\tilde\theta)$ denotes the softplus map above.

Given θ, the training targets are jointly Gaussian,

$$y \mid X, \theta \sim \mathcal{N}(0,\ K_\theta + \sigma_n^2 I_N), \qquad [K_\theta]_{nm} = k(x_n, x_m; \theta), \tag{17}$$

so the log marginal likelihood is

$$\log p(y \mid X, \theta) = -\frac{1}{2} y^\top C_\theta^{-1} y - \frac{1}{2}\log|C_\theta| - \frac{N}{2}\log(2\pi), \qquad C_\theta := K_\theta + \sigma_n^2 I_N. \tag{18}$$

Given a test input $x_\star$, the GP posterior predictive is Gaussian with mean and variance

$$\mu(x_\star) = k_\star^\top C_\theta^{-1} y, \qquad k_\star := \big(k(x_\star, x_n; \theta)\big)_{n=1}^N, \tag{19}$$

$$\Sigma(x_\star) = k(x_\star, x_\star; \theta) - k_\star^\top C_\theta^{-1} k_\star. \tag{20}$$

Training Objective. The training loss is the *per-sample-normalized negative* log marginal likelihood,

$$\mathcal{L}_{\text{train}}(\theta) := -\frac{1}{N}\log p(y \mid X, \theta) = \frac{1}{N}\left(\frac{1}{2} y^\top C_\theta^{-1} y + \frac{1}{2}\log|C_\theta|\right) + \frac{1}{2}\log(2\pi). \tag{21}$$

Equation (21) is evaluated exactly via a Cholesky factorization $C_\theta = LL^\top$ (with a small numerical jitter $\eta = 10^{-6}$ added to the diagonal of K_θ for stability), giving

$$y^\top C_\theta^{-1} y = \|\alpha\|_2^2, \quad L\alpha' = y,\ L^\top \alpha = \alpha', \qquad \log|C_\theta| = 2\sum_{n=1}^{N}\log L_{nn}. \tag{22}$$

Because the kernel Gram matrix couples every training point, there is no minibatching: each optimization step is a single **full-batch** gradient step over the entire training set, and gradients $\nabla_{\tilde\theta}\,\mathcal{L}_{\text{train}}$ are obtained by automatic differentiation through the Cholesky solve.

Validation and test performance are reported as the mean squared error of the posterior predictive mean (Equation (19)) against held-out (standardized) targets, conditioning on the full training set:

$$\mathcal{L}_{\text{eval}}(\theta) := \frac{1}{M} \sum_{m=1}^{M} \big(\mu(x_m^{\text{ho}}) - y_m^{\text{ho}}\big)^2, \tag{23}$$

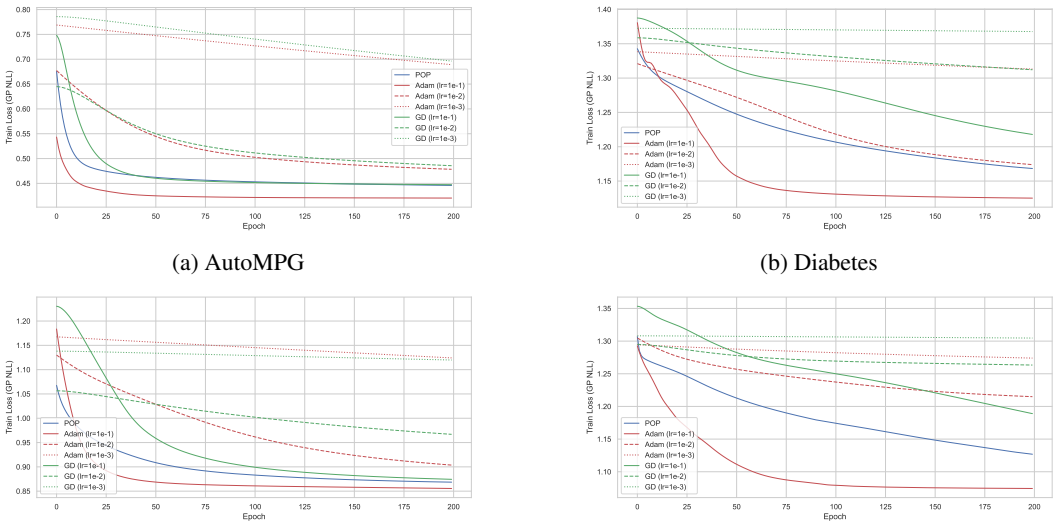
where $\{(x_m^{\text{ho}}, y_m^{\text{ho}})\}_{m=1}^{M}$ is the held-out (val or test) split.

Results. All methods start from the same random-restart procedure. K candidate raw vectors $\{\tilde{\theta}^{(k)}\}_{k=1}^{K}$ are drawn (lengthscale and output-scale components $\sim \mathcal{N}(0, 0.7^2)$, noise component $\sim \mathcal{N}(-1.5, 0.7^2)$). The training NLL equation 21 is evaluated at each candidate, and optimization proceeds from

$$\tilde{\theta}_0 = \tilde{\theta}^{(k^\star)}, \qquad k^\star = \arg\min_{k \in \{1,\dots,K\}} \mathcal{L}_{\text{train}}\big(g(\tilde{\theta}^{(k)})\big). \tag{24}$$

All methods then run for $T = 200$ full-batch steps. Adam ($\epsilon = 10^{-4}$, no weight decay) and SGD (momentum 0.9, no weight decay) are applied directly to the raw vector $\tilde{\theta}_t$, each swept over a fixed learning rate grid $\eta \in \{10^{-1}, 10^{-2}, 10^{-3}\}$.

The training loss curves in Figure 24 indicates POP's competitive performance in training a gaussian process regression on the benchmarks AutoMPG, Diabetes, California housing datasets, and Wine Quality. Notably, POP does not require a thorough hyperparameter search on the learning rate and only tuned Adam is superior to POP in terms of the training loss. However, the slightly better training loss is not reflected in a better MSE test loss as shown in Figure 25, where POP shows the best MSE on test data on two out of four benchmark datasets and is a close runner-up on diabetes, without any learning rate tuning of its own.



(a) AutoMPG (b) Diabetes

(c) California (d) Wine Quality

Figure 24: Training losses of the GP Regression Task on four benchmark datasets: AutoMPG, Diabetes, California, Wine Quality

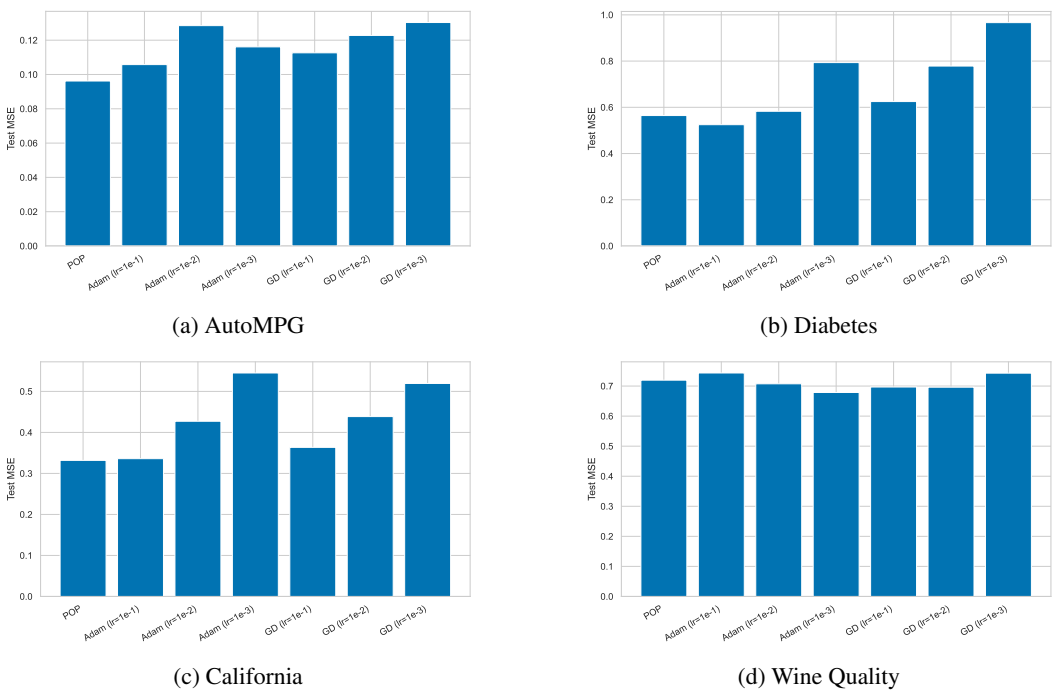
F.2 Neural Network Optimization

Setup. Each dataset is fit with the same two-layer network,

$$f_w(x) = W_2\ \text{GELU}\big(\text{LN}(W_1 x + b_1)\big) + b_2, \qquad x \in \mathbb{R}^d,\ f_w(x) \in \mathbb{R}^C, \tag{25}$$

where d is the number of input features, C the number of output classes, $\text{LN}(\cdot)$ is LayerNorm applied to the hidden pre-activation, and GELU is the Gaussian-error-linear-unit nonlinearity. The hidden width is fixed at $H = 32$ across all datasets. The full parameter vector is

$$w = (W_1, b_1, \gamma_{\text{LN}}, \beta_{\text{LN}}, W_2, b_2) \in \mathbb{R}^P, \tag{26}$$



(a) AutoMPG (b) Diabetes

(c) California (d) Wine Quality

Figure 25: MSE on test data for the GP Regression Task on four benchmark datasets: AutoMPG, Diabetes, California, Wine Quality

where $P = dH + H + 2H + HC + C$ (the two extra H-blocks are the LayerNorm scale γ_{LN} and shift β_{LN}).

Training Objective. The training loss is the mean cross-entropy over the batch,

$$\mathcal{L}_{\text{train}}(w) = -\frac{1}{N}\sum_{n=1}^{N} \log\left[\text{softmax}(f_w(x_n))\right]_{y_n}, \qquad y_n \in \{1, \ldots, C\}. \tag{27}$$

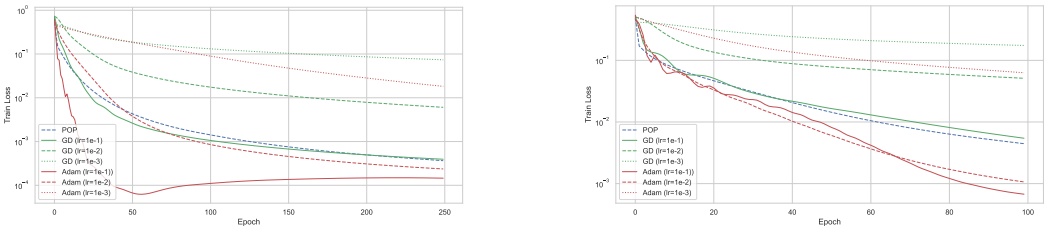
Held-out performance is reported both as cross-entropy (Eq. equation 27 evaluated on val/test) and as classification accuracy, $\frac{1}{M}\sum_{m=1}^{M} \mathbb{1}[\arg\max_c f_w(x_m)_c = y_m]$.

Results. All methods start from the best of $K = 10$ random restarts: K candidate weight vectors are drawn from the Xavier-style initialization scheme, the training loss equation 27 is evaluated for each on the first training batch, and optimization proceeds from the restart with the lowest loss,

$$w_0 = \arg\min_{k \in \{1,\ldots,K\}} \mathcal{L}_{\text{train}}(w^{(k)}). \tag{28}$$

Adam ($\epsilon = 10^{-4}$, weight decay 10^{-4}) and SGD (momentum 0.9, weight decay 10^{-4}) are applied directly to w_t, each swept over a fixed learning rate grid $\eta \in \{10^{-1}, 10^{-2}, 10^{-3}\}$.

The training loss curves on a 2-layer MLP are shown in Figure 26. Notably, without task-specific tuning, POP consistently reduces the training loss substantially. Generally, tuned conventional optimizers like Adam and SGD generally achieve lower final losses. The results indicate limited zero-shot transferability from synthetic objectives to unseen ML objectives. We leave the investigation of our prior's expressiveness w.r.t. training objectives of neural networks for future work.



(a) Banknote (b) Breastcancer

Figure 26: Training loss for a 2-layer MLP on two benchmark datasets: Banknote and Breastcancer

G On the Universality of Gaussian Process Prior for Pretraining Learnable Optimizers

This appendix summarizes theoretical consequences of classical results from the theory of Gaussian processes (GPs), reproducing kernel Hilbert spaces (RKHSs), and kernel interpolation in order to motivate our choice of GP prior. These results are included for completeness of the theoretical background and are not claimed as theoretical contributions of this paper; rather, they follow from established results on universal kernels, Gaussian process regression, and kernel interpolation (Rasmussen & Williams, 2006; Wendland, 2004; Steinwart & Christmann, 2008). In particular, the universal approximation property of a GP is determined by its covariance function, or kernel: universality is not a property of all GPs, but follows when the GP is endowed with a *universal kernel*. The corresponding approximation result relies on the connection between GPs and their associated reproducing kernel Hilbert spaces. When the GP is approximated using Random Fourier Features, an additional approximation error is introduced, which we absorb into ϵ.

Theorem. Let $K \subset \mathbb{R}^d$ be a compact set. Let $k : K \times K \to \mathbb{R}$ be a universal kernel. Let $f \in C(K)$ be any continuous function on K. Then, for any $\epsilon > 0$, there exists a set of points $X_n = \{x_1, \ldots, x_n\} \subset K$ and corresponding values $Y_n = \{f(x_1), \ldots, f(x_n)\}$ such that the posterior mean $\mu_n(x)$ of a Gaussian Process with prior mean $m(x) = 0$ and kernel $k(x, x')$ satisfies:

$$\sup_{x \in K} \big| f(x) - \mu_n(x) \big| < \epsilon.$$

The theorem is an existence result from classical kernel approximation theory and is included only to illustrate the approximation capability of universal kernels.

G.1 Preliminaries: Gaussian Processes and RKHS

A Gaussian Process $\mathrm{GP}(m(x), k(x, x'))$ is a stochastic process where any finite collection of random variables has a multivariate normal distribution. It is fully specified by its mean function $m(x)$ and covariance function (kernel) $k(x, x')$. For simplicity, we assume a zero prior mean, $m(x) = 0$.

Given a set of n noiseless observations $(X_n, Y_n) = \{(x_i, y_i)\}_{i=1}^n$ where $y_i = f(x_i)$, the posterior distribution of the GP at a new point x^* is also Gaussian. The posterior mean is given by:

$$\mu_n(x^*) \;=\; k(x^*, X_n)\, K_n^{-1}\, Y_n,$$

where $k(x^*, X_n) = [k(x^*, x_1), \ldots, k(x^*, x_n)]$ is a row vector and K_n is the $n \times n$ Gram matrix with entries $(K_n)_{ij} = k(x_i, x_j)$.

Associated with every positive definite kernel k is a unique Reproducing Kernel Hilbert Space (RKHS), denoted $\mathcal{H}_k$. This is a Hilbert space of functions on K with two key properties:

1. For every $x \in K$, the function $k_x(\cdot) := k(x, \cdot)$ is in $\mathcal{H}_k$.
2. (*Reproducing property*) For any $g \in \mathcal{H}_k$ and $x \in K$, we have $\langle g, k_x \rangle_{\mathcal{H}_k} = g(x)$.

A crucial result from GP theory is that the posterior mean $\mu_n(x)$ is the function in $\mathcal{H}_k$ that minimizes the RKHS norm $\|g\|_{\mathcal{H}_k}$ subject to the interpolation constraints $g(x_i) = y_i$ for all $i = 1, \ldots, n$.

G.2 Universal Kernels

The approximation power of a GP is entirely determined by the richness of its RKHS. This leads to the definition of a universal kernel.

Definition (Universal Kernel). A continuous kernel k on a compact set K is called *universal* if its corresponding RKHS $\mathcal{H}_k$ is dense in the space of continuous functions $C(K)$ with respect to the uniform norm (L^∞).

This means that for any continuous function $f \in C(K)$ and any $\delta > 0$, there exists a function $g \in \mathcal{H}_k$ such that:

$$\sup_{x \in K} |f(x) - g(x)| < \delta.$$

Examples of universal kernels include:

- **Gaussian (RBF) Kernel:**

$$k(x, x') = \exp\left(-\frac{\|x - x'\|^2}{2\sigma^2}\right).$$

- **Matérn Kernels:**

$$k(x, x') = \frac{2^{1-\nu}}{\Gamma(\nu)} \left(\frac{\sqrt{2\nu}\,\|x - x'\|}{\ell}\right)^{\nu} K_\nu \left(\frac{\sqrt{2\nu}\,\|x - x'\|}{\ell}\right), \qquad \nu > 0,$$

  where K_ν is the modified Bessel function of the second kind.

- **Laplacian Kernel:**

$$k(x, x') = \exp\left(-\frac{\|x - x'\|}{\sigma}\right).$$

The universality of these kernels is often established using characterizations like Bochner's theorem (a kernel is universal if its Fourier transform is positive on the entire frequency domain).

G.3 Proof of the Theorem

Let k be a universal kernel on a compact set K, and let $f \in C(K)$ be the target function. Let $\epsilon > 0$ be the desired precision.

Step 1: Density of RKHS in $C(K)$

Since k is a universal kernel, its RKHS $\mathcal{H}_k$ is dense in $C(K)$. Therefore, for our given $\epsilon > 0$, there exists a function $g \in \mathcal{H}_k$ such that:

$$\sup_{x \in K} |f(x) - g(x)| < \frac{\epsilon}{2}.$$

Step 2: Convergence of the Interpolant

Now we need to show that we can choose data points X_n such that the GP posterior mean $\mu_n(x)$ (which interpolates f on X_n) becomes arbitrarily close to $g(x)$.

Let g be the function from Step 1. Since $g \in \mathcal{H}_k$, it has a finite RKHS norm, $\|g\|_{\mathcal{H}_k} < \infty$. Let us choose a sequence of points $X_n = \{x_1, \ldots, x_n\}$ that becomes dense in K as $n \to \infty$.

Let μ_n^g be the GP posterior mean interpolating the values of g on X_n, i.e., $\mu_n^g(x_i) = g(x_i)$. It is a standard result in the theory of RKHS (see Wendland, *Scattered Data Approximation*) that if X_n becomes dense in K, then the sequence of interpolants μ_n^g converges uniformly to g:

$$\lim_{n \to \infty} \sup_{x \in K} |\mu_n^g(x) - g(x)| = 0.$$

Therefore, we can choose n large enough such that:

$$\sup_{x \in K} |\mu_n^g(x) - g(x)| < \frac{\epsilon}{2}.$$

STEP 3: COMBINING THE APPROXIMATIONS

Let μ_n^f be the GP posterior mean interpolating the target function f on the same set of points X_n, i.e., $\mu_n^f(x_i) = f(x_i)$.

The posterior mean is a linear operator on the data values. Let

$$L_n(Y)(x) = k(x, X_n)K_n^{-1}Y.$$

Then $\mu_n^f = L_n(f(X_n))$ and $\mu_n^g = L_n(g(X_n))$.

We have:

$$\sup_{x\in K} |\mu_n^f(x) - \mu_n^g(x)| = \sup_{x\in K} |L_n(f(X_n) - g(X_n))(x)| \leq \|L_n\|_\infty \max_{i=1,\dots,n} |f(x_i) - g(x_i)|.$$

While $\|L_n\|_\infty$ can grow with n, we can proceed using the triangle inequality:

$$\sup_{x\in K} |f(x) - \mu_n^f(x)| \leq \sup_{x\in K} |f(x) - g(x)| + \sup_{x\in K} |g(x) - \mu_n^g(x)| + \sup_{x\in K} |\mu_n^g(x) - \mu_n^f(x)|.$$

We have already bounded the first two terms. For the third term:

$$\mu_n^g(x) - \mu_n^f(x) = \sum_{i=1}^{n} c_i\, k(x, x_i),$$

where the coefficients satisfy

$$c = K_n^{-1}\big(g(X_n) - f(X_n)\big).$$

Hence,

$$\sup_{x\in K} |\mu_n^g(x) - \mu_n^f(x)| \leq C_n \max_i |g(x_i) - f(x_i)| < C_n\, \frac{\epsilon}{2},$$

where C_n depends on X_n.

A more elegant argument considers the projection of f onto the subspace spanned by $\{k(x_i, \cdot)\}_{i=1}^n$. The posterior mean μ_n^f is precisely this projection. As $n \to \infty$ and X_n becomes dense in K, the span of $\{k(x_i, \cdot)\}_{i=1}^n$ becomes dense in $\mathcal{H}_k$.

Thus, the sequence μ_n^f converges to the projection of f onto $\mathcal{H}_k$, denoted $P_{\mathcal{H}_k} f$. So, for large enough n:

$$\sup_{x\in K} |\mu_n^f(x) - P_{\mathcal{H}_k} f(x)| < \frac{\epsilon}{2}.$$

By the property of projections, $P_{\mathcal{H}_k} f$ is the best approximation to f within $\mathcal{H}_k$. Since $\mathcal{H}_k$ is dense in $C(K)$, we can find $g \in \mathcal{H}_k$ such that $\|f - g\|_\infty < \epsilon/2$. This implies that $\|f - P_{\mathcal{H}_k} f\|_\infty < \epsilon/2$. Combining these gives:

$$\sup_{x\in K} |f(x) - \mu_n^f(x)| < \frac{\epsilon}{2} + \frac{\epsilon}{2} = \epsilon.$$

CONCLUSION

The proof establishes that for any continuous function f and any precision ϵ, one can find a finite set of sample points from f such that the posterior mean of a GP with a universal kernel approximates f to within ϵ over the entire compact domain. The universality of the GP is therefore inherited directly from the denseness of its associated RKHS in the space of continuous functions. This is a powerful existence result, though it does not specify the number of points n required or the computational cost. The theorem serves as theoretical background for the proposed prior, while the experimental section evaluates its practical behavior.