α-PFN: IN-CONTEXT LEARNING ENTROPY SEARCH

Tom Viering[*♣], **Steven Adriaensen**[*♢], **Herilalaina Rakotoarison**[*♢],
Samuel Müller[♡], **Carl Hvarfner**[♠], **Frank Hutter**[★♦♢], **Eytan Bakshy**[♡]
Equal contribution[*], Delft University of Technology[♣], University of Freiburg[♢],
Lund University[♠], Meta[♡], Prior Labs[★], ELLIS Institute Tübingen[♦]
Correspondence to t.j.viering@tudelft.nl
and adriaens@informatik.uni-freiburg.de

ABSTRACT

We show how Prior-data Fitted Networks (PFNs) can be adapted to efficiently predict Entropy Search (ES), an information-theoretic acquisition function. PFNs were previously shown to be able to accurately approximate Gaussian Process (GP) predictions. To approximate ES we extend them to condition on information about the optimum of the underlying function. Conditioning on this information is not straightforward and previous methods relied on complex, handcrafted, and/or computationally heavy approximations. PFNs, however, offer learned approximations that require just a single forward pass. Additionally, we train α-PFN, a new type of PFN model, on the information gains predicted by the first, letting us directly predict the value of the acquisition function in a single forward pass, effectively avoiding the traditional sampling-based approximations. This approach makes using Entropy Search and its variations straightforward and efficient in practice. We validate our approach empirically on synthetic GP samples of up to six dimensions, where the α-PFN matches or improves upon the regrets obtained by current approximations to predictive and joint Entropy Search, at a reduced computational cost. While this provides an initial proof of concept, the real potential of our method lies in its ability to efficiently perform Entropy Search for arbitrary function priors, unlike the current GP-specific approximations.

1 INTRODUCTION

Bayesian Optimization (BO) is designed to optimize a black-box function with few iterations. It is especially beneficial in situations where function evaluation is costly. BO finds use in various fields (Shahriari et al., 2015), such as tuning hyperparameters of large neural networks (Snoek et al., 2012; Feurer and Hutter, 2019). Here, the function being optimized is the performance on a validation set, which necessitates a full training run for evaluation, and BO aims to identify the hyperparameter setting that will maximize this performance. To this end, the canonical BO framework maintains a probabilistic regression surrogate model of the black-box function fit to the performance observations thus far, and maximizes an acquisition function quantifying the exploration-exploitation trade-off for the given posterior.

The information-theoretic class of acquisition functions (Villemonteix et al., 2009), such as Entropy Search (ES, Hennig and Schuler, 2012) offers a principled way to perform global optimization of Gaussian Process (GP) surrogate models (Williams and Rasmussen, 2006) by selecting queries that maximize the expected information gain regarding the location of the optimum. Although ES offers an elegant theoretical framework, its performance is hampered by complex approximation schemes, and unlike classic acquisition functions (e.g., Expected Improvement (EI; Močkus, 1975), upper confidence bound (UCB; Srinivas et al., 2010)) these do not have analytical solutions for GPs.

Previously, Entropy Search has been adapted to improve computational efficiency when used with GPs. Hernández-Lobato et al. (2014) show that it is possible to rewrite the Entropy Search acquisition function to depend on terms that are more tractable to estimate and approximate, resulting in an acquisition function which they call Predictive Entropy Search (PES). Wang and Jegelka (2017) proposed Max-value Entropy Search (MES) and most recently Hvarfner et al. (2022) and Tu et al. (2022) proposed Joint Entropy Search (JES). Both adjust the original Entropy Search formulation

to enable more efficient approximations and/or better BO performance. All of these rely on hand-crafted, sampling-based approximations to estimate quantities that are not easily computed for a Gaussian Process regression model, with PES requiring the most costly approximations (Hernández-Lobato et al., 2014; Tu et al., 2022).

Recently, it has been shown that Prior-Data Fitted Networks (PFN; Müller et al., 2021), a transformer-based conditional neural process (Garnelo et al., 2018) can be used to speed up training and prediction of Gaussian Process regression models (Müller et al., 2023). The key idea of Müller et al. (2023) is to use transformers to emulate prediction under the GP prior: the transformer takes as input the training data (inputs and targets) and test data (inputs without targets), and output the posterior predictive distribution for each test sample. By training on millions of samples from the GP prior, the transformer learns to perform Bayesian prediction in a single forward pass. Inference can be computationally more efficient compared to fully-Bayesian GPs, which require the use Markov Chain Monte Carlo (MCMC; Robert et al., 1999). Furthermore, Müller et al. (2023) showed that their approach is easily extended to other surrogate models, e.g., Bayesian neural networks, and permits conditioning on additional context information (e.g., user expert priors).

We explore how the same technique can be applied to approximate quantities used in Entropy Search. The Entropy Search acquisitions functions for PES, MES and JES all require multiple Monte Carlo samples and several approximations derived by hand. Instead of mathematically deriving approximations, the transformer learns how to approximate the quantities we need by itself, offering learned approximations in a single forward pass. Furthermore, PES, MES, and JES require MCMC sampling for the fully Bayesian GP settings. To avoid these costly samples, *we replace sampling by learning* by training a second transformer, α-PFN, which directly predicts the acquisition. We test the α-PFN's performance compared to the existing sampling-based approximations of PES, MES and JES in terms of optimization performance (inference regret) and walltime (runtime of the optimization), for the simplest setting: BO on synthetic GP samples for fixed hyperparameters. Crucially, the fully Bayesian GP setting and other surrogate models such as Bayesian neural networks are left for future work. As such, we provide an initial proof of concept demonstrating the approach.

2 Background, Notation and Related Work

In this section, we introduce key concepts and related work our method builds upon. Further related works are discussed in Appendix A.

2.1 Bayesian Optimization (BO)

In Bayesian Optimization (BO), the goal is to maximize a function $f(x)$ over a domain A. We denote $x^* = \arg\max_{x \in A} f(x)$ and $f^* = f(x^*)$. We assume for simplicity that $A = [0, 1]^d$. We only have black box access to $f(x)$, i.e., we can only obtain function evaluations which are typically corrupted with noise: $y = f(x) + \epsilon$. Bayesian Optimization is an iterative process, in each iteration t a query x_t is made to the black box function and the corresponding observation y_t is revealed. Afterward the surrogate model, such as Gaussian Process or transformer, is fitted to the collected data so far, i.e., it is fitted on $D_t = \{(x_1, y_1), \ldots, (x_t, y_t)\}$ to predict the posterior predictive distribution $p(y|D_t, x)$. From the surrogate model, an acquisition function is derived, which we write as $\alpha = \alpha(x, D_t)$, that is maximized to determine the next query $x_{t+1} = \arg\max_{x \in A} \alpha(x, D_t)$.

2.2 Entropy Search (ES)

The original Entropy Search (ES) method selects queries by maximizing the expected reduction in the entropy of the optimum location x^* (Hennig and Schuler, 2012):

$$\alpha_{ES}(x, D_t) = H(p(x^*|D_t)) - \mathbb{E}_{y \sim p(y|D_t, x)} \left[H(p(x^*|D_t \cup \{(x, y)\})) \right]. \tag{1}$$

Here, $H(p(x^*|D_t))$ represents the entropy of the posterior distribution over the optimum location given the observed data D_t. The first term is constant with respect to query selection and can be ignored during optimization. The second term represents the expected entropy after obtaining the new observation (x, y), where the expectation is taken over the predictive distribution $p(y|D_t, x)$.

Computing $p(x^*|D_t)$ is generally intractable, as the entropy of the GP's maximizer does not have a closed-form expression and requires averaging over multiple samples of y. To approximate this, Hennig and Schuler (2012) propose two methods: Monte Carlo sampling and Expectation Propagation (Minka, 2001). However, both approaches are computationally expensive, making ES impractical for many real-world applications.

Predictive Entropy Search (PES). The entropy reduction in ES can be expressed in terms of mutual information (MI) between x^* and y conditioned on D_t. Utilizing the symmetry of the MI in x^* and y, Hernández-Lobato et al. (2014) propose to optimize

$$\alpha_{PES}(x, D_t) = H(p(y|D_t, x)) - \mathbb{E}_{x^* \sim p(x^*|D_t)} \left[H(p(y|D_t, x, x^*)) \right] \tag{2}$$

which is equivalent to ES but allows more efficient approximations. The first term is analytically tractable since the posterior of the GP is Gaussian. Hernández-Lobato et al. (2014) propose a sequence of approximations to accurately estimate the entropy in the second term, and the outer expectation over x^* is distributed according to $p(x^*|D_t)$. To obtain these, sample paths from the GP posterior are approximated using random Fourier features (RFF, Rahimi and Recht, 2007). Each sample path is maximized to obtain draws of the posterior over x^*. The predictive posterior $p(y|D_t, x^*)$ cannot be obtained exactly. Thus, conditioning on tractable alternatives, such as convexity at x^*, and constraints such as $f(x^*) \geq \max_{i \in [1,t]} y_i$, serves to approximate it.

Max-value Entropy Search (MES). Max-value Entropy Search (MES; Wang and Jegelka, 2017) aims to reduce uncertainty over the maximum function value, f^*, by selecting the query point that maximizes the expected information gain:

$$\alpha_{MES}(x, D_t) = H(p(y|D_t, x)) - \mathbb{E}_{f^* \sim p(f^*|D_t)}, \left[H(p(y|D_t, x, f^*)) \right] \tag{3}$$

Here, the expectation is taken over the posterior distribution of f^* given D_t. In addition to the RFF sampling approach for f^* and x^* proposed by Hernández-Lobato et al. (2014), Wang and Jegelka (2017) propose an simpler alternative for f^* using a Gumbel distribution. Compared to PES, MES reduces the expectation from d dimensions to one. Moreover, they assume that $p(y|D_t, x, f^*)$ can be well-approximated by a truncated normal distribution, which enables an analytical entropy calculation. However, this assumption holds only in noiseless settings (Takeno et al., 2020; Nguyen et al., 2022). Due to the simpler approximation, MES is substantially faster than PES, and has seen multiple extensions, e.g. to parallel queries (Moss et al., 2021) (GIBBON-MES), noisy (Takeno et al., 2020; 2022) and multi-fidelity (Moss et al., 2021) problems.

Joint Entropy Search (JES). The distributions $p(x^*|D_t)$ and $p(f^*|D_t)$ can be multimodal and only give a limited view. Therefore Hvarfner et al. (2022) and Tu et al. (2022) propose to reduce the uncertainty on the joint distribution of the maximum value and its location:

$$\alpha_{JES}(x, D_t) = H(p(y|D_t, x)) - \mathbb{E}_{(x^*, f^*) \sim p(x^*, f^*|D_t)} \left[H(p(y|D_t, x, x^*, f^*)) \right] \tag{4}$$

The expectation is approximated by sampling in the same manner as PES. For each sample, the pair (x^*, f^*) is added to the GP's training set so that the posterior can be updated using regular GP machinery, conditioning either on a *noiseless* optimal value f^* (Hvarfner et al., 2022), or a $y^* = f^* + \varepsilon$ containing observation noise (Tu et al., 2022). Both Hvarfner et al. (2022) and Tu et al. (2022) use a local constraint to condition on the maximum similar to MES. The resulting extended skew distribution (Nguyen et al., 2022; Hvarfner et al., 2022; 2023) does not admit a closed form for the entropy, and is therefore approximated either by Monte Carlo sampling of the integral (Tu et al., 2022) or moment matching with a Gaussian (Moss et al., 2021; Hvarfner et al., 2022), lower bounding the MI (Moss et al., 2021).

2.3 Prior-data Fitted Networks (PFNs)

Prior-data Fitted Networks (PFNs; Müller et al., 2021) are transformer neural networks that learn to perform Bayesian predictions in a single forward pass by training on synthetic data sampled from a predefined prior $p(D)$, which defines a distribution over datasets D of input and output pairs (x,y). During training, datasets from the prior are split in two parts: a training and test set. The

transformer takes the training set (x,y)'s as input, and is trained to predict the correct outputs for the test set inputs. The transformer is a decoder-only model that applies an attention mask to make sure that while objects of the training and test set can attend to those in the training set, test set samples cannot attend to each other and are predicted independently. This allows the model to learn how to learn from a new dataset. After training the PFN, its weights are frozen. At inference time, the training set and (unlabeled) test set are provided as input to the transformer, which then predicts test targets during the forward pass. Note that no fitting takes place at test time, but rather the PFN learns from the data in the input — also called the context — and therefore this is called in-context learning (Brown et al., 2020). More specifically, it can be shown that the PFN performs in-context Bayesian prediction (Müller et al., 2021), i.e., the PFN is a model $q(y|x, D)$, which approximates the posterior predictive distribution (PPD) $p(y|D, x)$ for our prior $p(D)$. PFN training is the largest cost (typically training for 1 day on a single GPU), which needs to be performed once beforehand. Afterward, the PFN can offer significantly reduced inference times.

The versatility of this paradigm, has lead to a variety of applications. PFNs has been used as a surrogate model in black-box (Müller et al., 2023) and Freeze-thaw Bayesian Optimization (Rakotoarison et al., 2024); as a model for forecasting time-series (Dooley et al., 2024) and learning curves (Adriaensen et al., 2024; Viering et al., 2024); and as a foundation model for tabular data (Hollmann et al., 2025).

Most relevant to our work, PFNs have been used to accurately approximate Gaussian Process regression (Müller et al., 2021; 2023), where datasets consist of pairs of feature vectors x and the regression targets y, for some function f sampled from a Gaussian Process prior. The feature vectors x and the regression target y for the transformer are encoded via a single-layer feed-forward neural network to generate an embedding for x and an embedding for y which are added together. The output of the transformer is a bar distribution for each test point. The bin sizes are optimized to work well for predicting targets from the prior. The bar distribution is trained via cross entropy loss. As such, the regression task is essentially treated as a classification task by binning the y's.

3 Extending PFNs to Entropy Search

We develop a methodology to perform Entropy Search using PFNs for PES, MES, and JES. To do this efficiently, we construct two PFN models. First, we train a base PFN model to learn how to do Bayesian prediction conditioned on information of the location and/or value of the optimum. While this base PFN can help generate more accurate Monte Carlo estimates for these acquisition functions, this approach still requires optimizing samples from the posterior to estimate the expected information gain, which is computationally expensive. Therefore, we train a second PFN model, the α-PFN, which directly predicts the acquisition function–PES, MES, or JES—in a single forward pass, using the information gain predicted by the base PFN model.

Precomputing Gaussian Process prior data. To construct our PFN models for GP inference, we need to train the model on millions of samples from a GP prior. Furthermore, we need to know x^* and f^* for each GP, which is quite expensive to compute. To make this more tractable, we approximate GP samples using Random Fourier Features (RFFs; Rahimi and Recht, 2007), whereby sample paths from the Gaussian Process are represented by a weight vector w in RFF space. To obtain a sample path, one simply samples a new w. By fixing w, it is possible to evaluate the sample path at any position x. For a fixed w, it is thus possible to maximize a function draw from a GP over the domain A to estimate x^* and f^*. Note that this is a difficult global optimization problem, which we solve approximately by an ensemble of optimizers that are restarted multiple times. See Appendix B for more details.

Learning the base PFN model to condition on the maximum of a GP. See Figure 1 which illustrates the predictions of the PFN after conditioning. When we are training on GP data, we have access to x^* and f^*. The idea is to feed these into the context of the PFN during training. The best estimates are obtained, in expectation, by taking this conditioning information into account. Since the PFN is trained with performance in expectation, if trained well, it therefore should learn to perform conditioning — see also the argument by Müller et al. (2021) that the transformer learns to perform Bayesian Inference. Thus if we feed in x^*, the PFN approximates $p(y|x, D, x^*)$ by $q(y|x, D, x^*)$. The same goes when conditioning on f^* or on both (x^*, f^*). We train a single

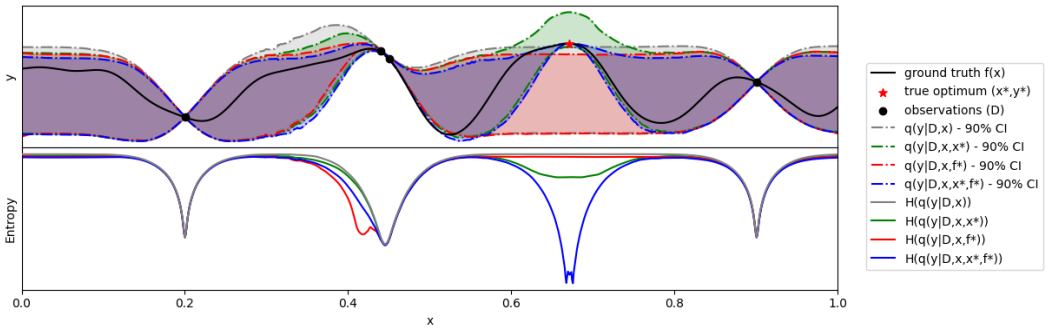

Figure 1: Here we illustrate the effect on the predictions made by the base PFN in 1D when conditioning on different types of information (top) and we plot the corresponding entropy $H(q(y|D, x, I))$ for the domain after conditioning (bottom). Top: when conditioning on $I = x^*$ (green line), note that the upper credible interval of the PPD around x^* increases as expected. Conditioning on $I = f^*$ (red) constrains the upper credible interval $\leq f^*$ globally, which given the near optimal observations, reduces this bound compared to the unconditioned model (grey). Conditioning on both (blue) as expected shrinks the credible interval around the true optimum (red star) to zero, while imposing a global max-value constraint.

base model to support all four cases: unconditional, conditioning on x^*, f^* and (x^*, f^*). These cases are all equally probable during training. For more training details see Appendix C. Note that our base PFN can be used an alternative to handcrafted approximations of the corresponding right hand terms in equations 2, 3, and 4 since the entropy H of its output bar distribution can be computed efficiently. This enables a universal approach to approximate all three Entropy Search variants, sampling x^* and/or f^* by optimizing samples from the posterior to produce a Monte Carlo estimate of the expected information gain in Equations 2, 3, and 4. That being said, depending on the posterior, it may be costly to obtain enough samples to sufficiently reduce the variance of this estimate. This motivates exploring α-PFN as an alternative, a learning-based approach for improved compute efficiency and approximation accuracy.

Learning acquisition functions with α-PFN. After training the model q, we train a second PFN model, the α-PFN, that takes the observation data D and query point x as input and predicts acquisition $\alpha(x, D)$ directly, see Figure 2 for an illustration of the capabilities of this model. We train this model to predict the following information gain target

$$H(q(y|D, x)) - H(q(y|D, x, I)), \tag{5}$$

where I is the data conditioned on. If we are training an PES-PFN, we have $I = x^*$, and $I = f^*$ for MES-PFN and $I = (x^*, f^*)$ for JES-PFN. Note that we train separate α-PFN's for each ES variant. During training, we feed in the true x^* and/or f^* that were precomputed. This info is only used to determine the prediction target and is not required at test time. α-PFN will learn the distribution of this information gain (the varying factor being the location and/or value of the optimum). The mean of this distribution, which is the expected information gain, coincides with the PES/MES/JES acquisition in equations 2/3/4 (note that these are also expectations with respect to I). For more training details see Appendix D.

Handling domain shift under adaptive data collection. Müller et al. (2023) train PFNs for BO by sampling inputs x uniformly from the domain. However, during the process of BO, x tends to cluster around the local optima. This domain shift in the pretraining procedure appears to adversely affect BO performance of our approach in higher dimensions. To combat this, we train both PFNs with a sampling procedure that mimics the clustering behavior observed during the BO process (see Appendix F for more detail and an ablation study).

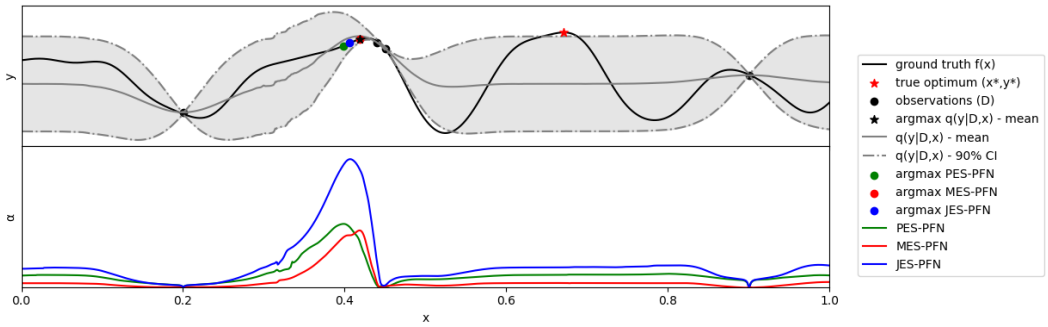

Figure 2: Here, we illustrate the acquisition functions as learned by the α-PFN's in 1D. While all three acquisitions may seem similar, they exhibit the expected differences around the inferred optimum (black star). PES (green) favors sampling around this optimum, as this provides more information about its location than sampling the optimum itself. MES (red) on the other hand is greedier, often sampling close to the optimum as to gain most info about its value. JES (blue) aims to find a middle ground between both approaches. It also worth noting that the joint expected gain of JES logically dominates the individual gains of PES and MES.

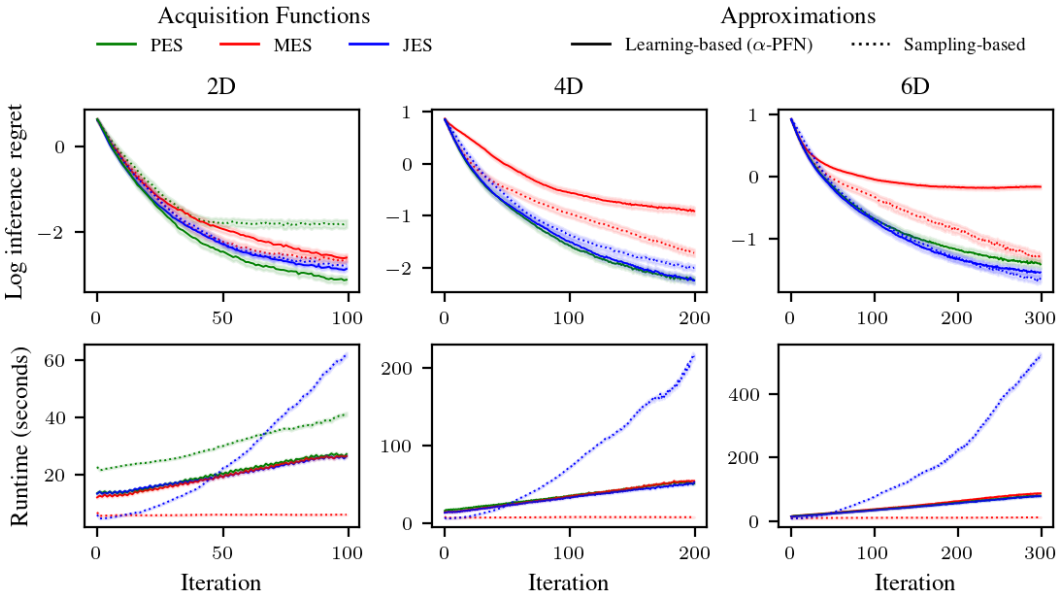

Figure 3: Average log inference regret (top) and runtime per iteration (bottom) for the different settings. Shaded region indicates standard error. Note that "Sampling-based" PES, MES and JES apply specific and different type of approximations that were derived by hand, while our PFN uses learned approximations. The JES-PFN closely matches the regret of the sampling-based JES implementation, but the MES-PFN performance is significantly worse. PES-PFN performs quite well. Sampling-based PES required overly much memory (4D required more than 48 GB of RAM), and therefore we did not include it for the 4D and 6D cases. As expected the PFN runtimes are approximately equal, and improve significantly upon the runtime of PES and JES, while sampling-based MES is clearly the fastest.

4 EXPERIMENTS AND RESULTS

Experimental setup. To validate our approach empirically, we assess it on Bayesian optimization of synthetic functions sampled from the GP prior. We replicate the first setup by Hvarfner et al. (2022) (JES) and consider their three settings (2D, 4D, and 6D) with fixed hyperparameters. See Table 1 (Appendix) for details on the settings and Appendix E for details on BO. We compare the performance of our α-PFN's with the existing Entropy Search approximations in the BoTorch library (Balandat et al., 2020): JES (Hvarfner et al., 2022), (GIBBON-)MES (Moss et al., 2021) and PES (Hernández-Lobato et al., 2014). We call these "sampling-based approximations", to contrast these with our α-PFN which uses learned approximations. We do not include sampling-based PES for 4D and 6D settings due its excessive memory requirement (>42 GB).

Evaluation metric. We evaluate all methods in terms of inference regret, which is the standard evaluation measure for methods using information theoretic acquisition functions. Inference regret is defined as $f(x^*) - f(\hat{x}^*)$, where $\hat{x}^*$ is the maximizer of the posterior predictive distribution, i.e., $\hat{x}^* = \text{argmax}_{x \in A} \mathbb{E}[q(y|D, x)]$. Note that this maximizer is approximated by performing gradient descent on $q(y|D, x)$, with a setup similar to that considered for the acquisition function optimization (see Appendix E).

Experimental results. Figure 3 reports the log inference regret and time taken per iteration, both averaged across 1000 GP function samples. It is useful to compare our results to those obtained by Hvarfner et al. (2022), since our experimental setting corresponds closely. The comparison shows that the regret obtained is in the correct range. The JES-PFN seems to closely match the performance of its sampling-based approximation. The performance of the MES-PFN is however orders of magnitude worse than its sampling-based approximation. Interestingly, the PES-PFN performs surprisingly well compared to JES, beating it in the 2D setting, but its advantage gets smaller with increasing dimensionality, and for the 6D setting it seems to perform slightly worse. This is surprising, since the PES approach of Hernández-Lobato et al. (2014) usually performs considerably worse than MES and JES, as shown also by Hvarfner et al. (2022) and Wang and Jegelka (2017). This indicates that the PFN may have learned a better approximation for computing the PES acquisition than the sampling-based approximation of Hernández-Lobato et al. (2014).

Runtime comparison. The PFN approximations follow a similar runtime complexity across the Entropy Search variants, as they primarily require only a forward pass of α-PFN to obtain the acquisition values. In the considered settings, GIBBON's MES approach of Moss et al. (2021) is faster than the MES-PFN variant. This advantage can be attributed to the more advanced approximations and optimized implementation, for example Moss et al. (2021) use efficient caching to speed up parallel acquisition computations. Note that similar optimizations could be explored for α-PFN and that, even without inference optimization, our PFN approximations are consistently faster than PES throughout the full BO run and surpass the state-of-the-art JES implementation after 50 iterations. Surprisingly, the runtime of JES in the 2D setting is similar to that of PES, while Hvarfner et al. (2022) reports JES to be considerably faster, which may be due to implementation details and warrants closer investigation in future work.

5 DISCUSSION AND CONCLUSION

Our initial results show the promise of our PFN approach to Entropy Search. We show it is capable of simulating the state-of-the-art (JES) at reduced runtimes. Also, the strong performance of the α-PFN for PES is encouraging, suggesting that the PFN can learn better approximations than handcrafted sampling-based state-of-the-art ones. This is even more noteworthy, as PES is the computationally most intractable, and our approach reduces overhead drastically. Our current MES results show an opposite pattern: while MES can be efficiently approximated by traditional methods, our PFN variant performs considerably worse, in terms of walltime and regret. A difficulty in evaluating this suboptimal performance of the MES-PFN is that the true acquisition function values for MES are not available. Perhaps, the approximations introduced by Wang and Jegelka (2017) and Moss et al. (2021) are crucial for its good empirical performance. On the other hand, it could be that α-PFN does not approximate MES well. Also, we observe a slight degradation in relative performance of

α-PFN in terms of regret in the 6D setting. It remains to be seen how our approach scales to higher dimensions and whether potential issues can be addressed by scaling up model training.

Another important question is the extent to which the results here generalize to real-world black-box optimization tasks. Previous works (Müller et al., 2023; Rakotoarison et al., 2024) have shown that PFNs are robust in out-of-distribution settings, but whether that observation generalizes to our α-PFNs remains to be validated. It is worth noting that the synthetic, fixed hyperparameter setting, considered, is ideal for our baselines both in terms of walltime and BO performance. While the PFNs naturally generalize to the fully Bayesian setting, without additional computational cost, the handcrafted approximations rely on sampling from the GP posterior, requiring MCMC in the fully Bayesian setting. As such, we expect that for fully Bayesian settings, the α-PFN will lead to more significant speedups, because learning can replace the MCMC samples (Müller et al., 2023).

Future research should include recent and faster approximation methods, such as parallel implementations for PES (Garrido-Merchán et al., 2023), and expand the experimentation setup to more challenging cases, i.e., real tasks and complex priors. Furthermore, given that the context points remain constant within a BO iteration, we plan to implement the caching mechanism (e.g., following the approach of Hollmann et al. (2025)) to further improve the inference speed of the α-PFN, which is critical to reduce the runtime for the acquisition function optimization. Finally, we would like to investigate the domain shift issue we observed and explore alternative solutions.

ACKNOWLEDGEMENTS

Tom Viering, Steven Adriaensen, Herilalaina Rakotoarison, and Frank Hutter acknowledge funding by TAILOR, a project funded by EU Horizon 2020 research and innovation programme under GA No 952215. This work wouldn't have happened without a research visit by Tom Viering made possible by the TAILOR Collaboration Exchange Fund. Frank Hutter acknowledges the financial support of the Hector Foundation. Steven Adriaensen, Herilalaina Rakotoarison, and Frank Hutter acknowledge funding by the state of Baden-Württemberg through bwHPC and the German Research Foundation (DFG), supporting grants INST 39/963-1 FUGG, 417962828, and INST 39/1232-1 FUGG (bwForCluster NEMO 2); and by the European Union (via ERC Consolidator Grant 'Deep Learning 2.0', grant no. 101045765). This research was funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under grant number 539134284, through EFRE (FEIH_2698644) and the state of Baden-Württemberg. Finally, the authors acknowledge Julien Siems for his help in the presentation of this work.

REFERENCES

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

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

Matthias Feurer and Frank Hutter. Hyperparameter optimization. *Automated machine learning: Methods, systems, challenges*, pages 3–33, 2019.

Julien Villemonteix, Emmanuel Vazquez, and Eric Walter. An informational approach to the global optimization of expensive-to-evaluate functions. *Journal of Global Optimization*, 44:509–534, 2009.

Philipp Hennig and Christian J Schuler. Entropy search for information-efficient global optimization. *Journal of Machine Learning Research*, 13(6), 2012.

Christopher KI Williams and Carl Edward Rasmussen. *Gaussian processes for machine learning*, volume 2. MIT press Cambridge, MA, 2006.

Jonas Močkus. On bayesian methods for seeking the extremum. In *Optimization techniques IFIP technical conference: Novosibirsk, July 1–7, 1974*, pages 400–404. Springer, 1975.

Niranjan Srinivas, Andreas Krause, Sham Kakade, and Matthias Seeger. Gaussian process optimization in the bandit setting: no regret and experimental design. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, page 1015–1022, 2010.

José Miguel Hernández-Lobato, Matthew W Hoffman, and Zoubin Ghahramani. Predictive entropy search for efficient global optimization of black-box functions. *Advances in neural information processing systems*, 27, 2014.

Zi Wang and Stefanie Jegelka. Max-value entropy search for efficient bayesian optimization. In *International Conference on Machine Learning*, pages 3627–3635. PMLR, 2017.

Carl Hvarfner, Frank Hutter, and Luigi Nardi. Joint entropy search for maximally-informed bayesian optimization. *Advances in Neural Information Processing Systems*, 35:11494–11506, 2022.

Ben Tu, Axel Gandy, Nikolas Kantas, and Behrang Shafei. Joint entropy search for multi-objective bayesian optimization. *Advances in Neural Information Processing Systems*, 35:9922–9938, 2022.

Samuel Müller, Noah Hollmann, Sebastian Pineda Arango, Josif Grabocka, and Frank Hutter. Transformers can do bayesian-inference by meta-learning on prior-data. In *Fifth Workshop on Meta-Learning at the Conference on Neural Information Processing Systems*, 2021.

Marta Garnelo, Dan Rosenbaum, Christopher Maddison, Tiago Ramalho, David Saxton, Murray Shanahan, Yee Whye Teh, Danilo Rezende, and SM Ali Eslami. Conditional neural processes. In *International conference on machine learning*, pages 1704–1713. PMLR, 2018.

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

Christian P Robert, George Casella, and George Casella. *Monte Carlo statistical methods*, volume 2. Springer, 1999.

Thomas Peter Minka. *A family of algorithms for approximate Bayesian inference*. PhD thesis, Massachusetts Institute of Technology, 2001.

Ali Rahimi and Benjamin Recht. Random features for large-scale kernel machines. *Advances in neural information processing systems*, 20, 2007.

Shion Takeno, Hitoshi Fukuoka, Yuhki Tsukada, Toshiyuki Koyama, Motoki Shiga, Ichiro Takeuchi, and Masayuki Karasuyama. Multi-fidelity Bayesian optimization with max-value entropy search and its parallelization. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 9334–9345. PMLR, 13–18 Jul 2020.

Quoc Phong Nguyen, Bryan Kian Hsiang Low, and Patrick Jaillet. Rectified max-value entropy search for bayesian optimization, 2022.

Henry B Moss, David S Leslie, Javier Gonzalez, and Paul Rayson. Gibbon: General-purpose information-based bayesian optimisation. *Journal of Machine Learning Research*, 22(235):1–49, 2021.

Shion Takeno, Tomoyuki Tamura, Kazuki Shitara, and Masayuki Karasuyama. Sequential and parallel constrained max-value entropy search via information lower bound. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato, editors, *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 20960–20986. PMLR, 17–23 Jul 2022.

Carl Hvarfner, Erik Orm Hellsten, Frank Hutter, and Luigi Nardi. Self-correcting bayesian optimization through bayesian active learning. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

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

Herilalaina Rakotoarison, Steven Adriaensen, Neeratyoy Mallik, Samir Garibov, Edward Bergman, and Frank Hutter. In-context freeze-thaw bayesian optimization for hyperparameter optimization. *arXiv preprint arXiv:2404.16795*, 2024.

Samuel Dooley, Gurnoor Singh Khurana, Chirag Mohapatra, Siddartha V Naidu, and Colin White. Forecastpfn: Synthetically-trained zero-shot forecasting. *Advances in Neural Information Processing Systems*, 36, 2024.

Steven Adriaensen, Herilalaina Rakotoarison, Samuel Müller, and Frank Hutter. Efficient bayesian learning curve extrapolation using prior-data fitted networks. *Advances in Neural Information Processing Systems*, 36, 2024.

Tom Julian Viering, Steven Adriaensen, Herilalaina Rakotoarison, and Frank Hutter. From epoch to sample size: Developing new data-driven priors for learning curve prior-fitted networks. In *AutoML Conference 2024 (Workshop Track)*, 2024.

Noah Hollmann, Samuel Müller, Lennart Purucker, Arjun Krishnakumar, Max Körfer, Shi Bin Hoo, Robin Tibor Schirrmeister, and Frank Hutter. Accurate predictions on small data with a tabular foundation model. *Nature*, 637(8045):319–326, 2025.

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

Eduardo C Garrido-Merchán, Daniel Fernández-Sánchez, and Daniel Hernández-Lobato. Parallel predictive entropy search for multi-objective bayesian optimization with constraints applied to the tuning of machine learning algorithms. *Expert Systems with Applications*, 215:119328, 2023.

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

Louis C Tiao, Aaron Klein, Matthias W Seeger, Edwin V. Bonilla, Cedric Archambeau, and Fabio Ramos. BORE: Bayesian optimization by density-ratio estimation. In *International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*. PMLR, 18–24 Jul 2021.

Jiaming Song, Lantao Yu, Willie Neiswanger, and Stefano Ermon. A general recipe for likelihood-free bayesian optimization. In *International Conference on Machine Learning*, pages 20384–20404. PMLR, 2022.

Michael Volpp, Lukas P. Fröhlich, Kirsten Fischer, Andreas Doerr, Stefan Falkner, Frank Hutter, and Christian Daniel. Meta-learning acquisition functions for transfer learning in bayesian optimization. In *International Conference on Learning Representations*, 2020.

Alexandre Max Maraval, Matthieu Zimmer, Antoine Grosnit, and Haitham Bou Ammar. End-to-end meta-bayesian optimisation with transformer neural processes. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

Paul E. Chang, Nasrulloh Loka, Daolang Huang, Ulpu Remes, Samuel Kaski, and Luigi Acerbi. Amortized probabilistic conditioning for optimization, simulation and inference. *CoRR*, abs/2410.15320, 2024.

Setting nr.	d	σ_f	σ_l	σ_n	μ
1	2	$\sqrt{10}$	0.1	0.1	0
2	4	$\sqrt{10}$	0.2	0.1	0
3	6	$\sqrt{10}$	0.3	0.1	0

Table 1: The different scenarios for which we evaluate α-PFN. d indicates the dimension of the input space. We use the squared exponential kernel, where σ_f^2 is the variance of the Gaussian Process (output scale) and σ_l is the length scale of the kernel. We add zero mean Gaussian noise $N(0, \sigma_n^2)$ to the GP, and we set the mean μ of the GP to zero.

A Further Related Work

In Section 2, we discussed the key concepts our approach builds upon (BO, ES, and PFNs). However, other works have explored related ideas and in this section we briefly discuss these. In particular, other works have proposed the use of transformers to learn to emulate optimization procedures. Chen et al. (2022) proposed the use of transformers to predict points to be selected for Bayesian optimization, based off empirical traces of Bayesian optimization runs utilizing handcrafted acquisition functions such as EI. Tiao et al. (2021) and later works by Song et al. (2022) propose the use of binary classification models to directly predict acquisition functions such as probability of improvement or expected improvement. Following a similar idea, end-to-end approaches Volpp et al. (2020); Maraval et al. (2023) have been also proposed to simultaneously learn both the acquisition function and the surrogate model from scratch using reinforcement learning algorithms. Concurrent with our work, Chang et al. (2024) proposes a similar approach to learning conditional probabilities, differing mainly in that their method outputs the parameters of a Gaussian mixture to model continuous output values. A comparison to their method is left for future work.

B Pre-computing Approximate GP Data and their Maxima

B.1 GP Approximation

For each setting, we generate 1M approximate samples from the corresponding GP prior (see Table 1) using the Random Fourier Feature approximation (RFF; Rahimi and Recht, 2007), 999K were used for training the PFNs and 1000 in our validation experiment in Section 4. We use 5000 RFFs when computing the GP approximations. We store the Random Fourier Features and the vector w that represents the Gaussian Process; using these, we can sample from a single GP any arbitrary number of points independently and quickly. Computing the maximum of a GP that uses a Random Fourier Features approximation is not straightforward, because the GP may have many local maxima. To that end, we perform a quite extensive search with random restarts with an ensemble of optimizers. First, we describe the optimizer and its hyperparameters, and afterward we describe the ensemble construction.

B.2 GP Maximizer

We use either SGD or Adam and we optimize a batch of size `num_samples` points over the GP in parallel. The initialization is done either uniformly at random over the domain $[0, 1]^d$ (`resample_init` is False) or by trying a large number of points (`num_repeats` times `num_samples` points are tried), computing their function values, and keeping the best `num_samples` (if `resample_init` is True). We always use the noiseless GP values. We optimize for `n_iterations_max` iterations using SGD or Adam. During the optimization, we monitor the current best seen GP value so far and store it. If the current best point does not improve (compared with a tolerance `tol`), we increase a counter indicating the patience, and otherwise the patience counter is reset. After the patience counter reaches a value of `patience`, we decay the learning rate by a factor of `decay_factor`. If the learning rate is decayed more than `max_decays` times, we stop the optimization early. One should take care with points that move outside the domain during optimization, as optima are often located at the edge, especially for higher length scales / dimensions. If `clamp` is active, we always move the points back inside the domain

Table 2: Hyperparameter grid values for building the GP Maximizer Ensemble.

hyperparameter	grid values	hyperparameter	grid values
adam	[True, False]	max_decays	[1, 5, 10, 15]
init_lr	[0.001, 0.01, 0.1, 1]	tol	[1e-1, 1e-3, 1e-6]
resample_init	[True, False]	decay_factor	[0.1, 0.5, 0.99, 1]
num_samples	[50,200,1000]	clamp	[True, False]
n_iterations_max	[10, 100, 1000]	patience	[1, 5, 10, 20]
num_repeats	[10, 100, 1000]		

by clamping. If `clamp` is not true, we randomly initialize these points. If not specified, we use the default values for the optimizers as specified in Pytorch version 2.3.1.

B.3 Maximizer Ensemble Construction

We use the first 1000 RFF GPs to build the optimizer ensemble. We build an candidate set of a 1000 optimizers with hyperparameters sampled from the grid as defined in Table 2. Because we want to determine the maxima for a large number of RFF GPs, we need to design a small ensemble with a low runtime yet good performance. We measure performance in terms of the regret of the ensemble, that is, the performance of the ensemble compared to the performance of the best optimizer in the candidate set. To reliably estimate the ensemble performance, we use 5-fold cross validation, where a train fold is used for the ensemble building, and the test fold is used for ensemble evaluation. The ensemble is constructed greedily by forward selection, where only candidates are considered that have an average runtime of less than 10 seconds per GP optimization. We keep adding ensemble members until the regret indicates we find the optimum with an approximate precision of 1e-6. We merge the ensembles together over the different training folds to come to a final ensemble. Note that the ensemble is build separately for each prior of Table 1.

C Base PFN Training Details

We trained three base PFN models, one per setting considered. We closely follow the original PFN architecture and training pipeline, used in previous works (Müller et al., 2021; 2023). We use a small ($<$15M parameter) decoder-only transformer, with 12 layers, each using an embedding size of 512, 4 attention heads, and 1024 units in the hidden expansion layer. We use the PFN regression head proposed by Müller et al. (2021) to model the output distribution, treating Bayesian regression as a classification problem, dividing the output range in discrete ranges (bins) and minimize the cross-entropy loss. We apply a linear transformation $\text{norm}(y) = \frac{y}{16\sqrt{10}} + 0.5$ which effectively projects our GP output range in [0,1], which we divide in 1000 equal-sized bins. Note that, with very high likelihood, e.g., all 3M max-values in our precomputed data fall in this range after normalization. At test time, we clip hypothetical values outside this range. We minimize the cross-entropy loss, using AdamW with a batch size of 100 datasets, a cosine decay learning rate schedule with maximum 0.0001, and linear warmup over the first 25% iterations of the training run. The models for higher dimensions, were trained on fewer, but larger datasets. More specifically, the base models for 2D, 4D and 6D, were trained on 60M, 30M, and 20M datasets of size $75d$ respectively. The GP sample used for a dataset is selected randomly from the 999K pregenerated samples, and reused multiple times. To counteract overfitting, and to encourage symmetry, we perform mirror and reflection augmentations on the domain A. The context size C (train split) is fixed per batch and selected $C \sim U(0, 50 \cdot D)$. This choice keeps the expected total number of queries (points in test split) invariant (6 Billion) and wall-clock training time under 1 GPU day on a single L40S GPU. The context and query points are chosen as described in Appendix F. To achieve the conditioning, we add a special conditioning ($x^*?$,$f^*?$) token in the context. During training, we randomly sample the conditioning to train the base model: (1) no conditioning, (2) condition on x^*, (3) condition on f^*, (4) condition on both. If optimum location or value (or both) are not given, a placeholder vector/value of 0.5 is used. This conditioning token is encoded as any other, except for adding one of four independently learned embeddings to its encoding to allow the PFN to discriminate between the four cases.

D α-PFN TRAINING DETAILS

The PFN model directly predicting the (expected) information gain closely follows the architecture and training of the base model. The only differences are that we trained all 9 models (one for each setting and ES variant) on 10M datasets, no conditioning tokens were used, and that the prediction targets for queries are not $f(x)$, but the oracle information gain from equation 5. We again use 1000 bins, but these are not equal-sized, and rather determined such that roughly equally many of the training targets fall in each bin. Furthermore, we use the full-support output head (Müller et al., 2021), where the outmost "bins" are modeled as half-normals.

E BAYESIAN OPTIMIZATION EXPERIMENT DETAILS

For all settings considered, we run Bayesian Optimization with $50d$ iterations and d uniform points as the initial design. At each BO iteration, the acquisition function is optimized by sampling 1024 uniform points and then optimizing them with gradient-based optimization with 20 restarts. Since the acquisition for PES in BoTorch is not be differentiable, we use finite differences for optimizing PES instead. We follow a similar optimization procedure to compute the maximizer of predictive posterior distribution, required for computing the inference regret. Each BO run was conducted on a single CPU (AMD EPYC 7763 64-Core Processor) with a 12GB memory budget, except for sampling-based PES in the 2D case, which required 48GB.

F SYNTHETIC OPTIMIZATION TRACE GENERATION

F.1 MOTIVATION

Previous work (Müller et al., 2021; 2023; Rakotoarison et al., 2024) considered context and query points sampled uniformly. However, in real Bayesian Optimization (BO) traces, the context points follow a structured search pattern, dynamically balancing global and local search and often forming clusters around local optima. Likewise, uniform query points, in high dimensions, are unlikely to be near any of the context points, limiting the opportunity to learn to exploit the information they provide. Ideally, we would use actual optimization traces from the Entropy Search BO procedures. However, this would introduce a dependency on our surrogate model, leading to a chicken-and-egg problem. Alternatively, using BoTorch traces would restrict ourselves to the GP priors it supports. Instead, we propose a simple and efficient synthetic procedure that generates context and query points in a manner that mimics real BO traces.

F.2 PROCEDURE

Our synthetic optimization trace generation procedure, detailed in Algorithm 1, aims to replicate the characteristics of real BO traces by blending global and local search. The key components are:

- **Global search**: Points are sampled uniformly at random within the search space, with an additional probability ϵ of selecting a point exactly on the edge. This helps in exploring boundary effects which are important in higher dimensions as the optimizer increasingly often lies exactly on the edge.
- **Local search**: The next context points are drawn from a Gaussian distribution centered on the best observed context point so far. For query points, we select an arbitrary context point as the center. If the optimizer x^* is provided, it is sometimes chosen as the center, which facilitates learning the effect of conditioning on x^*.
- **Dynamic search adaptation**: BO dynamically transitions from global to local search over time. To model this, we define a local search probability α_i that linearly decreases over L steps. This ensures that earlier points explore the space globally, while later points refine the search locally.
- **Avoiding duplicate points**: Though not explicitly shown in the pseudocode, we ensure that no duplicate points occur in the trace. This frequently happens in corner regions. If a newly generated point coincides with an existing corner point, it is resampled.

This procedure effectively balances exploration and exploitation, producing synthetic traces that resemble real BO optimization trajectories while remaining computationally efficient.

F.3 Ablation Experiment

To evaluate the importance of our synthetic trace procedure, we conduct an ablation study comparing it to uniform sampling. Our results demonstrate that this structured trace generation significantly outperforms uniform sampling, especially in higher dimensions, achieving performance levels comparable to that obtained using sampling-based approximations. Interestingly, the procedure is also essential for EI, which only uses the unconditional base PFN model in a setup that closely resembles the one by Müller et al. (2023). One explanation could be the fact that our base model is trained on less data, considers larger context sizes / four conditioning cases, and greater resilience to domain-shift may be attained with additional training. Future work should investigate this discrepancy, further refine the procedure, perform a more fine-grained ablation, and explore alternatives for synthetic trace generation.

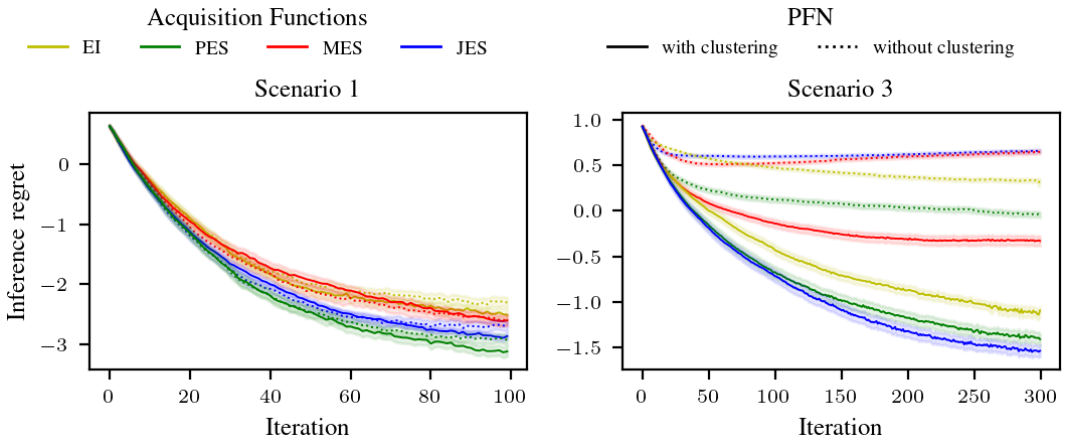

Figure 4: Ablating the BO traces used in training the α-PFN.

Algorithm 1 Generate Optimization Trace

Inputs:
  L: length of the trace
  C: number of context points (i.e., we have $L - C$ query points)
  d: dimension of search space
  GP_sample: function to evaluate GP values
  x^*: The optimizer of the GP (False in the unconditional / α-PFN case)
Outputs:
  $trace$: matrix of shape (L, d) with context/query points in the trace
  y: vector of length C with function values for context points
Procedure:
Initialize $trace \leftarrow$ zero matrix of size (L, d)
Initialize $y \leftarrow$ zero vector of size C
$\epsilon = (1 - u^{\frac{d}{6}})$ with $u \sim U(0, 1)$ ▷ Sample edge probability
$\sigma \sim$ LogNormal$(-3, 0.5)$ ▷ Sample local search step size
Sample initial / final local search probability:
  $\alpha_0 = \min(v_1, v_2, v_3)$
  $\alpha_L = \max(v_1, v_2, v_3)$
  with $v_1, v_2, v_3 \sim U(0, 1)$
Start trace from a random point in search space:
  $best\_point = trace[0] \leftarrow clip(w, 0, 1)$ with $w \sim U^d(-\frac{\epsilon}{2}, 1 + \frac{\epsilon}{2})$
  $y_{best} = y[0] \leftarrow GP\_sample(trace[0])$
for $i = 1$ to $L - 1$ **do**
  $\alpha_i \leftarrow \alpha_0 + (\alpha_L - \alpha_0) \cdot (i/L)$ ▷ Determine local search probability
  $local \leftarrow$ Bernoulli(α) ▷ Determine local or global search
  if $local$ **then**
    if $i < C$ **then**
      $inc \leftarrow best\_point$ ▷ Sample near the best point thus far
    else
      $z \sim U(0, 1)$
      if $x^* \wedge z < \frac{5d}{(L - cutoff)}$ **then**
        $inc \leftarrow x^*$ ▷ Sample near the true optimizer
      else
        Choose inc randomly from context points ($trace[: C]$)
      end if
    end if
    $trace[i] \leftarrow clip(x, 0, 1)$ with $x \sim \mathcal{N}^d(inc, \sigma^2)$
  else
    $trace[i] \leftarrow clip(x, 0, 1)$ with $x \sim U^d(-\frac{\epsilon}{2}, 1 + \frac{\epsilon}{2})$ ▷ Global search
  end if
  if $i < C$ **then** ▷ For context point, sample value and update best
    $y[i] \leftarrow GP\_sample(trace[i])$
    if $y[i] > y_{best}$ **then**
      $best\_point \leftarrow trace[i]$
      $y_{best} \leftarrow y[i]$
    end if
  end if
end for
return $trace, y$

G Comparison with Expected Improvement (EI)

In Figure 5 we also include a comparison with Expected Improvement (EI). We evaluate EI both when used in closed form with the Gaussian Process (normal line) and when derived from the base PFN model (following the approach of Müller et al. (2023)). Interestingly, EI-PFN is faster than α-PFN, possibly because optimizing the EI acquisition function is inherently easier and requires fewer iterations compared to the ES acquisition functions. The regret obtained by EI for both the PFN and closed form case match well, which validates that our base PFN is well-trained.

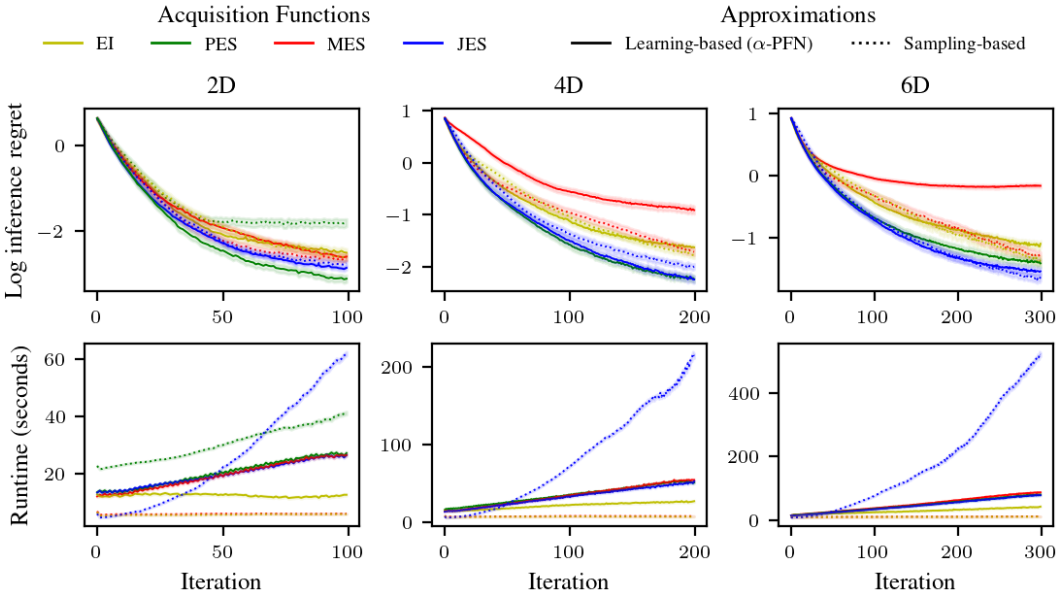

Figure 5: Average log inference regret (top) and runtime per iteration (bottom) for the different settings. Shaded region indicates standard error. In contrast with the main body, here we also include Expected Improvement (EI) to further place our results in context.