Posterior Sampling using Prior-Data Fitted Networks for Optimizing Complex AutoML Pipelines

Amir Rezaei Balef
Department of Computer Science
University of Tübingen
amir.balef@inf.uni-tuebingen.de

Katharina Eggensperger
Department of Computer Science
University of Tübingen
katharina.eggensperger@uni-tuebingen.de

Abstract

Combined Algorithm Selection and Hyperparameter Optimization (CASH) has been fundamental to traditional AutoML systems. However, with the advancements of pre-trained models, modern ML workflows go beyond hyperparameter optimization and often require fine-tuning, ensembling, and other adaptation techniques. While the core challenge of identifying the best-performing model for a downstream task remains, the increasing heterogeneity of ML pipelines demands novel AutoML approaches. This work extends the CASH framework to select and adapt modern ML pipelines. We propose PS-PFN to efficiently explore and exploit adapting ML pipelines by extending Posterior Sampling (PS) to the max k-armed bandit problem setup. PS-PFN leverages prior-data fitted networks (PFNs) to efficiently estimate the posterior distribution of the maximal value via in-context learning. We show how to extend this method to consider varying costs of pulling arms and to use different PFNs to model reward distributions individually per arm. Experimental results on one novel and two existing standard benchmark tasks demonstrate the superior performance of PS-PFN compared to other bandit and AutoML strategies. We make our code and data available at `https://github.com/amirbalef/CASHPlus`.

1 Introduction

What model will perform best? The best-performing answer to this crucial question depends on the given task. We study this question in the context of predictive ML for tabular tasks, where possible model classes range from pre-trained models [Zhu et al., 2023], tree-based methods [Chen and Guestrin, 2016, Prokhorenkova et al., 2018], modern deep learning approaches [Holzmüller et al., 2024, Gorishniy et al., 2021], in-context learning with foundation models [Hollmann et al., 2025, Cui et al., 2024], and many more. Despite significantly different modelling approaches, with different implicit biases and strengths and weaknesses, all models have one thing in common: They must be adapted to the task at hand to perform best. This adaptation is typically an iterative procedure like optimizing hyperparameters, fine-tuning weights, learning embeddings, adapting the architecture, or running an AutoML system (see Figure 1).

A popular task for automatically allocating resources across workflows that include hyperparameter optimization (HPO) is the *Combined Algorithm and Hyperparameter Optimization (CASH)* [Thornton et al., 2013] problem. A common approach to the CASH problem is to model it as a single hierarchical HPO task over a joint search space that includes the choice among K ML algorithms and their associated hyperparameters. This is typically done by treating the algorithm selection as a categorical hyperparameter and using conditional dependencies to define the model-specific subspaces. State-of-the-art approaches either tackle this as a single-level optimization problem via Bayesian optimization on the combined hierarchical search space (we refer to this as *combined CASH*), as done by AutoML

18th European Workshop on Reinforcement Learning (EWRL 2025).

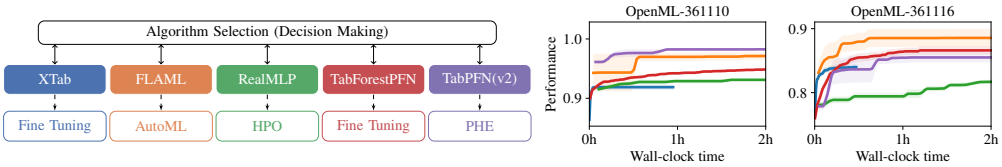


Figure 1: (Left) AutoML needs to perform algorithm selection and resource allocation across heterogeneous optimization tasks, such as hyperparameter tuning, fine-tuning, ensembling, and more. (Right) The performance of each workflow on two datasets demonstrates the variability of the optimization trajectories and the importance of algorithm selection .

Systems [Thornton et al., 2013, Feurer et al., 2015, Komer et al., 2014, Kotthoff et al., 2017, Feurer et al., 2022]. However, this approach requires optimizing over a large, high-dimensional, and heterogeneous space using a single HPO method, which can be inefficient and difficult to scale. Alternatively, one can decompose it into a two-level optimization problem and leverage bandit methods at the top level and HPO methods at the lower level [Li et al., 2020, Hu et al., 2021, Balef et al., 2024, 2025] (we refer to this as *decomposed CASH*). This framework has also been recently adapted to other types of workflows, e.g., selecting and fine-tuning pre-trained models [Arango et al., 2024, van den Nieuwenhuijzen et al., 2024].

Here, we want to optimally allocate resources across different, heterogeneous workflows, with different performance distributions, optimization behaviour, and cost per iteration (see Figure 1). Existing *combined and decomposed CASH* methods are not straightforward to extend since they were designed to perform well for a single type of workflow.

We address the algorithm selection task by resource allocation for complex AutoML pipelines using a bandit algorithm that leverages a flexible pre-trained machine learning model performing in-context learning to predict the posterior distribution. Our contributions are as follows:

1. We introduce CASH+ as a generalization of CASH, extending to any performance optimization strategy, covering modern ML workflows.
2. We address this as a two-level optimization problem and at the top level as a Max-K-armed Bandit (MKB) problem. We extend Posterior Sampling to optimize the maximal reward.
3. We identify prior-data fitted networks (PFNs) [Müller et al., 2022] as a flexible model class to enable posterior sampling in practice and analyze HPO trajectories and learning curves to design effective priors. We call the resulting method PS-PFN.
4. We extend PS-PFN to handle varying costs and to use different PFNs per arm.
5. Finally, we demonstrate superior empirical performance of PS-PFN on three AutoML benchmark tasks where we compare PS-PFN with state-of-the-art MAB algorithms for CASH.

We start by discussing our problem setup and introducing CASH+ in Section 2. Section 3 provides a short introduction to Posterior Sampling. Next, we introduce PS-PFN in Section 4, and evaluate it in Section 5. We conclude by discussing its limitations and future work in Section 6.

2 Combined Algorithm Selection and Optimization (CASH+)

The CASH problem, at the core of traditional AutoML pipelines [Thornton et al., 2013], involves selecting the optimal ML algorithm $A^{(i^*)}$ and its hyperparameters $\boldsymbol{\lambda}^*$ to minimize a loss metric $\mathcal{L}$ (e.g., validation error). Given a dataset $\mathbb{D} = \{D_{train}, D_{valid}\}$ and a set of K candidate algorithms $\mathcal{A} = \{A^{(1)}, ..., A^{(K)}\}$, each with its own hyperparameter space $\boldsymbol{\Lambda}^{(i)}$, the goal of CASH is to optimize over the joint algorithm-hyperparameter space. Here, we generalize this to any iterative optimization procedure where the goal is to select the optimal ML algorithm $A^{(i^*)}$ and its state $\boldsymbol{s}^*$ as found by its optimization procedure. Formally,

$$A^{(i^*)}_{\boldsymbol{s}^*} \in \underset{A^{(i)} \in \mathcal{A}, \boldsymbol{s} \in \boldsymbol{S}^{(i)}}{\arg\min} \mathcal{L}(A^{(i)}_{\boldsymbol{s}}, \mathbb{D}). \tag{1}$$

By going beyond standard model selection and hyperparameter tuning, CASH+ brings a broader range of optimization strategies to the AutoML pipeline. We note that this is very similar to the original definition of CASH as introduced by Thornton et al. [2013], but explicitly allows for including *any* optimization method, thus covering methods like hyperparameter tuning, fine-tuning, and ensembling. These methods often involve highly diverse search spaces. For example, fine-tuning neural networks typically has a large and complex space, while Post-Hoc Ensembling (PHE) may not have a well-defined search space at all. In such cases, *combined CASH* approaches are no longer applicable due to the incompatibility of the search spaces and, thus, search methods. Instead, similar to prior work [Li et al., 2020, Hu et al., 2021, Balef et al., 2024, 2025], we tackle CASH+ as a two-level optimization problem (as depicted in Figure 1), which we formally define as:

$$A^{(i^*)} \in \underset{A^{(i)} \in \mathcal{A}}{\arg\min}\, \mathcal{L}(A^{(i)}_{\boldsymbol{s}^*}, \mathbb{D}) \quad \text{s.t. } \boldsymbol{s}^* \in \underset{\boldsymbol{s} \in \boldsymbol{S^{(i)}}}{\arg\min}\, \mathcal{L}(A^{(i)}_{\boldsymbol{s}}, \mathbb{D}). \tag{2}$$

At the upper level, we aim to find the overall best-performing ML model $A^{(i^*)}$ by selecting model $A^{(i)} \in \mathcal{A}$ iteratively and at the lower level, we optimize the selected model $A^{(i)}$.

CASH+ as a Bandit problem. To approach the upper-level decision-making task (see Figure 1) with a bandit method, we denote $r_{i,t}$ to be the feedback to arm i obtained by evaluating $\boldsymbol{s}_t \in \boldsymbol{S}^{(i)}$ at time step $t \leq T$. To be consistent with the bandit literature, we consider the negative loss as the reward and maximize it:

$$r_{i,t} = -\mathcal{L}(A^{(i)}_{\boldsymbol{s}_t}, \mathbb{D}). \tag{3}$$

The goal is then to find the best-performing algorithm $A^{(i^*)}$ and its optimal state $\boldsymbol{s}^*$ given a time horizon T. This can be framed as minimizing the regret $R(T)$ with I_t being the selected arm at time t by:

$$R(T) = \max_{k \leq K} \mathbb{E}[\max_{t \leq T} r_{k,t}] - \mathbb{E}[\max_{t \leq T} r_{I_t,t}]\,. \tag{4}$$

This regret describes the gap between the highest possible oracle reward that could be obtained by pulling only the arm with the highest performance (left part) and the actual observed rewards obtained by applying our strategy (right part).

3 Background

The two-level optimization problem allows each model to be optimized independently. Existing bandit algorithms used in the *decomposed CASH* literature generally assume that all arms (i.e., optimization methods) share similar characteristics [Li et al., 2020, Hu et al., 2021, Balef et al., 2024, 2025]. However, in the CASH+ setting, the arms can differ significantly in their reward processes and costs, making those assumptions invalid. For this, we explore Posterior Sampling as a much more flexible framework for developing our algorithm.

3.1 Posterior Sampling

Posterior sampling, first introduced in 1933 [Thompson, 1933], has become a fundamental technique in the field of Multi-Armed Bandits (MAB) known as Thompson Sampling (TS). This approach maintains a posterior over reward models and samples from it at each decision point to select actions. Posterior sampling is highly flexible, as it does not require closed-form solutions or restrictive assumptions. It can be applied with any probabilistic model, provided that sampling from the (possibly approximate) posterior is feasible [Kveton et al., 2024].

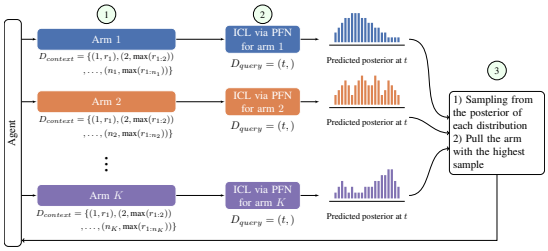
In Algorithm 1 at each round t, the agent maintains a dataset D_i of all observed rewards from arm i and models the posterior distribution accordingly. It then samples $s_i \sim p(\cdot \mid D_i, f(t))$ from the posterior distribution. The function $f(t)$ can account for potential temporal dependencies in the reward process, such as rested or non-stationary rewards [Liu et al., 2023], which can be viewed as using time information as context for modeling reward processes in contextual bandits [Shen et al., 2023]. After pulling arm I_t, the agent observes the reward $r_{I_t,n_{I_t}+1}$ and updates the dataset D_{I_t} with the new observation.

Posterior sampling relies fundamentally on the agent's ability to reason about the posterior distribution over rewards. This leads to optimal performance in maximizing the expected *cumulative* reward in classical bandit settings [Russo and Van Roy, 2014]. In contrast, the goal in our setting is to optimize the expected *maximum* observed reward (see Equation 4), which aligns with the MKB framework. This shift in objective requires reasoning about the posterior distribution of maximum values across arms, introducing new challenges. In this paper, we develop approximation methods designed to estimate this distribution effectively.

Algorithm 1 Posterior Sampling (Thompson Sampling)

Require: time horizon T, time function $f(t)$, number of arms K
for each arm $i \in \{1, \dots, K\}$ **do**
  Pull arm i once; observe reward $r_{i,1}$
  Initialize dataset $D_i \leftarrow \{(1, r_{i,1})\}$ and counter $n_i \leftarrow 1$
end for
for $t = K + 1$ to T **do**
  for each arm $i \in \{1, \dots, K\}$ **do**
    Draw $s_i \sim p(\cdot \mid D_i, f(t))$ {Sample from posterior}
  end for
  Select arm $I_t \leftarrow \arg\max_{i \in \{1,\dots,K\}} s_i$
  Pull arm I_t; observe reward $r_{I_t, n_{I_t}+1}$
  $n_{I_t} \leftarrow n_{I_t} + 1$
  Update $D_{I_t} \leftarrow D_{I_t} \cup \{(n_{I_t}, r_{I_t, n_{I_t}})\}$
end for

Figure 2: A single iteration of PS-PFN. (1) Context construction: Format observed rewards as PFN input; (2) Posterior prediction: Query PFN for time step t; (3) Decision: Sample from posterior to select arm.



3.2 Model-based Posterior Approximation

Computing the exact posterior distribution is often analytically or computationally infeasible for complex reward models. Consequently, approximate inference methods such as Markov Chain Monte Carlo or Variational Inference are commonly employed to estimate posterior distributions in practice [Phan et al., 2019]. Recently, transformers have demonstrated the capability to perform in-context learning (ICL) when trained on large amounts of data [Brown et al., 2020]. ICL methods are very sample efficient, yielding high accuracy with only a few observations, making them very suitable for tasks with short time horizons.

So-called Decision Transformers (DTs) reframe reinforcement learning (RL) as a sequence modeling task, by conditioning on desired returns, past states, and actions [Chen et al., 2021]. DTs predict future actions and leverage ICL to generalize with minimal adaptation. Lin et al. [2024] studies the ICL capability of pretrained transformers for RL in theory. Recent work shows that Decision Pretrained Transformers can approximate Bayesian posterior sampling over actions, enabling principled exploration to solve bandit problems [Lee et al., 2023]. While prior work focuses on predicting optimal actions, our approach instead models the posterior distribution over rewards to achieve better scalability, for example, concerning the number of actions.

4 Methodology

Accurately estimating the posterior distribution of the maximum reward is crucial for effective decision-making in the MKB settings. We first show how to extend PS to the MKB setting by finding the posterior distribution of maximum values under the simplified assumption of i.i.d. rewards and sufficiently fast concentration around the maximum. Since this assumption can be limiting for addressing CASH+, we then discuss ICL using prior-fitted networks as a more flexible and expressive approximation of the posterior.

4.1 PS-Max: Posterior Sampling for Max K-armed Bandit

We need to accurately predict the posterior distribution of the maximum reward to extend classical Thompson Sampling to our setting. Based on Algorithm 1 and the regret definition (Equation 4), we introduce PS-Max for K arms with a limited time horizon of T. Let $r_{i,1:t}$ denote sequence of rewards from time 1 till time t.

Assumption 1. *We assume that the i.i.d. random variable $r_{i,t}$, representing the reward of pulling arm i at time t, follows a sub-Gaussian distribution with cumulative distribution function (CDF) $F(x)$.*

When Assumption 1 is held, line 7 in Algorithm 1 can be replaced by:

$$s_i = F_i^{-1}(U^{1/t}), \quad U \sim \text{Uniform}(0, 1) \tag{5}$$

where $f(t) = t$ and F_i^{-1} is the inverse CDF of our sub-Gaussian prior for arm i. We call the resulting method **PS-Max**.

Theorem 1 (Analysis of PS-Max)**.** *If the expected maximum satisfies:*

$$\mathbb{E}[\max(r_{i,1:t})] \geq F_i^{-1}\left(1 - \frac{1}{t^2}\right),$$

then the number of times sub-optimal arm i is pulled grows logarithmically with time.

PS-Max operates similarly to classical Thompson Sampling. At each iteration, it updates the sub-Gaussian prior based on observed rewards and then draws samples from the posterior of the maxima (Equation 5) rather than from the posterior of the sub-Gaussian. For now, we choose $f(t) = t$ and assume constant costs to ensure anytime performance, as this choice aligns with the objective of identifying the arm with the highest expected maximum at each time step t. [1]

We provide the proof of Theorem 1 in Appendix A.1. MAB algorithms can surely achieve optimal performance *only* when the reward distribution aligns with their assumptions. The condition in Theorem 1 highlights that the distribution must have a very light tail near its maximum. In CASH problems, because of the left-skewed reward distribution and almost non-stationarity, the conditions of Assumption 1 and Theorem 1 are often satisfied [Balef et al., 2025].

However, for CASH+ tasks, the heterogeneity of the lower-level optimization methods results in rewards drawn from different distributions. This brings up three main limitations to overcome:

- **The reward distributions do not follow a common form** and exhibit a highly negatively skewed distribution as shown on the left side of Figure 3.
- **The reward distribution varies across the arms** as different optimization methods can induce distinct reward processes.
- **The reward distribution may shift over time** when pulling the same arm (rested setting) as shown on the right side of Figure 3 where the reward distribution drifts over time.

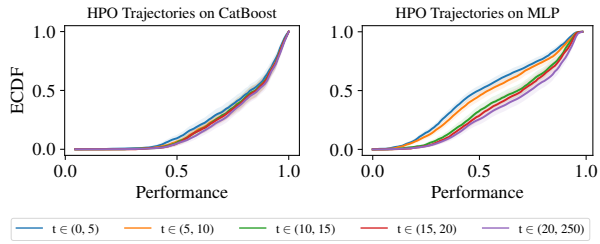


Figure 3: Exemplary HPO trajectories (from the Reshuffling benchmark) exhibit distributions that standard parametric models cannot capture.

Addressing these limitations is vital to address CASH+. As a consequence, extending PS-Max for complex reward processes requires addressing three key challenges: (2) the presence of heterogeneous reward distributions [Baudry et al., 2021]; (3) changes in the reward process over time [Fiandri et al., 2024]; and (4) varying costs associated with pulling different arms [Xia et al., 2015a]. To tackle them, we employ in-context learning through PFNs, as we discuss next.

[1] In MKB problems, there is no single best arm and the oracle arm depends on the budget T [Nishihara et al., 2016]. If T is known and sufficiently large, with $f(t) = T$, the agent aims to achieve the best final performance. With $f(t) = n_i + T - t$, the agent accounts for how many more iterations it can pull a given arm.

4.2 PS-PFN: Posterior sampling using PFNs

We need to accurately predict the posterior distribution of the maximum reward while accounting for changes in the reward distribution and left skewness of the distributions. We achieve this using prior-data fitted networks (PFNs) [Müller et al., 2022]. PFNs have demonstrated strong performance, particularly for predictive ML tasks on small tabular datasets [Hollmann et al., 2023, 2025], and have been successfully applied in related tasks such as learning curve extrapolation [Adriaensen et al., 2023] and Bayesian optimization [Müller et al., 2023a].

PFNs are pre-trained on synthetic data generated from a prior to perform approximate Posterior Predictive Distributions (PPD), without explicit retraining for new tasks. In other words, they are trained to predict some output y, conditioned on an input n and a training set D_{train} of given input-output examples. Müller et al. [2022] proved that minimizing this loss over many sampled tasks $(n, y) \cup D_{train}$ directly coincides with minimizing the KL divergence between the PFN's predictions and the true PPD. In essence, the PFN meta-learns to perform approximate posterior inference on synthetic tasks sampled from the prior, and at inference time, also does so for an unseen and real task. We focus on this model class since PFNs particularly perform well on small data tasks, enable instant predictions due to in-context learning, can be efficiently trained entirely on synthetic data, and the prior design allows incorporating assumptions and prior knowledge about data we expect to observe in the real world.

We use PFNs to model and estimate the unknown per-arm reward distributions as shown in Figure 2. Specifically, we use the maximum of the reward as output $y = \max(r_{i,1:t})$. For a sequence of rewards (and potentially additional information) from arm i with n_i observations, we obtain $D_{i,1:n_i} = \{(1, r_{i,1}), (2, \max(r_{i,1:2})), \ldots, (n_i, \max(r_{i,1:n_i}))\}$. We then apply in-context learning with PFNs, using the observed data $D_{i,1:n_i}$ as the context to output the posterior distribution $p(\max(r_{i,1:t}) \mid D_{i,1:n}, t)$. This distribution is then used to guide the agent's next arm selection.

To apply PFNs to our setup, we need the key ingredients: (a) a pre-training task that matches our problem formulation and (b) a prior to generate synthetic reward sequences that are close to what we expect to observe in the real world.

Architecture and pre-training task for reward distribution estimation. We use the same architecture used by Müller et al. [2022] which is a sequence Transformer [Vaswani et al., 2017]. For training, we use $D_{train} = \{(1, r_1), (2, \max(r_{1:2})), \ldots, (n_i, \max(r_{1:n_i}))\}$ as context and $D_{test} = (t,)$ as a test output. Following Adriaensen et al. [2023], we use a similar setup; we omit positional encodings since the input already includes a positional feature, and we mask the attention matrix so that each position only attends to training data positions. However, instead of uniformly sampling the cutoff points to split trajectories into training and test datasets for each batch, we use a harmonic distribution to assign more weight to smaller cutoff points. This is because predictions with fewer observations are more crucial in our setting. We provide more details on our architecture and training in Appendix D.

Once pre-trained, the PFN (with parameters θ) outputs a discretized distribution $P_\theta(\max(r_{i,1:t}) \mid D_{\text{test}}, t)$ with a fixed number of bins, and predicting the probability mass for each bin [Müller et al., 2022]. We will use this distribution to generate samples for line 7 in Algorithm 1 using $s_i \sim P_\theta(\cdot \mid D_{i,1:n_i}, t)$.

A prior for reward distributions. The prior design is critical to the effectiveness of PFNs [Breejen et al., 2024], and our reward distributions heavily depend on the optimization landscape of the AutoML task and can vary considerably in structure and complexity; thus, we define three priors (flat, semi-flat, and curved) to generate synthetic data for training PFNs. The resulting PFNs cover common characteristics of optimization problems:

1. The **flat prior** covers optimization landscapes common in HPO, which are characterized by large plateaus in the objective space, where many hyperparameter configurations yield similar performance [Pushak and Hoos, 2022]. As a result, the distribution of observed rewards tends to be left-skewed, as most configurations cluster around suboptimal yet similar values. Importantly, there is no significant shift in the distribution over time (see Figure 4, left).
2. The **semi-flat prior** covers trajectories where rewards gradually improve over time, reflecting a shift with a rapid decay in the distribution (see Figure 4, middle), for example, when the

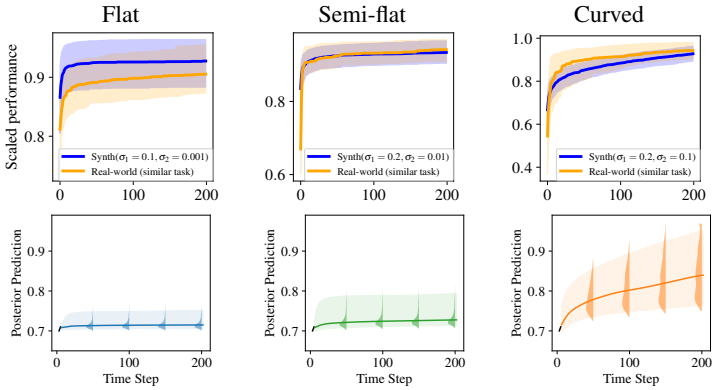


Figure 4: (Top) Different optimization trajectories for CASH+ and synthetically generated counterparts. (Bottom) Corresponding posterior predictions by PFNs given the same input (black line).

search space for optimization is large. This means that the magnitude of the shift is larger in the initial steps and gradually decreases over time when the optimization converges.

3. The **curved prior** covers scenarios where the amount of shift is considerably high and does not decay rapidly (see Figure 4, right). This behavior can be observed when training neural networks from scratch.

The semi-flat prior serves as a robust and broad prior covering most settings. The flat prior should be reserved for settings where the lower-level optimization converges fast and only little exploration is needed to approximate the posterior distribution. In contrast, the curved prior should be used for workflows that require more exploration to reduce uncertainty. In Appendix B.1, we describe how the reward sequence is generated for each prior.

4.3 Cost-awareness and handling budgets

Finally, we need to address varying costs for pulling different arms. In classical bandit settings, cost-awareness or budgeted bandits are typically incorporated by normalizing the reward by the cost, i.e., maximizing the reward-to-cost ratio [Ding et al., 2013, Xia et al., 2015b, 2016]. This approach is justified because pulling the arm with the highest reward rate decreases the cumulative regret the most [Cayci et al., 2020]. A similar heuristic is used in Bayesian optimization, where improvement-based acquisition functions are divided by (the logarithm of) cost [Snoek et al., 2012]; however, it may underperform without proper controlling [Xie et al., 2024]. In our setting, regret is only reduced when a higher reward than previously observed is obtained, making this strategy less appropriate.

Instead, we propose a different perspective: rather than adjusting the reward process by a cost estimate, we predict the posterior distribution of the maximal reward for a future time step adjusted by the remaining budget.

Figure 6 exemplarily shows that, for a fixed budget, e.g., 5 minutes, MLP achieves over 30 pulls (iterations), whereas CatBoost can be pulled only 20 times since each iteration has a higher cost (runtime). This means that the maximum number of pulls, i.e., the time step t for which we want to predict the distribution of maximal reward, varies across models. To consider this in our algorithm and to estimate the effective posterior for each arm, we must modify $f(t)$ in Algorithm 1 based on the remaining pulls per arm.

To estimate costs per pull per arm, we introduce B as the total (time) budget. Since runtimes can be stochastic, we model them as random variables and sample from their distribution e.g. log-normal distribution. Specifically, the cost of pulling arm i is $c_{i,t}$, and we adjust our decision-making (exploration vs. exploitation) based on both the observed empirical cost and the spent budget b. Let $c \sim p_c(\cdot \mid \{c_{i,t}\}_{t=1}^{n_i})$ denote a sample draw from the *posterior distribution of the cost* for arm i, given observed costs $\{c_{i,t}\}_{t=1}^{n_i}$. We then define the corrected time function as:

$$f_i(t) = n_i + \frac{B - b}{c \sim p_c(\cdot \mid \{c_{i,t}\}_{t=1}^{n_i})} \quad \text{where} \quad b = \sum_{i \leq k} \sum_{t=1}^{n_i} c_{i,t} \tag{6}$$

The intuition behind this definition is as follows: arm i has already been pulled n_i times, and $B - b$ represents the remaining budget. By dividing the remaining budget by the cost, we estimate the maximum number of additional pulls that can be performed. This allows us to predict the posterior distribution corresponding to that future iteration (see Appendix A.2 for more details).

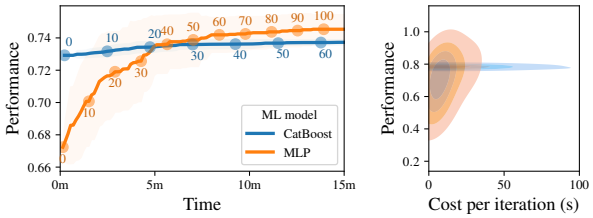


Figure 5: (Left) for 30 iterations of HPO, CatBoost outperforms MLP, while with the same budget, it is possible to run MLP for 50 iterations, outperforming CatBoost. (Right) The cost of one iteration is noisy.

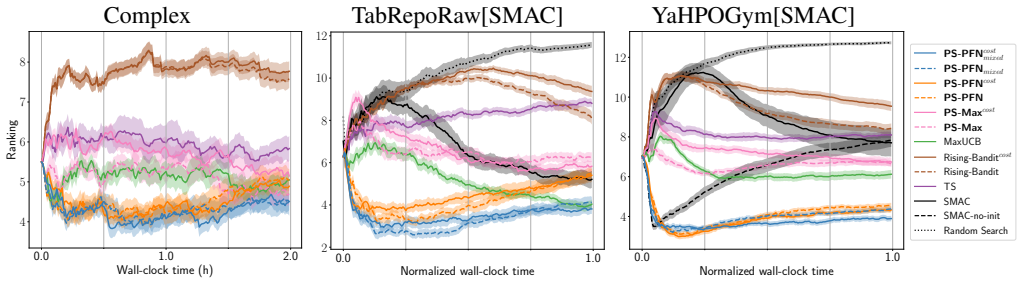


Figure 6: Average rank of algorithms on different benchmarks, lower is better. *SMAC* and *random search* perform *combined CASH* across the joint space.

5 Experiments on AutoML tasks

Next, we examine the empirical performance of our method in an AutoML setting via reporting average ranking for each benchmark.We will first briefly overview the experimental setup used across all experiments, and then discuss several research questions focusing on different aspects of this comparison.

Experimental setup. We use three AutoML benchmarks, implementing CASH and CASH+ tasks for tabular supervised learning, differing in the considered ML models, optimization method, and datasets (see Table C.1. For the existing CASH benchmarks, TabRepoRaw and YaHPOGym, we use available pre-computed HPO trajectories [Salinas and Erickson, 2024, Balef et al., 2025]. For our newly developed CASH+ task, Complex, we run 5 different methods across 30 datasets implementing heterogeneous ML workflows as seen in Figure 1.[2]

Baselines. We compare our methods against *MaxUCB* [Balef et al., 2025], *Rising Bandits* and *Rising Bandits*^{cost} [Li et al., 2020], which have been developed for the decomposed CASH task. Additionally, we compare against *Thompson Sampling* as a classical posterior sampling method. We use the default hyperparameter settings for all methods. As *combined CASH* baselines, we consider Bayesian optimization (*SMAC*) and *random search*.[3]

Our Methods. We evaluate three different methods and their cost-aware variants:

- *PS-Max* as the most basic baseline. We assume Gaussian rewards (see Appendix E).

[2] For all experiments, we used the AutoML Toolkit (AMLTK) [Bergman et al., 2024] and ran the optimization methods on a compute cluster equipped with Intel Xeon Gold 6240 CPUs and NVIDIA 2080 Ti GPUs, requiring 800 CPU and 50 GPU days.

[3] Only available for TabRepoRaw and YaHPOGym.

- *PS-PFN* uses the same PFN for modeling the reward distribution of each arm, pre-trained on the semi-flat prior.
- *PS-PFN*$_{mixed}$ uses a different PFN for each arm (see Appendix E).
- *PS-Max*^{cost}, *PS-PFN*^{cost}, and *PS-PFN*$^{cost}_{mixed}$ are the cost-aware extensions of the aforementioned methods. We assume costs follow a log-normal distribution (see Appendix E).

How does PS-Max and PS-PFN compare against *Thompson Sampling*? In Figure 6, we compare the average rank over time of PS-Max (- -) and PS-PFN (- -) to classical *Thompson Sampling* (–). While PS-Max and *Thompson Sampling* initially perform similarly, this gap increases with higher budget, highlighting that targeting maximum values is crucial for effectively solving CASH problems.

How does using a PFN improve the performance? Figure 6 shows a large improvement in ranking when comparing PS-Max (- -) with PS-PFN (- -). Table **??** also shows that this difference is statistically significant. Furthermore, using different PFNs per arm, each trained on a different prior, can further improve ranking, especially for high budgets, as seen by PS-PFN$_{mixed}$ (- -) performing better than PS-PFN (- -). However, this gap is smaller for YaHPOGym and Complex, underlining the importance of carefully choosing a prior.

How does PS-PFN compare against other *decomposed CASH* bandit methods? In Figure 6, we observe that *Rising Bandits* (–) perform substantially worse than our method. While the performance of *MaxUCB* (–) can be competitive, it tends to perform worse for low budgets. Since the non-PFN-based variant PS-Max (–) consistently performs worse than *MaxUCB* (–), we attribute the superior performance not to the bandit framework, but to leveraging an ML model.

How does cost-awareness improve the performance? Finally, we evaluate our cost-aware extension. For this, we compare the cost-aware variants (solid) to their counterpart (dashed) for PS-Max (– vs - -), PS-PFN (– vs - -), and PS-PFN$_{mixed}$ (– vs - -) in Figure 6.

Table 1: Percentage gain in number of pulled arms when using the cost-aware variant of each method.

	PS-PFN$^{cost}_{mixed}$	PS-PFN^{cost}	PS-Max^{cost}
YaHPOGym	7.50%	4.66%	2.94%
TabRepoRaw	15.36%	6.74%	7.55%
Complex	15.32%	17.35%	4.69%

Table 1 shows that the number of arms pulled increases when cost-awareness is enabled, meaning the agent observes more rewards within the same budget. However, this does not always improve final performance if cheaper arms are worse.

6 Discussion and Future Work

By extending the CASH framework to CASH+, we address the selection and adaptation of heterogeneous pipelines, covering modern ML workflows using fine-tuning, ensembling, and hyperparameter optimization. We address the resulting bi-level optimization problem as a max k-armed bandit problem and identify Posterior Sampling as a flexible framework for resource allocation. To efficiently model reward distributions that do not follow common forms, are different for each arm, and potentially shift over time, we exploit prior-fitted networks [Müller et al., 2023b]. The resulting method, PS-PFN and its extensions, outperforms prior approaches and paves the way for a data-driven development of AutoML solutions for CASH+. In the following, we briefly discuss limitations and avenues for future research.

Limitations. PS-PFN is computationally more expensive than other baselines since it leverages in-context learning for estimating the posterior distribution. Thus, in cases where iterations of the lower-level (optimization) method are fast, PS-PFN may dominate compute costs. Additionally, quadratic scaling in context length limits application for large budgets since we use reward observations as context. However, we note that in CASH+ tasks, iterations of the underlying method are typically cost-intensive, with tight budget constraints. We note that theoretically analyzing PS-PFN might be challenging due to the heuristic synthetic data generation and using ML models to approximate the posterior. Furthermore, while our cost-aware extension increases the number of arms pulled, performance does not constantly improve. We model the training cost with a log-normal distribution; however, it could be modeled more accurately based on the dataset and configurations. Lastly, though

our method can be used beyond AutoML tasks, it is sensitive to the prior choice for training the PFNs, and applying it to other tasks might require careful adaptation.

Acknowledgments

The authors are funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy – EXC number 2064/1 – Project number 390727645. The authors also thank the International Max Planck Research School for Intelligent Systems (IMPRS-IS).

References

Bingzhao Zhu, Xingjian Shi, Nick Erickson, Mu Li, George Karypis, and Mahsa Shoaran. Xtab: Cross-table pretraining for tabular transformers. In A. Krause, E. Brunskill, K. Cho, B. Engelhardt, S. Sabato, and J. Scarlett, editors, *Proceedings of the 40th International Conference on Machine Learning (ICML'23)*, volume 202 of *Proceedings of Machine Learning Research*. PMLR, 2023.

T. Chen and C. Guestrin. XGBoost: A scalable tree boosting system. In B. Krishnapuram, M. Shah, A. Smola, C. Aggarwal, D. Shen, and R. Rastogi, editors, *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'16)*, pages 785–794. ACM Press, 2016.

L. Prokhorenkova, G. Gusev, A. Vorobev, A. Dorogush, and A. Gulin. Catboost: Unbiased boosting with categorical features. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Proceedings of the 31st International Conference on Advances in Neural Information Processing Systems (NeurIPS'18)*, page 6639–6649. Curran Associates, 2018.

D. Holzmüller, L. Grinsztajn, and I. Steinwart. Better by default: Strong pre-tuned mlps and boosted trees on tabular data. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang, editors, *Proceedings of the 37th International Conference on Advances in Neural Information Processing Systems (NeurIPS'24)*, 2024.

Yury Gorishniy, Ivan Rubachev, Valentin Khrulkov, and Artem Babenko. Revisiting deep learning models for tabular data. In M. Ranzato, A. Beygelzimer, K. Nguyen, P. Liang, J. Vaughan, and Y. Dauphin, editors, *Proceedings of the 35th International Conference on Advances in Neural Information Processing Systems (NeurIPS'21)*. Curran Associates, 2021.

Noah Hollmann, Samuel Müller, 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. URL `https://www.nature.com/articles/s41586-024-08328-6`.

Wei Cui, Rasa Hosseinzadeh, Junwei Ma, Tongzi Wu, Yi Sui, and Keyvan Golestan. Tabular data contrastive learning via class-conditioned and feature-correlation based augmentation. *arXiv preprint arXiv:2404.17489*, 2024.

C. Thornton, F. Hutter, H. Hoos, and K. Leyton-Brown. Auto-WEKA: combined selection and Hyperparameter Optimization of classification algorithms. In I. Dhillon, Y. Koren, R. Ghani, T. Senator, P. Bradley, R. Parekh, J. He, R. Grossman, and R. Uthurusamy, editors, *The 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'13)*, pages 847–855. ACM Press, 2013.

M. Feurer, A. Klein, K. Eggensperger, J. Springenberg, M. Blum, and F. Hutter. Efficient and robust automated machine learning. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Proceedings of the 29th International Conference on Advances in Neural Information Processing Systems (NeurIPS'15)*, pages 2962–2970. Curran Associates, 2015.

B. Komer, J. Bergstra, and C. Eliasmith. Hyperopt-sklearn: Automatic hyperparameter configuration for scikit-learn. In F. Hutter, R. Caruana, R. Bardenet, M. Bilenko, I. Guyon, B. Kégl, and H. Larochelle, editors, *ICML workshop on Automated Machine Learning (AutoML workshop 2014)*, 2014.

Lars Kotthoff, Chris Thornton, Holger H Hoos, Frank Hutter, and Kevin Leyton-Brown. Auto-WEKA 2.0: Automatic model selection and hyperparameter optimization in WEKA. *Journal of Machine Learning Research*, 18(25):1–5, 2017.

M. Feurer, K. Eggensperger, S. Falkner, M. Lindauer, and F. Hutter. Auto-Sklearn 2.0: Hands-free automl via meta-learning. *Journal of Machine Learning Research*, 23(261):1–61, 2022.

Y. Li, J. Jiang, J. Gao, Y. Shao, C. Zhang, and B. Cui. Efficient automatic CASH via rising bandits. In F. Rossi, V. Conitzer, and F. Sha, editors, *Proceedings of the Thirty-Fourth Conference on Artificial Intelligence (AAAI'20)*, pages 4763–4771. Association for the Advancement of Artificial Intelligence, AAAI Press, 2020.

Y. Hu, X. Liu, and S. Liand Y. Yu. Cascaded algorithm selection with extreme-region UCB bandit. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(10):6782–6794, 2021.

Amir Rezaei Balef, Claire Vernade, and Katharina Eggensperger. Towards bandit-based optimization for automated machine learning. In *5th Workshop on practical ML for limited/low resource settings*, 2024. URL `https://openreview.net/forum?id=S5da3rzyuk`.

Amir Rezaei Balef, Claire Vernade, and Katharina Eggensperger. Put CASH on bandits: A max k-armed problem for automated machine learning. *arXiv preprint*, 2025.

Sebastian Pineda Arango, Fabio Ferreira, Arlind Kadra, Frank Hutter, and Josif Grabocka. Quicktune: Quickly learning which pretrained model to finetune and how. In S. Jegelka, T. Liang, D. Alvarez-Melis, A. Banerjee, M. Dudik, S. Kumar, H. Larochelle, and K. Q. Weinberger, editors, *Proceedings of the Twelfth International Conference on Learning Representations (ICLR'24)*, 2024.

Matt van den Nieuwenhuijzen, Carola Doerr, Jan N van Rijn, and Henry Gouk. Selecting pretrained models for transfer learning with data-centric meta-features. In *AutoML Conference 2024 (Workshop Track)*, 2024.

S. Müller, N. Hollmann, S. Arango, J. Grabocka, and F. Hutter. Transformers can do Bayesian inference. In *Proceedings of the International Conference on Learning Representations (ICLR'22)*, 2022. Published online: `iclr.cc`.

William R Thompson. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25(3-4):285–294, 1933.

Branislav Kveton, Boris Oreshkin, Youngsuk Park, Aniket Anand Deshmukh, and Rui Song. Online posterior sampling with a diffusion prior. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang, editors, *Proceedings of the 37th International Conference on Advances in Neural Information Processing Systems (NeurIPS'24)*, 2024.

Yueyang Liu, Benjamin Van Roy, and Kuang Xu. Nonstationary bandit learning via predictive sampling. In *International Conference on Artificial Intelligence and Statistics*, pages 6215–6244. PMLR, 2023.

Chenglei Shen, Xiao Zhang, Wei Wei, and Juncai Xu. Hyperbandit: Contextual bandit with hypernewtork for time-varying user preferences in streaming recommendation. In ACM, editor, *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management (CIKM)'23*. ACM, 2023.

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

My Phan, Yasin Abbasi Yadkori, and Justin Domke. Thompson sampling and approximate inference. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alche Buc, E. Fox, and R. Garnett, editors, *Proceedings of the 33rd International Conference on Advances in Neural Information Processing Systems (NeurIPS'19)*. Curran Associates, 2019.

T. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei. Language models are few-shot learners. In H. Larochelle, M. Ranzato, R. Hadsell, M.-F. Balcan, and H. Lin, editors, *Proceedings of the 34th International Conference on Advances in Neural Information Processing Systems (NeurIPS'20)*, pages 1877–1901. Curran Associates, 2020.

Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Michael Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. Decision transformer: Reinforcement learning via sequence modeling. In M. Ranzato, A. Beygelzimer, K. Nguyen, P. Liang, J. Vaughan, and Y. Dauphin, editors, *Proceedings of the 35th International Conference on Advances in Neural Information Processing Systems (NeurIPS'21)*. Curran Associates, 2021.

Licong Lin, Yu Bai, and Song Mei. Transformers as decision makers: Provable in-context reinforcement learning via supervised pretraining. In S. Jegelka, T. Liang, D. Alvarez-Melis, A. Banerjee, M. Dudik, S. Kumar, H. Larochelle, and K. Q. Weinberger, editors, *Proceedings of the Twelfth International Conference on Learning Representations (ICLR'24)*, 2024.

Jonathan Lee, Annie Xie, Aldo Pacchiano, Yash Chandak, Chelsea Finn, Ofir Nachum, and Emma Brunskill. Supervised pretraining can learn in-context reinforcement learning. In A. H. Oh, T. Hofmann, D. Belgrave, K. Cho, L. Metz, M. Raghu, and S. Nowozin, editors, *Proceedings of the 36th International Conference on Advances in Neural Information Processing Systems (NeurIPS'23)*, 2023.

Robert Nishihara, David Lopez-Paz, and Léon Bottou. No regret bound for extreme bandits. In *Artificial Intelligence and Statistics*, pages 259–267. PMLR, 2016.

Dorian Baudry, Patrick Saux, and Odalric-Ambrym Maillard. From optimality to robustness: Adaptive re-sampling strategies in stochastic bandits. In M. Ranzato, A. Beygelzimer, K. Nguyen, P. Liang, J. Vaughan, and Y. Dauphin, editors, *Proceedings of the 35th International Conference on Advances in Neural Information Processing Systems (NeurIPS'21)*. Curran Associates, 2021.

Marco Fiandri, Alberto Maria Metelli, and Francesco Trovò. Thompson sampling-like algorithms for stochastic rising rested bandits. In *Seventeenth European Workshop on Reinforcement Learning*, 2024. URL `https://openreview.net/forum?id=jaFhipqjxR`.

Yingce Xia, Haifang Li, Tao Qin, Nenghai Yu, and Tie-Yan Liu. Thompson sampling for budgeted multi-armed bandits. In Q. Yang and M. Wooldridge, editors, *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI'15)*, 2015a.

N. Hollmann, S. Müller, K. Eggensperger, and F. Hutter. TabPFN: A transformer that solves small tabular classification problems in a second. In *International Conference on Learning Representations (ICLR'23)*, 2023. Published online: `iclr.cc`.

Steven Adriaensen, Herilalaina Rakotoarison, Samuel Müller, and Frank Hutter. Efficient Bayesian learning curve extrapolation using prior-data fitted networks. In A. H. Oh, T. Hofmann, D. Belgrave, K. Cho, L. Metz, M. Raghu, and S. Nowozin, editors, *Proceedings of the 36th International Conference on Advances in Neural Information Processing Systems (NeurIPS'23)*, 2023.

Samuel G. Müller, Matthias Feurer, Noah Hollmann, and Frank Hutter. PFNs4BO: In-context learning for bayesian optimization. In A. Krause, E. Brunskill, K. Cho, B. Engelhardt, S. Sabato, and J. Scarlett, editors, *Proceedings of the 40th International Conference on Machine Learning (ICML'23)*, volume 202 of *Proceedings of Machine Learning Research*. PMLR, 2023a.

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. In I. Guyon, U. von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Proceedings of the 31st International Conference on Advances in Neural Information Processing Systems (NeurIPS'17)*. Curran Associates, Inc., 2017.

Felix den Breejen, Sangmin Bae, Stephen Cha, and Se-Young Yun. Fine-tuned in-context learning transformers are excellent tabular data classifiers. *arXiv preprint arXiv:2405.13396*, 2024.

Y. Pushak and H. Hoos. Automl loss landscapes. *ACM Transactions on Evolutionary Learning and Optimization*, 2(3):1–30, 2022.

Wenkui Ding, Tao Qin, Xu-Dong Zhang, and Tie-Yan Liu. Multi-armed bandit with budget constraint and variable costs. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 27, pages 232–238, 2013.

Yingce Xia, Haifang Li, Tao Qin, Nenghai Yu, and Tie-Yan Liu. Thompson sampling for budgeted multi-armed bandits. *arXiv preprint arXiv:1505.00146*, 2015b.

Yingce Xia, Wenkui Ding, Xu-Dong Zhang, Nenghai Yu, and Tao Qin. Budgeted bandit problems with continuous random costs. In *Asian conference on machine learning*, pages 317–332. PMLR, 2016.

Semih Cayci, Atilla Eryilmaz, and Rayadurgam Srikant. Budget-constrained bandits over general cost and reward distributions. In *International Conference on Artificial Intelligence and Statistics*, pages 4388–4398. PMLR, 2020.

Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. In P. Bartlett, F. Pereira, C. Burges, L. Bottou, and K. Weinberger, editors, *Proceedings of the 26th International Conference on Advances in Neural Information Processing Systems (NeurIPS'12)*. Curran Associates, 2012.

Qian Xie, Raul Astudillo, P. Frazier, Ziv Scully, and Alexander Terenin. Cost-aware bayesian optimization via the pandora's box gittins index. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang, editors, *Proceedings of the 37th International Conference on Advances in Neural Information Processing Systems (NeurIPS'24)*, 2024.

David Salinas and Nick Erickson. TabRepo: A large scale repository of tabular model evaluations and its AutoML applications. In Katharina Eggensperger, Roman Garnett, Joaquin Vanschoren, Marius Lindauer, and Jacob R. Gardner, editors, *Proceedings of the Third International Conference on Automated Machine Learning (AutoML 2024)*, volume 256 of *Proceedings of Machine Learning Research*, 2024.

Edward Bergman, Matthias Feurer, Aron Bahram, Amir Rezaei Balef, Lennart Purucker, Sarah Segel, Marius Lindauer, Frank Hutter, and Katharina Eggensperger. AMLTK: A Modular Automl Toolkit in Python. *Journal of Open Source Software*, 9(100):6367, 2024. doi: 10.21105/joss.06367. URL `https://doi.org/10.21105/joss.06367`.

S. Müller, M. Feurer, N. Hollmann, and F. Hutter. Pfns4bo: In-context learning for bayesian optimization. In A. Krause, E. Brunskill, K. Cho, B. Engelhardt, S. Sabato, and J. Scarlett, editors, *Proceedings of the 40th International Conference on Machine Learning (ICML'23)*, volume 202 of *Proceedings of Machine Learning Research*. PMLR, 2023b.

Tor Lattimore and Csaba Szepesvári. *Bandit Algorithms*. Cambridge University Press, 2020.

C. Wang, Q. Wu, M. Weimer, and E. Zhu. Flaml: A fast and lightweight automl library. In A. Smola, A. Dimakis, and I. Stoica, editors, *Proceedings of Machine Learning and Systems 3*, volume 3, pages 434–447, 2021.

Bernd Bischl, Giuseppe Casalicchio, Taniya Das, Matthias Feurer, Sebastian Fischer, Pieter Gijsbers, Subhaditya Mukherjee, Andreas C. Müller, László Németh, Luis Oala, Lennart Purucker, Sahithya Ravi, Jan N. van Rijn, Prabhant Singh, Joaquin Vanschoren, Jos van der Velde, and Marcel Wever. Openml: Insights from 10 years and more than a thousand papers. *Patterns*, 6(7):101317, 2025. ISSN 2666-3899. doi: https://doi.org/10.1016/j.patter.2025.101317.

F. Pfisterer, L. Schneider, J. Moosbauer, M. Binder, and B. Bischl. YAHPO Gym – an efficient multi-objective multi-fidelity benchmark for hyperparameter optimization. In I. Guyon, M. Lindauer, M. van der Schaar, F. Hutter, and R. Garnett, editors, *Proceedings of the First International Conference on Automated Machine Learning*. Proceedings of Machine Learning Research, 2022.

Thomas Nagler, Lennart Schneider, B. Bischl, and Matthias Feurer. Reshuffling resampling splits can improve generalization of hyperparameter optimization. *ArXiv*, abs/2405.15393, 2024.

A. Cowen-Rivers, W. Lyu, R. Tutunov, Z. Wang, A. Grosnit, R. Griffiths, A. Maraval, H. Jianye, J. Wang, J. Peters, and H. Ammar. HEBO: Pushing the limits of sample-efficient hyper-parameter optimisation. *Journal of Artificial Intelligence Research*, 74:1269–1349, 2022.

D. Kingma and J. Ba. Adam: A method for stochastic optimization. In *Proceedings of the International Conference on Learning Representations (ICLR'15)*, 2015. Published online: `iclr.cc`.

I. Loshchilov and F. Hutter. SGDR: Stochastic gradient descent with warm restarts. In *Proceedings of the International Conference on Learning Representations (ICLR'17)*, 2017. Published online: `iclr.cc`.

Eric Hans Lee, Valerio Perrone, Cédric Archambeau, and Matthias Seeger. Cost-aware bayesian optimization. In J. Peters and D. Sontag, editors, *Proceedings of The 36th Uncertainty in Artificial Intelligence Conference (UAI'20)*. PMLR, 2020.

Table of Contents for the Appendices

A Preliminiaries

A.1 Proof of Theorem

Theorem 1. (Analysis of PS-Max) If the expected maximum satisfies:

$$\mathbb{E}[\max(r_{i,1:t})] \geq F_i^{-1}\left(1 - \frac{1}{t^2}\right),$$

Then the number of times sub-optimal arm i is pulled grows logarithmically with the time horizon.

Proof. **Sampling the Maximum.** Let $r_1, \ldots, r_t$ be i.i.d. with CDF $F(x)$. The CDF of the maximum of reward $\max(r_{1:t})$ is:

$$F_{\max}(x) = P(\max(r_{1:t}) \leq x) = \prod_{i=1}^{t} P(r_i \leq x) = [F(x)]^t. \tag{7}$$

To sample from $F_{\max}$, we use inverse transform sampling:

1. Draw $U \sim \text{Uniform}(0, 1)$
2. Solve $F_{\max}(x) = U$:

$$[F(x)]^t = U \implies F(x) = U^{1/t} \implies x = F^{-1}(U^{1/t})$$

Thus, $\max(r_{1:t}) = F^{-1}(U^{1/t})$ follows the correct distribution.

Convergence Analysis. For the convergence analysis, let $F_n(x)$ denote the empirical CDF estimated from n i.i.d. samples. Using the Dvoretzky-Kiefer-Wolfowitz (DKW) inequality:

$$\mathbb{P}\left(\sup_x |F_n(x) - F(x)| \geq \epsilon\right) \leq \delta, \quad \text{with } \epsilon = \sqrt{\frac{\ln(2/\delta)}{2n}}. \tag{8}$$

Now we want to bound $|F_n^t(x) - F^t(x)|$. We know:

$$|F_n^t(x) - F^t(x)| \leq t|F_n(x) - F(x)|$$

Hence,

$$\mathbb{P}\left(\sup_x |F_n^t(x) - F^t(x)| \geq t\epsilon\right) \leq \delta. \tag{9}$$

This implies a worst-case deviation that grows linearly with t, leading to potentially linear regret. However, we are particularly interested in the concentration around the maximum $\max(r_{1:t})$, where tighter bounds can be established under some conditions.

We know the variance of the empirical CDF satisfies:

$$\text{Var}(F_n(x)) = \frac{F(x)(1 - F(x))}{n} \leq \frac{1 - F(x)}{n}.$$

Applying Bernstein's inequality, for any $\lambda > 0$ and near the right tail $x = \max(r_{1:t})$:

$$\mathbb{P}\left(|F_n(\max(r_{1:t})) - F(\max(r_{1:t}))| \geq \lambda\right) \leq 2\exp\left(-\frac{n\lambda^2}{2(1 - F(\max(r_{1:t}))) + \frac{2}{3}\lambda}\right).$$

With rewriting it as the form of ϵ at confidence level δ, we obtain:

$$\mathbb{P}\left(|F_n(\max(r_{1:t})) - F(\max(r_{1:t}))| \geq \epsilon)\right) \leq \delta, \quad \text{where } \epsilon = \sqrt{\frac{2(1 - F(\max(r_{1:t}))\ln(2/\delta)}{n}}. \tag{10}$$

This shows that the deviation is significantly smaller near the maximum when

$$1 - F(\max(r_{1:t})) \leq \frac{1}{t^2}.$$

By applying Equation 10 to Equation 9, we obtain the following refined bound under the additional condition:

$$\mathbb{P}\left(|F_n^t(\max(r_{1:t})) - F(\max(r_{1:t}))| \geq \epsilon\right) \leq \delta, \tag{11}$$

$$\text{where } \epsilon = \sqrt{\frac{\ln(2/\delta)}{n}}, \quad \text{and } \mathbb{E}[\max(r_{1:t})] \geq F^{-1}\left(1 - \frac{1}{t^2}\right) \tag{12}$$

under which the concentrations are fast enough. With this fast concentration, following the classical Thompson sampling proof (see Theorem 36.2 from [Lattimore and Szepesvári, 2020]), we can show that the number of times sub-optimal arms are pulled grows logarithmically with the time horizon. □

A.2 Budget Constraint and Variable Costs

In addition to the main paper, we provide more details on implementing cost-awareness in our method. For this, we consider CASH+ with a time budget constraint B and variable costs per arm, i.e. the evaluation of the model for the model state $A^{(i)}_{\boldsymbol{s}}$ for dataset $\mathbb{D}$ costs $\mathcal{C}(A^{(i)}_{\boldsymbol{s}}, \mathbb{D})$, with $\boldsymbol{L_S}^{(i)}$ being the list of previously evaluated configurations for model i. Formally,

$$A^{(i^*)} \in \arg\min_{A^{(i)} \in \mathcal{A}} \mathcal{L}(A^{(i)}_{\boldsymbol{s}^*}, \mathbb{D}) \tag{13}$$

$$\text{s.t. } \boldsymbol{S}^* \in \arg\min_{\boldsymbol{s} \in \boldsymbol{S^{(i)}}} \mathcal{L}(A^{(i)}_{\boldsymbol{s}}, \mathbb{D}). \tag{14}$$

$$\text{s.t. } \sum_{i \in K} \sum_{\boldsymbol{s} \in \boldsymbol{L^{(i)}_S}} \mathcal{C}(A^{(i)}_{\boldsymbol{s}}, \mathbb{D}) \leq B \tag{15}$$

Given an overall budget of B, we define $\boldsymbol{s}_t$ to be the model state proposed by the optimizer in the lower level and $r_{i,t}$ to be the feedback to arm i obtained by evaluating $\boldsymbol{s}_t$. To be consistent with the bandit literature, we maximize the negative loss with cost $c_{i,t}$:

$$\begin{aligned} r_{i,t} &= -\mathcal{L}(A^i_{\boldsymbol{s}_t}, \mathbb{D}) \\ c_{i,t} &= \mathcal{C}(A^{(i)}_{\boldsymbol{s}_t}, \mathbb{D}) \end{aligned} \tag{16}$$

The goal is then to find the best-performing algorithm A^* and its state $\boldsymbol{s}^*$ with spending a time budget B, and we assume the cost of pulling arm i follows a sub-Gaussian distribution with expected mean c_i.

$$R(B) = \max_{i \leq K} \mathbb{E}\left[\max_{t \leq \lfloor B/c_i \rfloor} r_{i,1:t}\right] - \mathbb{E}\left[\max_{t \leq T} r_{I_t,1:t}\right] \tag{17}$$

$$\text{s.t.} \quad \sum_{t=1}^{T} c_{I_t,t} \leq B \tag{18}$$

Each arm experiences a different time horizon $T_i = \frac{B}{c_i}$. If we sort arms based on optimality (denoting the optimal arm as 1 and the most sub-optimal arm as K), we have:

$$\mathbb{E}\left[\max_{t \leq \lfloor B/c_1 \rfloor} r_{1,1:t}\right] > \mathbb{E}\left[\max_{t \leq \lfloor B/c_2 \rfloor} r_{2,1:t}\right] > \cdots > \mathbb{E}\left[\max_{t \leq \lfloor B/c_K \rfloor} r_{K,1:t}\right] \tag{19}$$

Let at time step t, arm i be pulled for n_i times and the spent budget be $b = \sum_{i \leq k} \sum_{t=1}^{n_i} c_{i,1:t}$. The remaining budget at time step t is $B - b$. And arm i has been pulled for n_i times. In the base case, we can only pull this arm for $\frac{B-b}{c_i}$ times. Meaning that instead of $f(t) = t$ we need to predict the posterior at $f(t) = n_i + \frac{B-b}{c_i}$. However, the cost of pulling arms is noisy (e.g., the cost of pulling an arm implementing hyperparameter optimization will depend on the model size and training hyperparameters for each configuration), and we need to estimate c_i. We model these costs as a random distribution to estimate c_i, and sample the cost from this distribution.

$$f_i(t) = n_i + \frac{B - b}{c \sim p_c(\cdot \mid \{c_{i,t}\}_{t=1}^{n_i})} \quad \text{where} \quad b = \sum_{i \leq k} \sum_{t=1}^{n_i} c_{i,t}. \tag{20}$$

Principally, without having prior information about maximum budget B, one can use $f(t)$ as follows:

$$f_i(t) = \frac{b}{c \sim p_c(\cdot \mid \{c_{i,t}\}_{t=1}^{n_i})} \quad \text{where} \quad b = \sum_{i \leq k} \sum_{t=1}^{n_i} c_{i,t}. \tag{21}$$

This is the cost-scaled version of $f(t) = t$.

A.3 Importance of priors

Here, we study the sensitivity of the PFN concerning its prior. For this, we pre-trained three separate PFNs using rewards generated from a truncated skewed normal distribution. The priors mainly differ in the parameter range used to sample from to initiate the distribution , as shown in Table A.1. We then evaluated them as a model for PS-PFN on a synthetic multi-armed bandit task with 7 arms, with the reward distributions characterized by the mean μ and standard deviation σ values listed in Table A.2. To assess generalization across skewness, we varied the skewness parameter a within the range $[-100, 100]$ in increments of 5.

Dataset	Parameters
Neg	$a \sim \mathcal{U}(-100, -20)$
	$\mu \sim \mathcal{U}(0.0, 1.0)$
	$\sigma \sim \mathcal{U}(0, 0.2)$
Pos	$a \sim \mathcal{U}(20, 100)$
	$\mu \sim \mathcal{U}(0.0, 1.0)$
	$\sigma \sim \mathcal{U}(0, 0.2)$
Mix	$a \sim \mathcal{U}(-100, 100)$
	$\mu \sim \mathcal{U}(0.0, 1.0)$
	$\sigma \sim \mathcal{U}(0, 0.2)$

Table A.1: Parameters of the priors per dataset type used to train the PFNs.

Arm	Parameter μ	Parameter σ
1	0.80	0.05
2	0.75	0.05
3	0.70	0.05
4	0.60	0.05
5	0.70	0.10
6	0.60	0.10
7	0.50	0.10

Table A.2: Reward distribution parameters for the synthetic MAB tasks.

We analyze the impact of prior distributions by evaluating both ranking and normalized regret plots. We define three types of environments based on the skewness parameter a: **Neg** for strongly left-skewed distributions ($a \in [-100, -45]$), **Mix** for mildly skewed or symmetric distributions ($a \in [-45, 45]$), and **Pos** for right-skewed distributions ($a \in [45, 100]$).

As shown in the ranking plot (Figure A.1), an interesting observation is that *PS-PFN(Pos)* underperforms even in positively skewed environments. This is likely because right-skewed reward distributions make the problem more exploration-heavy, requiring more time horizon than the available budget allows. This trend is also reflected in the normalized regret plot (Figure A.2), where *PS-PFN(Pos)* exhibits higher regret.

Furthermore, the number of arm pulls reported in Figure A.2 shows that *PS-PFN(Pos)* explores more frequently than the other models. Finally, the heatmap in Figure A.4 highlights that *PS-PFN(Neg)* performs best in environments with left-skewed reward distributions.

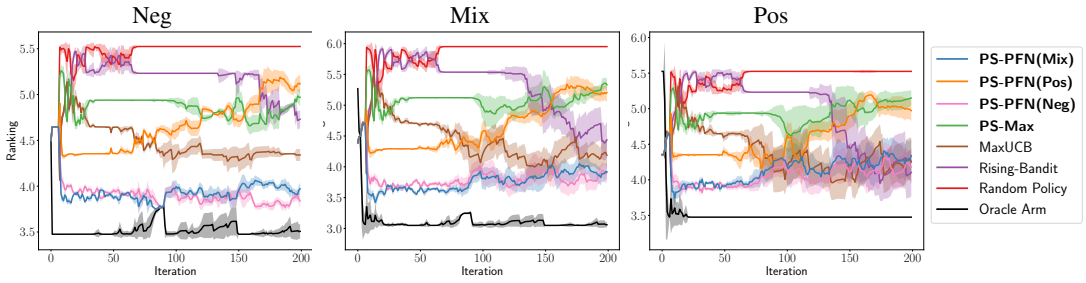


Figure A.1: Comparing ranking of *PS-PFN* with different priors on synthetic tasks.

A.4 Out of distribution

Here, we examine two PFNs trained on a negatively skewed distribution. However, one of them rarely sees values lower than 0.8, we call it *Neg(limited)*. In the prediction, as seen in Figure A.5 it cannot extrapolate properly. Therefore, for generating synthetic trajectories, we always choose the parameter distribution such that the trajectories cover the whole range $[0, 1]$ during pre-training PFNs.

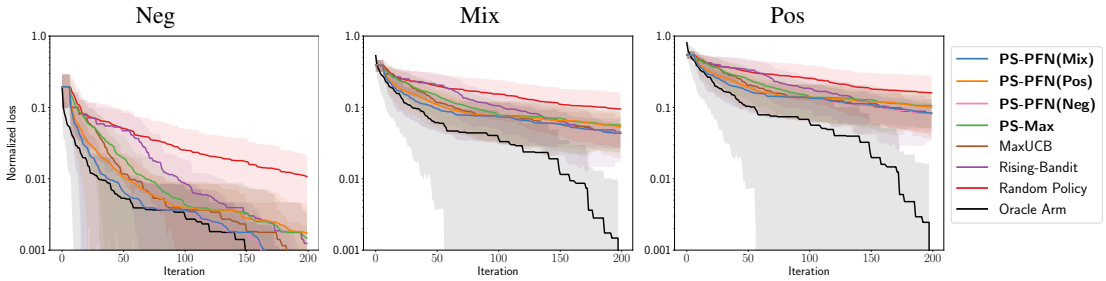


Figure A.2: Comparing normalized loss of *PS-PFN* with different priors on synthetic tasks.

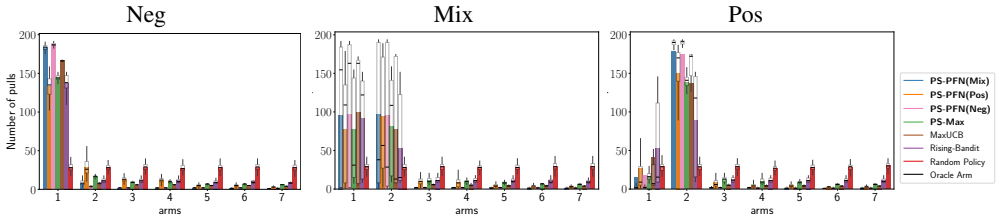


Figure A.3: Comparing the number of pulls for each arm for different methods, arms are sorted by optimality.

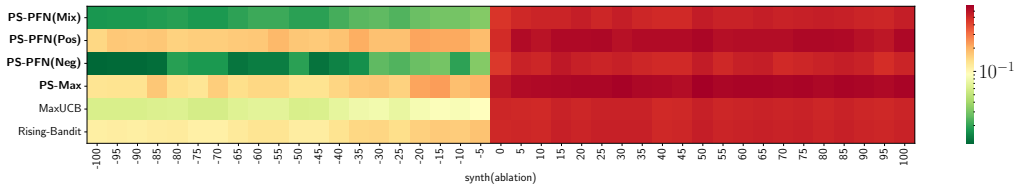


Figure A.4: Heatmap shows the final regret of each algorithm for different skewness in the arms, lower (green) is better.

Dataset	Parameters
Neg	$a \sim \mathcal{U}(-100, -20$ $\mu \sim \mathcal{U}(0, 1)$ $\sigma \sim \mathcal{U}(0, 0.2)$
Neg(limited)	$a \sim \mathcal{U}(-100, -20$ $\mu \sim \mathcal{U}(0.8, 1)$ $\sigma \sim \mathcal{U}(0, 0.2)$

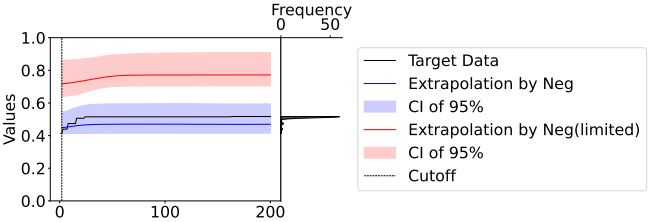


Figure A.5: out of distribution analysis

B Synthetic data

B.1 Generating synthetic reward trajectories

To train our PFNs, we must generate synthetic trajectories for each prior. We implement this using two distributions d_1 and d_2.

The first distribution, d_1, models the rewards and captures characteristics of the optimization output and its uncertainty, i.e., left skewness when rewards are i.i.d. as is typical with random search. The second distribution, d_2, models the shift over time and introduces non-stationarity by modeling changes in the reward distribution over time. The reward sequence is generated by element-wise multiplication of the samples from d_1 with the sorted samples from d_2. Concretely, to generate a synthetic trajectory, we draw one sample for each parameter of the distributions, e.g., μ_1. We then draw $t = 200$ samples from d_1 and d_2, denoted by $r_{1,1:t}$ and $r_{2,1:t}$, respectively. Next, we sort $r_{2,1:t}$ in ascending order and perform element-wise multiplication with $r_{1,1:t}$ to obtain the final reward sequence $r_{1:t} = r_{1,1:t} \cdot \mathrm{sort}(r_{2,1:t})$.

Choosing appropriate distributions for d_1 and d_2 and the range of their parameters is challenging, and suitable values depend on the task at hand. We derive suitable ranges based on holdout tasks as detailed in Appendix B.2.

Distribution d_1 (Reward Uncertainty) The distribution d_1 models the uncertainty in the rewards and is designed to be **left-skewed**. We use a truncated skewed normal distribution over the interval $[0, 1]$, where the mean μ_1 is sampled from Uniform$(0, 1)$ (see Appendix A.4), the skewness is sampled from Uniform$(-100, -20)$, and the standard deviation σ_1 controls the level of model uncertainty.

Distribution d_2 (Non-Stationarity) The distribution d_2 captures non-stationarity by introducing time-dependent shifts in the reward distribution. Similar to d_1, it is also modeled using a truncated skewed normal distribution over $[0, 1]$, with a fixed mean of 1, skewness sampled from Uniform$(-100, -20)$, and standard deviation σ_2 controlling the severity of the non-stationary behavior.

For each prior, we specify different values for σ_1 and σ_2 as summarized in Table B.1. Figure 4 shows the posterior predictions of these three PFNs for the same input sequence. As seen, the output closely approximates the synthetic trajectories. Additionally, in Appendix A.3, we illustrate the sensitivity of our PFNs to prior selection by evaluating PS under different assumptions. Notably, the term $(1 - \mu_1)$ is used because reward processes with a low-performing mean tend to exhibit greater non-stationarity.

Table B.1: Parameters for the distributions used to generate synthetic reward trajectories for each prior.

Type	**Parameters**	
Flat	$\sigma_1 \sim$ Uniform$(0, 0.1)$	$\sigma_2 \sim$ Uniform$(0.001(1 - \mu_1), 0.001)$
Semi-flat	$\sigma_1 \sim$ Uniform$(0, 0.2)$	$\sigma_2 \sim$ Uniform$(0.01(1 - \mu_1), 0.01)$
Curved	$\sigma_1 \sim$ Uniform$(0, 0.2)$	$\sigma_2 \sim$ Uniform$(0.1(1 - \mu_1), 0.1)$

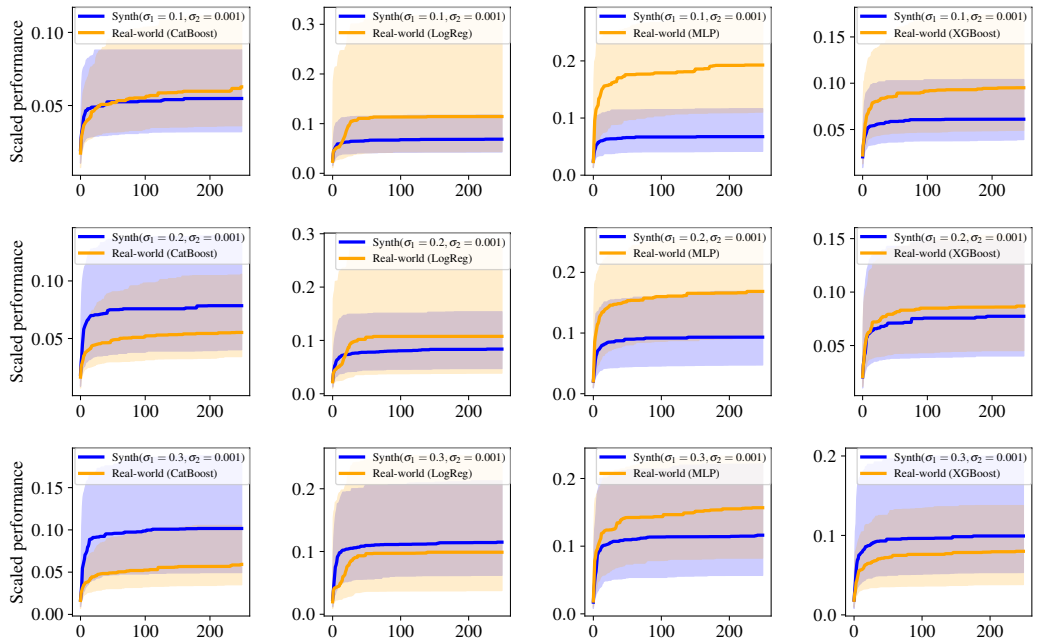


Figure B.1: Studying the effect of $\hat{\sigma}_1$

B.2 Trajectory Analysis

Here, we detail how we designed the priors for training our PFNs, based on analyzing the optimization trajectories of the Reshuffling benchmark. We aim to find robust default parameters for d_1 and d_2. For simplicity and clarity, we define $\hat{\sigma}_1$ and $\hat{\sigma}_2$ as parameters to control range of σ_1 and σ_2 such that $\sigma_1 \sim \text{Uniform}(0, \hat{\sigma}_1)$ and $\sigma_2 \sim \text{Uniform}(\hat{\sigma}_2(1-\mu_1), \hat{\sigma}_2)$. Our goal is to identify suitable values for $\hat{\sigma}_1$ and $\hat{\sigma}_2$.

We start by using $\hat{\sigma}_2 = 0.001$, which is close to zero. This eliminates the effect of d_2 and yields stationary trajectories. In Figure B.1, we show the effect of $\hat{\sigma}_1$. Each row uses the same value for $\hat{\sigma}_1$, and each column shows results for a different ML pipeline (arm). Ideally, the blue area fully covers the yellow area, and the shape of the average trajectory is similar in terms of flatness or incremental trend. Comparing the blue curve (average of synthetic trajectories) with the yellow curve (average of real-world trajectories), we found that most often $\hat{\sigma}_1 = 0.3$ is too high (last row). For CatBoost, $\hat{\sigma}_1 = 0.1$ yields good results, while for XGBoost, $\hat{\sigma}_1 = 0.2$ fits better. We use $\hat{\sigma}_1 = 0.1$ and $\hat{\sigma}_2 = 0.001$ for **flat** prior.

We now increase the amount of non-stationarity by changing $\hat{\sigma}_2$, while keeping $hat\sigma_1$ fixed at 0.2. As seen in Figure B.2, $\hat{\sigma}_2 = 0.01$ covers better slight non-stationarity. For instance, in the case of MLP and LogReg, the increasing trend of the synthetic trajectories with $\hat{\sigma}_2 = 0.01$ aligns more closely with the real-world trajectories compared to $\hat{\sigma}_2 = 0.001$ (see Figure B.1). We use $\hat{\sigma}_1 = 0.2$ and $\hat{\sigma}_2 = 0.01$ for **semi-flat** prior.

Notably, we also use $\hat{\sigma}_1 = 0.2$ and $\hat{\sigma}_2 = 0.1$ for our **curved** prior as seen in Figure B.2.

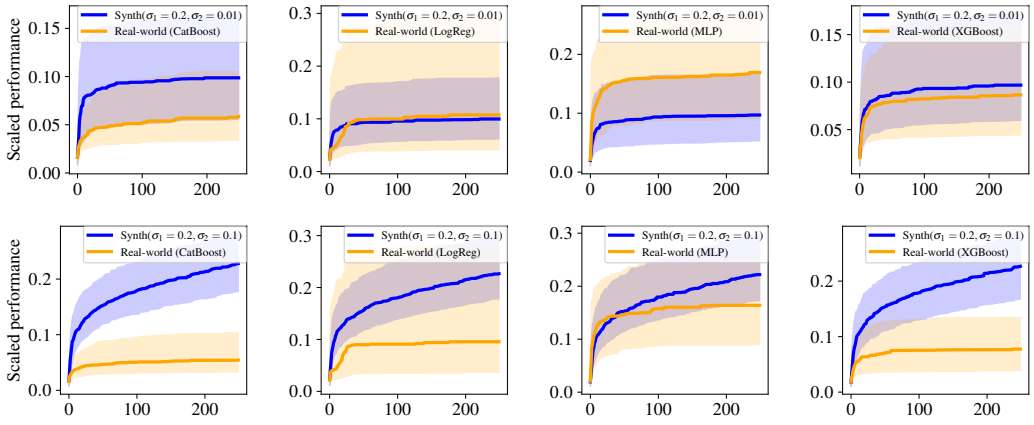


Figure B.2: Studying the effect of $\hat{\sigma}_2$

C Benchmarks

In our empirical evaluation, we use three AutoML benchmarks (TabRepoRaw, YaHPOGym and Complex) as described in Table C.1. Additionally, we make use of the Reshuffling benchmark during development (as detailed in Section B.2). We describe all of them in more detail here.

Complex is a novel CASH+ benchmark which we build for this work, covering optimization methods for 5 different workflows covering fine-tuning XTAB [Zhu et al., 2023] and TabForestPFN [Breejen et al., 2024], running the AutoML system FLAML [Wang et al., 2021], optimizing hyperparameters of a shallow neural network [Holzmüller et al., 2024] and PHETabPFNv2 [Hollmann et al., 2025]. Table C.2 provides details for each model, including the optimization method and the corresponding hyperparameter search space. Additionally, Table C.3 lists the datasets from OpenML [Bischl et al., 2025] used in these experiments. We ran the optimization with four different random seeds, using seven folds to train the ML model and its optimization process, and the remaining three folds to evaluate the loss. Each run produced 12 distinct optimization trajectories as repetitions.

YaHPOGym [Pfisterer et al., 2022] is a surrogate benchmark that covers six ML models (details in Table C.4) on 103 datasets and uses a regression model (surrogate model) to predict performances for queried hyperparameter settings. We re-use HPO trajectories provided by Balef et al. [2025]. In our empirical evaluation, we exclude the last 3 datasets we used to derive our priors (see Section B.2).

TabRepoRaw uses the non-discretized search space from TabRepoRaw [Salinas and Erickson, 2024] (details in Table C.5). We re-use HPO trajectories provided by Balef et al. [2025], which cover 30 datasets (see context name `D244_F3_C1530_30` in the original TabRepoRaw code repository). We run Bayesian optimization for 8 more datasets to study trajectories, which we exclude from the evaluation.

Reshuffling is an existing benchmark provided by Nagler et al. [2024] from which we analyze HPO trajectories to develop our PFNs. It covers 4 ML models (details in Table C.6) across 10 datasets, with 10 repetitions and 3 different validation split ratios within a budget of 250 iterations. As an HPO method, it uses *HEBO* [Cowen-Rivers et al., 2022]. Since this benchmark does not contain cost information, we do not use it in our empirical evaluation.

name	#models	#tasks	type	HPO meth. (rep.)	budget	reference
YaHPOGym	6	103	surrogate	*SMAC* (32)	200 iterations	[Pfisterer et al., 2022]
TabRepoRaw	7	30	raw	*SMAC* (32)	200 iterations	-
Complex	5	30	raw	Various (12)	2 hours	-

Table C.1: Overview of AutoML tasks

Table C.2: ML models in Complex.

ML model	Optimization method	Hyperparameters	Search Space	Range	Info
XTab	Fine-tuning (AdamW)				
FLAML	HPO (LightGBM)	num_leaves	integer	[4, 32768]	log
		max_depth	integer	[-1, 15]	-1 = no limit
		learning_rate	continuous	[0.0001, 1.0]	log
		n_estimators	integer	[16, 32768]	log
		min_child_samples	integer	[1, 100]	
		subsample	continuous	[0.1, 1.0]	
		colsample_bytree	continuous	[0.1, 1.0]	
		reg_alpha	continuous	[0.0, 10.0]	
		reg_lambda	continuous	[0.0, 10.0]	
	HPO (XGBoost)	max_depth	integer	[3, 15]	
		learning_rate	continuous	[0.0001, 1.0]	log
		n_estimators	integer	[16, 32768]	log
		subsample	continuous	[0.1, 1.0]	
		colsample_bytree	continuous	[0.1, 1.0]	
		reg_alpha	continuous	[0.0, 10.0]	
		reg_lambda	continuous	[0.0, 10.0]	
	HPO (CatBoost)	learning_rate	continuous	[0.0001, 1.0]	log
		depth	integer	[4, 10]	
		l2_leaf_reg	continuous	[1.0, 10.0]	
		iterations	integer	[16, 32768]	log
		rsm	continuous	[0.5, 1.0]	
		border_count	integer	[32, 255]	
		bootstrap_type	categorical	{Bayesian, Bernoulli, MVS}	
		bagging_temperature	continuous	[0, 1]	for Bayesian
	HPO (Random Forest)	max_depth	integer	[2, 15]	
		n_estimators	integer	[16, 32768]	log
		max_features	categorical	{sqrt, log2, None}	
	HPO (Logistic Regression)	C	continuous	[1e-6, 1e6]	log
RealMLP	HPO (SMAC)	num_emb_type	categorical	{none, pbld, pl, plr}	
		add_front_scale	categorical	{True, False}	
		lr	continuous	[0.02, 0.3]	log
		p_drop	categorical	{0.00, 0.15, 0.30}	
		act	categorical	{selu, relu, mish}	
		hidden_sizes	categorical	{[256,256,256], [64,64,64,64,64], [512]}	
		wd	ordinal	{0.0, 0.02}	
		plr_sigma	continuous	[0.05, 0.5]	log
		ls_eps	ordinal	{0.0, 0.1}	
		epochs	ordinal	{16, 32, 64, 128, 256}	default=256
		batch_size	ordinal	{32, 64, 128, 256}	default=256
TabForestPFN	Fine-tuning (AdamW)				
TabPFN (v2)	Post-Hoc Ensembling				

Table C.3: Dataset in Complex.

index	task id	dataset name	number of samples	number of features	number of categorical features	number of calsses
1	3593	2dplanes	40768	11	1	2
2	3627	cpu-act	8192	22	1	2
3	3688	houses	20640	9	1	2
4	3844	kdd-internet-usage	10108	69	69	2
5	3882	pendigits	10992	17	1	2
6	3904	jm1	10885	22	1	2
7	7295	Click-prediction-small	39948	10	1	2
8	9933	volcanoes-c1	28626	4	1	5
9	9965	skin-segmentation	245057	4	1	2
10	9977	nomao	34465	119	30	2
11	9986	gas-drift	13910	129	1	6
12	9987	gas-drift-different-concentrations	13910	130	1	6
13	10106	CreditCardSubset	14240	31	1	2
14	14965	bank-marketing	45211	17	10	2
15	34537	PhishingWebsites	11055	31	31	2
16	34539	Amazon-employee-access	32769	10	10	2
17	361056	california	20634	9	1	2
18	361063	house-16H	13488	17	1	2
19	361071	jannis	57580	55	1	2
20	361110	electricity	38474	9	2	2
21	361111	eye-movements	7608	24	4	2
22	361112	KDDCup09-upselling	5032	46	12	2
23	361113	covertype	423680	55	45	2
24	361114	rl	4970	13	8	2
25	361115	road-safety	111762	33	4	2
26	361116	compass	16644	18	10	2
27	361282	albert	58252	32	11	2
28	362098	Dota2-Games-Results-Data-Set	102944	117	117	2
29	362407	adult	48790	15	9	2
30	363550	cdc-diabetes	253680	22	1	2

Table C.4: Hyperparameter spaces for ML models in YaHPOGym.

ML model	Hyperparameter	Type	Range	Info
-	trainsize	continuous	[0.03, 1]	=0.525 (fixed)
	imputation	categorical	{mean, median, hist}	=mean (fixed)
Glmnet	alpha	continuous	[0, 1]	
	s	continuous	[0.001, 1097]	log
Rpart	cp	continuous	[0.001, 1]	log
	maxdepth	integer	[1, 30]	
	minbucket	integer	[1, 100]	
	minsplit	integer	[1, 100]	
SVM	kernel	categorical	{linear, polynomial, radial}	
	cost	continuous	[4.5e-05, 2.2e4]	log
	gamma	continuous	[4.5e-05, 2.2e4]	log, kernel
	tolerance	continuous	[4.5e-05, 2]	log
	degree	integer	[2, 5]	kernel
AKNN	k	integer	[1, 50]	
	distance	categorical	{l2, cosine, ip}	
	M	integer	[18, 50]	
	ef	integer	[7, 403]	log
	ef_construction	integer	[7, 403]	log
Ranger	num.trees	integer	[1, 2000]	
	sample.fraction	continuous	[0.1, 1]	
	mtry.power	integer	[0, 1]	
	respect.unordered.factors	categorical	{ignore, order, partition}	
	min.node.size	integer	[1, 100]	
	splitrule	categorical	{gini, extratrees}	
	num.random.splits	integer	[1, 100]	splitrule
XGBoost	booster	categorical	{gblinear, gbtree, dart}	
	nrounds	integer	[7, 2980]	log
	eta	continuous	[0.001, 1]	log, booster
	gamma	continuous	[4.5e-05, 7.4]	log, booster
	lambda	continuous	[0.001, 1097]	log
	alpha	continuous	[0.001, 1097]	log
	subsample	continuous	[0.1, 1]	
	max_depth	integer	[1, 15]	booster
	min_child_weight	continuous	[2.72, 148.4]	log, booster
	colsample_bytree	continuous	[0.01, 1]	booster
	colsample_bylevel	continuous	[0.01, 1]	booster
	rate_drop	continuous	[0, 1]	booster
	skip_drop	continuous	[0, 1]	booster

Table C.5: Hyperparameter spaces for ML models in TabRepoRaw.

ML model	Hyperparameter	Type	Range	Info	Default value
NN(PyTorch)	learning rate	continuous	[1e-4, 3e-2]	log	3e-4
	weight decay	continuous	[1e-12, 0.1]	log	1e-6
	dropout prob	continuous	[0, 0.4]		0.1
	use batchnorm	categorical	False, True		
	num layers	integer	[1, 5]		2
	hidden size	integer	[8, 256]		128
	activation	categorical	relu, elu		
NN(FastAI)	learning rate	continuous	[5e-4, 1e-1]	log	1e-2
	layers	categorical	[200], [400], [200, 100], [400, 200], [800, 400], [200, 100, 50], [400, 200, 100]		
	emb drop	continuous	[0.0, 0.7]		0.1
	ps	continuous	[0.0, 0.7]		0.1
	bs	categorical	256, 128, 512, 1024, 2048		
	epochs	integer	[20, 50]		30
CatBoost	learning rate	continuous	[5e-3 ,0.1]	log	0.05
	depth	integer	[4, 8]		6
	l2 leaf reg	continuous	[1, 5]		3
	max ctr complexity	integer	[1, 5]		4
	one hot max size	categorical	2, 3, 5, 10		
	grow policy	categorical	SymmetricTree, Depthwise		
LightGBM	learning rate	continuous	[5e-3 ,0.1]	log	0.05
	feature fraction	continuous	[0.4, 1.0]		1.0
	min data in leaf	integer	[2, 60]		20
	num leaves	integer	[16, 255]		31
	extra trees	categorical	False, True		
XGBoost	learning rate	continuous	[5e-3 ,0.1]	log	0.1
	max depth	integer	[4, 10]		6
	min child weight	continuous	[0.5, 1.5]		1.0
	colsample bytree	continuous	[0.5, 1.0]		1.0
	enable categorical	categorical	False, True		
Extra-trees	max leaf nodes	integer	[5000, 50000]		
	min samples leaf	categorical	1, 2, 3, 4, 5, 10, 20, 40, 80		
	max features	categorical	sqrt, log2, 0.5, 0.75, 1.0		
Random-forest	max leaf nodes	integer	[5000, 50000]		
	min samples leaf	categorical	1, 2, 3, 4, 5, 10, 20, 40, 80		
	max features	categorical	sqrt, log2, 0.5, 0.75, 1.0		

Table C.6: Hyperparameter spaces for ML models in Reshuffling.

ML model	Hyperparameter	Type	Range	Info
Funnel-Shaped MLP	learning rate	continuous	[1e-4, 1e-1]	log
	num layers	integer	[1, 5]	
	max units	categorical	64, 128, 256, 512	
	batch size	categorical.	16, 32, ..., max_batch_size	
	momentum	continuous.	[0.1, 0.99]	
	alpha	continuous.	[1e-6, 1e-1]	log
Elastic Net	C	continuous	[1e-6, 10e4]	log
	l1 ratio	continuous	[0.0, 1.0]	
XGBoost	max depth	integer	[2, 12]	log
	alpha	continuous	[1e-8, 1.5]	log
	lambda	continuous	[1e-8, 1.0]	log
	eta	continuous	[0.01, 0.3]	log
CatBoost	learning rate	continuous	[0.01 ,0.3]	log
	depth	integer	[2, 12]	
	l2 leaf reg	continuous	[0.5, 30]	

D PFN architecture

The PFN architecture inherits standard Transformer hyperparameters: the number of layers (`nlayers`), attention heads (`nheads`), embedding dimension (`emsize`), and hidden dimension (`nhidden`). Specifically, our configuration uses 6 layers, 4 attention heads, and a hidden size of 512. For training, we adopt the Adam optimizer [Kingma and Ba, 2015] (learning rate 10^{-4}, batch size 100) with cosine annealing [Loshchilov and Hutter, 2017], including a linear warmup phase over the first 25% of epochs. We set $m = 200$, meaning the PFN is trained to extrapolate sequences of up to 200 steps (e.g., HPO or fine-tuning iterations). The PFN output is a discretized mass distribution with a fixed number of bins. We set the number of bins to $1,000$ as a hyperparameter. The model is trained for $1,000$ epochs, resulting in training with $100,000$ synthetic trajectories, which takes approximately one GPU-day.

E Priors

We use three types of priors throughout our work: (a) a prior to efficiently estimate costs to make our methods cost-aware, (b) a prior to estimate reward distributions to evaluate PS-Max, and (c) a prior to generate synthetic reward trajectories to train PFNs to be used for PS-PFN$_{mixed}$. We discuss the use of these in the following.

Priors for cost-awareness. We assume that the cost (i.e., runtime) follows a log-normal distribution, with corresponding parameters provided in Table E.1. The log-normal distribution is a natural choice for modeling the cost of training machine learning models, as training costs often exhibit a skewed, right-tailed behavior where higher costs are less frequent but more extreme. This makes the log-normal distribution a good fit for such scenarios (as illustrated in Figure 1 from [Lee et al., 2020]). We truncated the log-normal distribution to avoid unreasonably high or low costs, as shown in Table E.1.

Likelihood Model	Prior Parameters	Notes
Log-Normal	$\mu_0 = 1.0$ $\lambda_0 = 1.0$ $\alpha_0 = 1.0$ $\beta_0 = 0.0$	Log-cost modeled with Normal-Inverse-Gamma prior. Costs truncated to: $0.1 \leq \text{cost} \leq B/10$

Table E.1: Prior distributions and posterior sampling procedures for cost model.

Priors for PS-Max. We assume that the rewards follow a Gaussian distribution, with initial parameter values specified in Table E.2.

Likelihood Model	Prior Parameters	Notes
Gaussian	$\mu_0 = 1.0$ $\lambda_0 = 1.0$ $\alpha_0 = 1.0$ $\beta_0 = 0.0$	Normal-Inverse-Gamma prior. Posterior updates: • $\sigma^2 \sim \text{InvGamma}(\alpha_n, \beta_n)$ • $\theta \sim \mathcal{N}(\mu_n, \sigma^2/\lambda_n)$

Table E.2: Prior distributions and posterior sampling procedures for reward model.

Priors for PS-PFN$_{mixed}$. Our methodology allows using different priors for each arm and benchmark. We make use of this advantage for PS-PFN$_{mixed}$, as shown in Table E.3. We note that the choice of priors is critical, since they should match expected real-world data while simultaneously impacting the exploration behavior of the agent. For example, choosing a *curved* prior instead of a *semi-flat* or *flat* one for the same arm leads to increased exploration of that arm (independently of whether this is the optimal choice for this arm). This implies that arms with a *curved* prior will be sampled more often regardless of the true trajectory type. If such arms are consistently optimal,

overall performance will improve. However, from another perspective, the possibility of prioritizing arms that are more likely to be optimal can be important. For instance, in TabRepoRaw, XGBoost has flat trajectories while it outperforms other methods more frequently. As a result, assigning a *flat* prior to XGBoost reduces overall performance. For our evaluation, we selected the prior that most closely matched the trajectory shape on the holdout datasets, as measured by the root mean square error (RMSE), as seen in Figure E.1 and E.2, but we emphasize that investigating this trade-off is a research direction for future work.

Benchmark	Model Class (Arm)	PFN Type
Complex	XTab	semi-flat
	FLAML	semi-flat
	RealMLP	flat
	TabForestPFN	semi-flat
	TabPFN_v2	semi-flat
TabRepoRaw	CatBoost	semi-flat
	ExtraTrees	semi-flat
	LightGBM	curved
	NeuralNet(FastAI)	curved
	NeuralNet(Torch)	curved
	RandomForest	semi-flat
	XGBoost	flat
YaHPOGym	AKNN	curved
	GLMNet	semi-flat
	RPart	semi-flat
	Ranger	curved
	SVM	curved
	XGBoost	curved

Table E.3: PFNs types for each arm across different benchmarks.

F Results in details

We compare the performance of PS-PFN under different priors by showing the ranking plot in Figure F.1 and the normalized loss (regret) over time in Figure F.2.

In Figure F.3, we show the normalized loss or regret over time. Figure F.4 displays the regret heatmap for each dataset in the Complex benchmark. Similarly, Figures F.5 and F.6 present the regret heatmaps for the TabRepoRaw and YaHPOGym benchmarks, respectively.

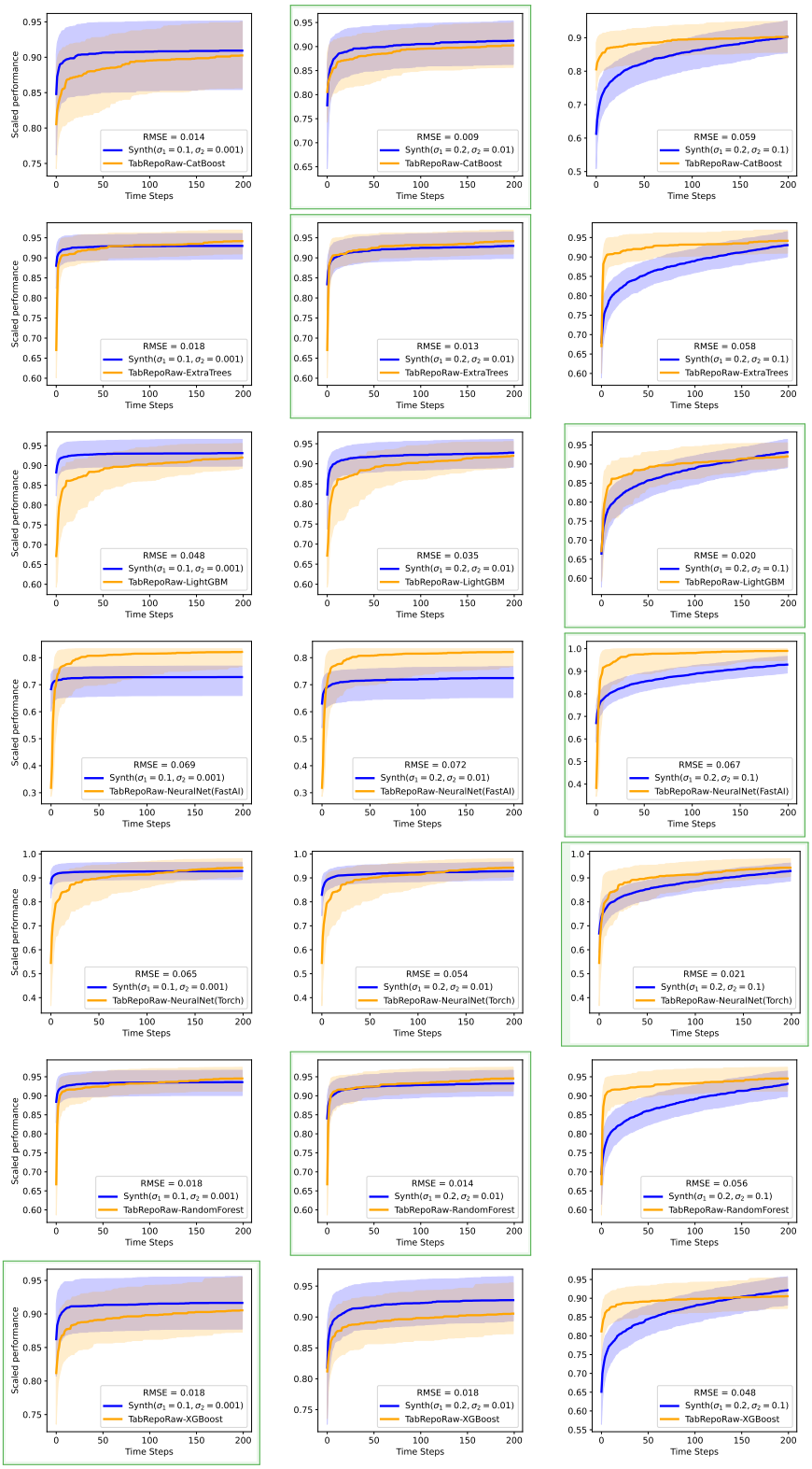


Figure E.1: Priors for TabRepoRaw

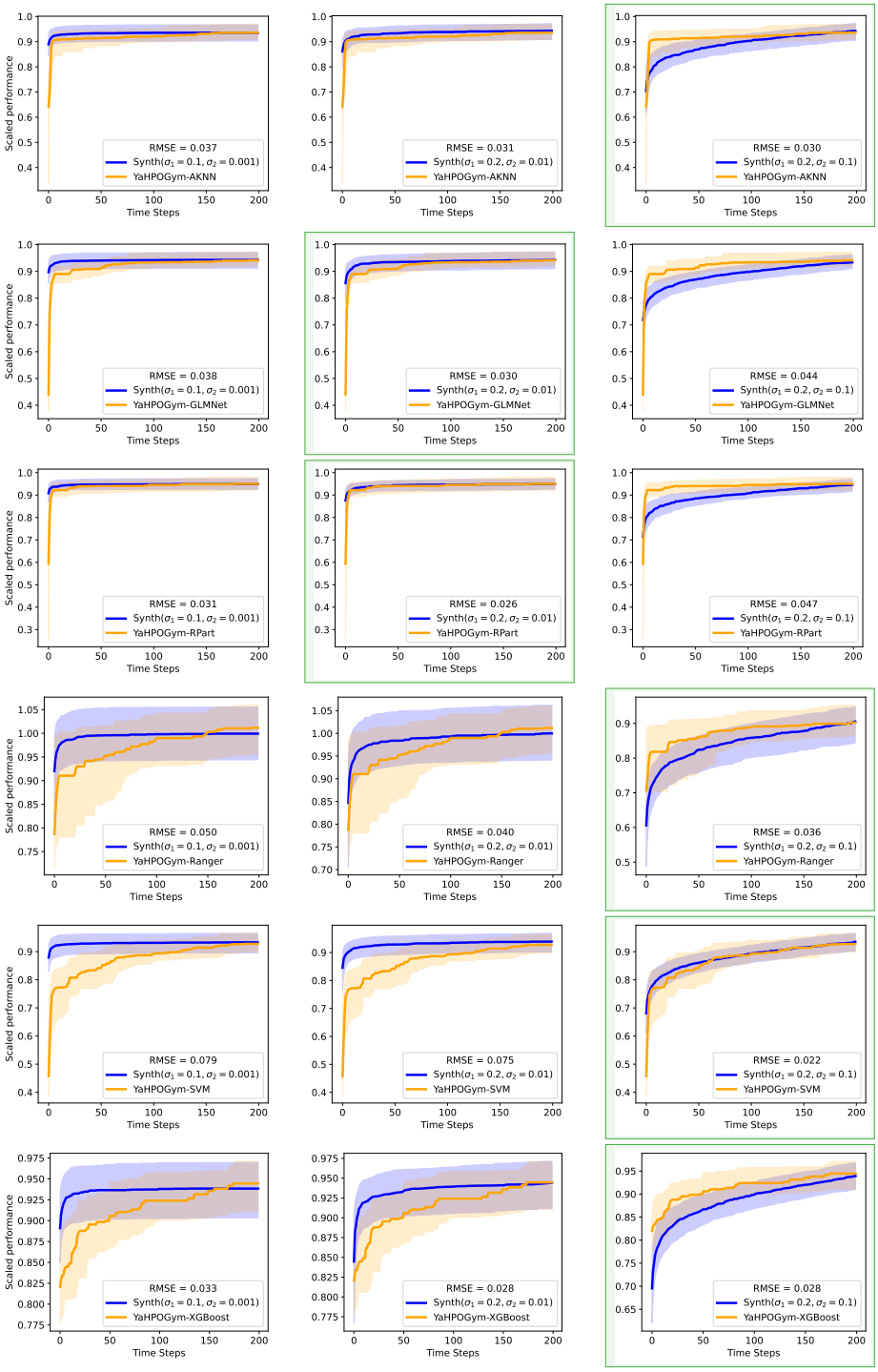


Figure E.2: Priors for YaHPOGym

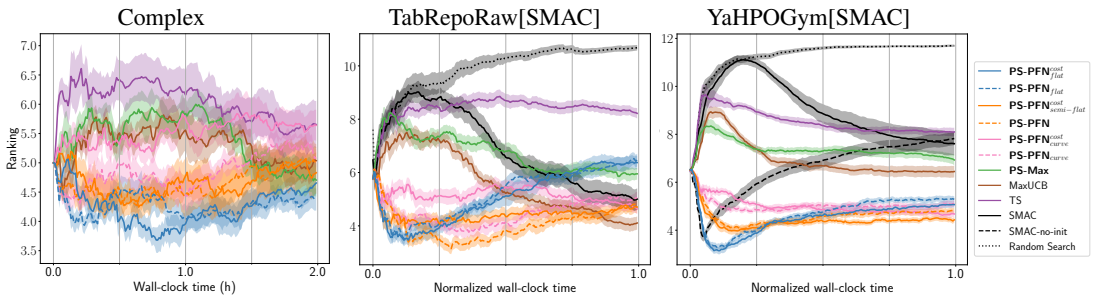


Figure F.1: Ranking of algorithms on different benchmarks, lower is better. *SMAC* and *random search* perform *combined CASH* across the joint space.

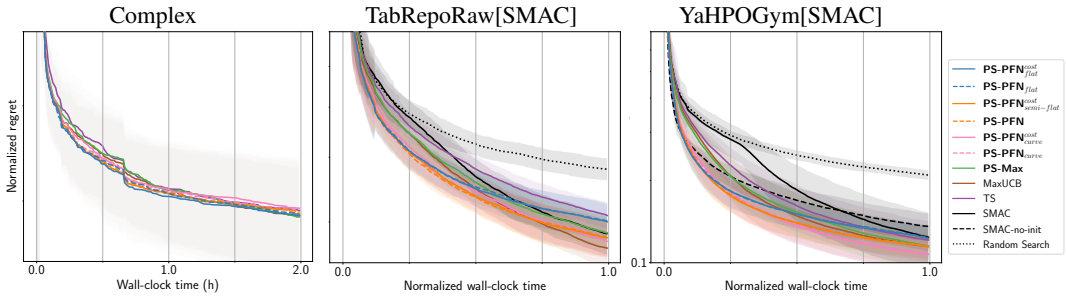


Figure F.2: Regret of algorithms on different benchmarks, lower is better. *SMAC* and *random search* perform *combined CASH* across the joint space.

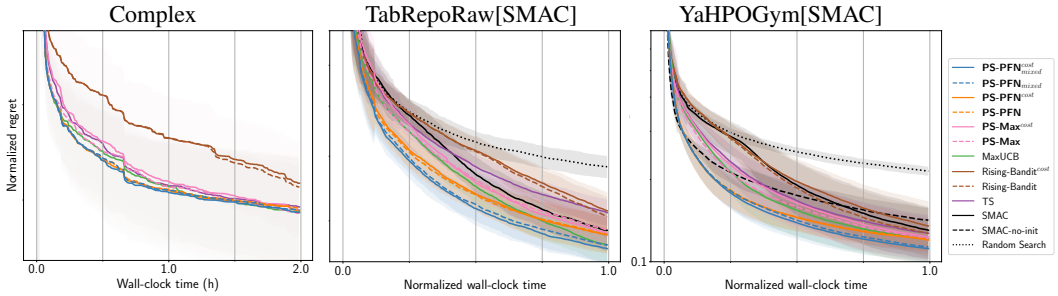


Figure F.3: Regret of algorithms on different benchmarks, lower is better. *SMAC* and *random search* perform *combined CASH* across the joint space.

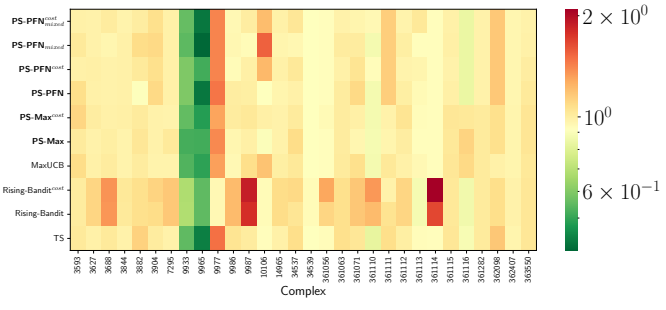


Figure F.4: Heatmap of regret of algorithms on different datasets of Complex benchmark, lower (green) is better.

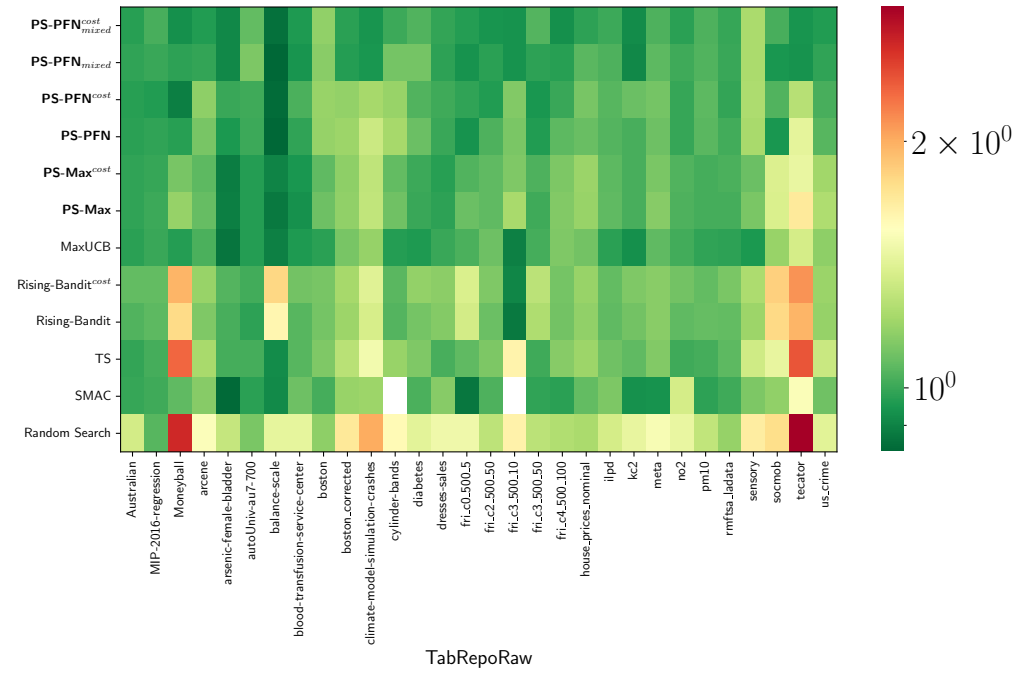


Figure F.5: Heatmap of regret of algorithms on different datasets of TabRepoRaw benchmark, lower (green) is better.

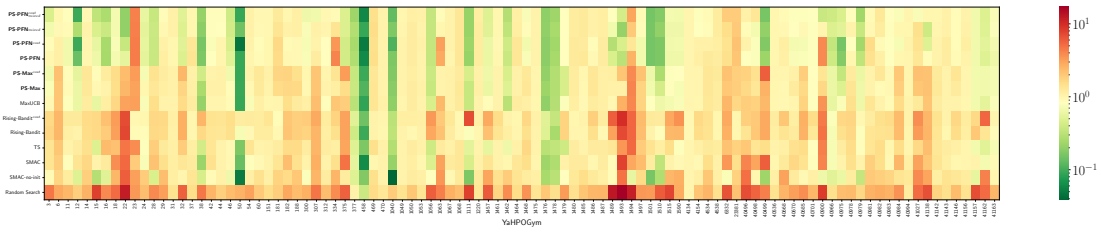


Figure F.6: Heatmap of regret of algorithms on different datasets YaHPOGym benchmark, lower (green) is better.