SIMPLE META-LEARNING FOR HPO AND CASH USING PERFORMANCE REPRESENTATIONS

Anonymous authors
Paper under double-blind review

ABSTRACT

Bayesian optimization for Hyperparameter Optimization (HPO) and Combined Algorithm Selection and Hyperparameter Optimization (CASH) often relies on explicit hyperparameter encodings as surrogate inputs. In high-dimensional search spaces, however, such representations can make it difficult for standard surrogates to learn useful performance models. We instead study an alternative meta-learning representation that describes each configuration directly as its historical performance across prior tasks. This representation can be paired directly with standard surrogate models. Our scope is the iterative selection component of automated machine learning, deciding which configuration to evaluate next from a fixed candidate set. We further apply this representation to maximal marginal relevance, a diversity-based re-ranking criterion from recommender systems, as a posterior-free acquisition rule. For CASH on TabArena, a benchmark for tabular data, the performance-based representation paired with a linear Bayesian ridge regression model outperforms dedicated meta-learning baselines, achieving the same tuned performance in roughly half the number of trials. For single-model HPO, it remains a competitive second. Across both benchmarks this method closely matches or improves on regret at a fraction of the complexity and cost, requiring no pre-training or tuning and scaling independently of search space dimensionality.

1 INTRODUCTION

Automated machine learning (AutoML) aims to remove the trial-and-error of selecting an algorithm and tuning its hyperparameters for a given task. The dominant paradigm casts this as hyperparameter optimization (HPO), or, when the algorithm itself is a decision variable, as Combined Algorithm Selection and Hyperparameter Optimization (CASH) (Thornton et al., 2013). Both are typically solved with Bayesian Optimization (BO), in which a surrogate model predicts the performance of a candidate configuration and an acquisition function selects what to evaluate next (Brochu et al., 2010). When historical results on related tasks are available, meta-learning can transfer this information to improve optimization on a new task (Brazdil et al., 2022). Such transfer has been realised through multi-task and transfer surrogates that share information across related optimization problems (Swersky et al., 2013; Wistuba et al., 2018; Feurer et al., 2018; Wistuba & Grabocka, 2021; Jomaa et al., 2021; Wang et al., 2024), or more recent meta-trained or in-context models that learn from collections of prior tasks end-to-end (Wei et al., 2019; Ruhkopf et al., 2023; Arango et al., 2024).

Despite their differences, most of these methods share the same configuration representation: an encoding of the hyperparameter vector, mapped to performance prediction. In CASH, this representation often concatenates the hyperparameter spaces of every candidate algorithm together with an algorithm identifier, leading to a high-dimensional conditional space in which most input dimensions are inactive for any given configuration (Thornton et al., 2013; Feurer et al., 2015b; Lindauer et al., 2022). Thus, learning useful interactions with such an input space is challenging.

An alternative line of work, inspired by collaborative filtering (Koren et al., 2009), represents configurations by their historical performance in prior tasks to tackle the AutoML problem (Fusi et al., 2018; Yang et al., 2019). This view aligns naturally with recommender systems, where tasks play the role of users, configurations of items, and validation scores of ratings. Prior matrix-based meth-

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

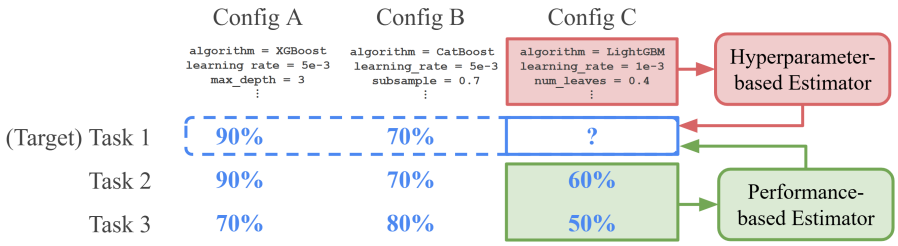

Figure 1: **Hyperparameter-based vs. performance-based configuration representations.** To estimate performance on a target task, standard BO represents a configuration by its hyperparameter vector λ (top) and possibly uses results on prior tasks for meta-learning, whereas our approach uses the historical performance vector P_λ across prior tasks directly as input representation (bottom). Our approach can be interpreted as learning to describe the target task in terms of prior tasks, for example as weighted sum of prior tasks in the case of a linear estimator.

ods typically exploit this structure through compressed or latent representations derived from the task–configuration performance matrix (Fusi et al., 2018; Yang et al., 2019).

We instead ask whether the raw historical performance vector itself can serve directly as a configuration representation for standard surrogate models. Figure 1 illustrates this contrast between the standard hyperparameter-based view and our performance-based representation. Additionally, we find that historical performance vectors also support a useful measure for representing diversity of configurations. In combination with maximal marginal relevance (MMR), a classical re-ranking criterion from information retrieval and recommender systems (Carbonell & Goldstein, 1998), and using historical mean performance as the relevance, this yields MeanMMR, a model-free acquisition rule that can stand in for uncertainty-driven exploration normally seen in Bayesian optimisation.

We benchmark these methods on TabArena (Erickson et al., 2026), a recent benchmark that covers diverse algorithms with randomly sampled hyperparameter configurations each across carefully curated realistic tabular classification and regression tasks, yielding a high-dimensional CASH search space. The performance-based representation paired with Bayesian ridge regression outperforms dedicated meta-learning baselines on CASH and remains a competitive second on HPO, at a fraction of their complexity and cost. Surprisingly, our simple model-free MeanMMR baseline also obtains lower average regret than these baselines on CASH.

Scope. We study the selection loop of AutoML, not a complete AutoML framework. Systems such as AutoGluon (Erickson et al., 2020) and auto-sklearn (Feurer et al., 2022) add features such as runtime estimation, cost-aware budget allocation and ensembling, which would confound a comparison that isolates the configuration representation, so we do not compare against them. Our representation is a component such frameworks could adopt for their selection step.

To summarize, our contributions are:

- We propose to use the raw historical performance vectors directly as configuration representations in BO, offering a simple route to meta-learning that allows standard surrogate models to meta-learn from previous tasks without specialized meta-learning procedures.
- We introduce MMR as a diversity-based, posterior-free acquisition function, and show that combined with historical mean performance it forms MeanMMR, a strong model-free meta-learning baseline.
- We benchmark popular surrogate models crossed with both representations on the TabArena benchmark, isolating the effect of the representation, and find that the performance representation makes a simple Bayesian ridge surrogate competitive with or better than dedicated meta-learning baselines at a fraction of their cost.

2 Related Work

Table 1 provides an overview of related work and the positioning of our work.

Hyperparameter Optimization and CASH. HPO aims to find the hyperparameter configuration of a learning algorithm that maximizes a chosen performance metric on a target dataset (Hutter et al., 2019). Thornton et al. (2013) extend HPO to the case where the algorithm itself is also a decision variable, Combined Algorithm Selection and Hyperparameter Optimization (CASH). As the number of hyperparameters grows, exhaustive enumeration becomes infeasible, and random search becomes a simple and widely used baseline (Bergstra & Bengio, 2012; Hutter et al., 2019).

To handle large search spaces, practical HPO and CASH methods treat configuration search as a sequential allocation problem, deciding which candidate to evaluate next under a limited evaluation budget. Hutter et al. (2011) introduce Sequential Model-Based Optimization (SMBO), which fits a surrogate model to observed configuration performances and uses an acquisition function to select the next configuration. Bayesian Optimization (BO) (Snoek et al., 2012) is a common probabilistic instance of this framework, using the surrogate's predictive distribution to trade off exploiting promising configurations and exploring uncertain regions (Brochu et al., 2010). Modern AutoML systems typically operate on explicit hyperparameter-space encodings of candidate configurations (Lindauer et al., 2022; Falkner et al., 2018; Thornton et al., 2013; Kotthoff et al., 2017; Feurer et al., 2015a; 2022).

The CASH search space is structurally more challenging than ordinary HPO, concatenating hyperparameters across every candidate algorithm into a high-dimensional, heavily conditional space (Feurer et al., 2015a; Lindauer et al., 2022). Prior work also finds that only a small subset of hyperparameters tends to dominate performance variation (Hutter et al., 2014; Van Rijn & Hutter, 2018; Probst et al., 2019), so a surrogate using an explicit hyperparameter encoding must infer the relevant dimensions and interactions from few target-task evaluations.

Algorithm Selection. A related problem setting is algorithm selection (AS) from a finite portfolio of algorithms, without explicitly considering any hyperparameter values (Brazdil et al., 2022). The simplest strategy ranks candidates by their average performance across prior tasks. Active Testing (Leite et al., 2012) makes this adaptive: it selects which candidate to evaluate next on the target task by estimating, from cross-task performance similarity, which evaluation is most informative for identifying the best candidate. These methods rely entirely on historical performance rather than a hyperparameter-space surrogate, similar to our setting, but use fixed or heuristic tournament-style ordering without Bayesian optimisation's uncertainty-driven exploration. We instead reframe HPO and CASH as the algorithm selection problem setting, while retaining Bayesian optimisation techniques.

Meta-Learning and Transfer for Bayesian Optimization. When historical results on related tasks are available, Bayesian optimization can transfer information to a new task (Brazdil et al., 2022). Swersky et al. (2013) introduce multi-task BO, which couples related problems through a joint task kernel. Ranking-weighted ensembles such as RGPE (Feurer et al., 2018) and two-stage transfer surrogates such as TST-R (Wistuba et al., 2018) combine per-task Gaussian process models using cross-task ranking agreement. ABLR (Perrone et al., 2018) meta-trains a shared hyperparameter feature embedding network with a per-task Bayesian linear head. Further related work spans deep-kernel transfer, pre-trained Gaussian processes, and in-context meta-models (Wistuba & Grabocka, 2021; Jomaa et al., 2021; Wang et al., 2024; Wei et al., 2019; Arango et al., 2024; Ruhkopf et al., 2023). These methods share two traits: a hyperparameter-encoded configuration input, and a need for specialized meta-learning machinery. We instead move the meta-learning into the input representation, so that a standard surrogate suffices.

Collaborative Filtering and Matrix-Based AutoML. A parallel line of work, inspired by collaborative filtering, observes that a task-configuration performance matrix mimics the user-item ratings matrix used in recommender systems. Tasks act as users, configurations as items, and validation scores as ratings (Koren et al., 2009; Yang et al., 2019; Fusi et al., 2018). Existing matrix-based methods typically compress this matrix into latent representations. Yang et al. (2019) introduce OBOE, which constructs compact configuration representations using low-rank matrix factorization and PCA-based embeddings. Fusi et al. (2018) introduce PMF, which learns latent configuration representations by gradient-based optimization and naturally handles sparse performance matrices. Two-Tower models add deep learning to factorization, generating separate task and configuration embeddings via multilayer perceptrons before scoring their compatibility (Huang et al., 2013; Cov-

Table 1: Positioning of our approach against representative methods from four lines of related work. We combine the performance-matrix representation of algorithm selection with the active surrogate-and-EI loop of Bayesian optimization, using the raw performance vector and any standard surrogate, without bespoke meta-training or latent compression. Abbreviations: BO, Bayesian optimization; AS, algorithm selection; CF, collaborative filtering.

Method	Line of work	Config. representation	Surrogate	Selection
SMAC3 (Lindauer et al., 2022)	BO	Hyperparameter	Random forest	EI
GP-BO (Snoek et al., 2012)	BO	Hyperparameter	Gaussian process	EI
Average ranking (Brazdil et al., 2022)	AS	Performance	None (ranking)	Static
Active Testing (Leite et al., 2012)	AS	Performance	None (matrix)	Active
Multi-task GP (Swersky et al., 2013)	Meta-learning BO	Hyperparameter	Multi-task GP	EI
RGPE (Feurer et al., 2018)	Meta-learning BO	Hyperparameter	GP ensemble	EI
TST-R (Wistuba et al., 2018)	Meta-learning BO	Hyperparameter	GP ensemble	EI
ABLR (Perrone et al., 2018)	Meta-learning BO	Hyperparameter	NN + Bayesian linear	EI
OBOE (Yang et al., 2019)	CF / matrix	Performance (PCA)	Linear	Custom
PMF (Fusi et al., 2018)	CF / matrix	Performance (latent)	Gaussian process	EI
Two-Tower (Huang et al., 2013)	CF / matrix	Hyper. / perf. (latent)	Two-tower network	Greedy / MMR
Ours	**AS + BO**	**Performance**	Any regression model	EI / MMR

ington et al., 2016; Yi et al., 2019; Raza et al., 2026). In contrast to all of these, we use the observed historical performance vector directly as a surrogate input, without an additional compression or factorization step. Our linear model is related to OBOE, and we compare Two-Tower variants embedding either hyperparameters or performance directly (see Appendix B.10).

2.1 Diversity-Based Acquisition and MMR

Beyond the surrogate model, our work also revisits the acquisition function used to select the next configuration. A common BO acquisition function is Expected Improvement (EI) (Jones et al., 1998), which uses the surrogate's predictive posterior to balance promising predicted performance against uncertainty (see Appendix B.5 for a short primer). When the surrogate does not provide a calibrated predictive posterior, standard uncertainty-driven rules such as EI do not directly apply. Prior matrix-based AutoML methods face the same issue: OBOE's bilinear low-rank surrogate (Yang et al., 2019) is not posterior-providing, so it replaces EI with a custom procedure, motivating more general posterior-free alternatives.

Maximal marginal relevance (MMR) was originally introduced for document retrieval and summarization and has since been used as a diversification strategy in recommender and retrieval-oriented settings (Carbonell & Goldstein, 1998; Vargas & Castells, 2011; Di Noia et al., 2017; Chen et al., 2021; Do et al., 2025; Korikov & Sanner, 2026). Related diversity-aware ideas have also appeared in ensemble-oriented CASH and diversified batch Bayesian optimization (Shen et al., 2022; Nava et al., 2022). However, to the best of our knowledge, we are the first to formulate MMR as acquisition rule for diversity-aware exploration in standard HPO and CASH, providing a simple posterior-free alternative to uncertainty-driven acquisition.

3 Performance-Based Representations for Simple Meta-Learning

3.1 Problem Setup

We study HPO and CASH as sequential configuration selection over a finite candidate set Λ. Each $\lambda \in \Lambda$ denotes a fully specified candidate that can be evaluated on a learning task t, yielding a performance score $\mathcal{P}(t, \lambda)$ where larger values indicate better performance. In the HPO setting, Λ contains configurations of a single algorithm; in the CASH setting, Λ spans a joint conditional configuration space over all candidate algorithms and their hyperparameters, represented through a one-hot algorithm indicator together with the full hyperparameter encoding. For a target task t, an optimization method iteratively selects an unevaluated $\lambda \in \Lambda$, observes $\mathcal{P}(t, \lambda)$, and updates a surrogate estimator $\hat{\mathcal{P}}$ for unobserved configurations. We restrict all methods to the same Λ and

instantiate each surrogate model family with two alternative inputs x_λ: the standard hyperparameter-based representation and a performance-based representation built from the historical performance across prior tasks, allowing simple meta-learning.

3.2 Performance-Based Representation for Meta-Learning

We propose a performance-based representation that describes a configuration by how it performed on previous tasks, instead of describing a configuration in terms of hyperparameter values. Let T_{train} be the set of prior tasks available for meta-learning. We collect the corresponding scores in a historical performance matrix $P \in \mathbb{R}^{|T_{\text{train}}| \times |\Lambda|}$, where each entry is $P_{t,\lambda} = \mathcal{P}(t, \lambda)$ for $t \in T_{\text{train}}$ and $\lambda \in \Lambda$. Each column of P corresponds to one configuration and each row corresponds to one prior task. The column $P_\lambda := P_{:,\lambda}$ is therefore the historical performance vector of configuration λ, and we use this column as the estimator input, $x_\lambda^{\text{perf}} = P_\lambda$. Configurations are thus compared by their empirical behaviour across previous tasks rather than by their hyperparameter values. This moves meta-learning into the input representation itself, allowing any standard surrogate that accepts vector inputs to exploit previous tasks by replacing x_λ^{hyper} with x_λ^{perf}, without changing the surrogate architecture, kernel, or training objective.

3.3 Surrogate Models for Performance Based Representations

We focus on the surrogate central to our contribution, Bayesian ridge regression, which to the best of our knowledge we are the first to study for HPO and CASH with the proposed performance-based representation. We additionally study a Two-Tower collaborative-filtering network as a deep meta-learning comparison; as it is not a central contribution, its description and results are deferred to Appendix B.10.

Linear (Bayesian ridge regression). Since the performance-based representation already encodes how each configuration behaves across prior tasks, we use Bayesian ridge regression (MacKay, 1992; Tipping, 2001) as a low-capacity surrogate that remains stable with few target-task observations and provides the predictive posterior needed for EI (Jones et al., 1998). Specifically, we use a linear-Gaussian model with a zero-mean isotropic Gaussian prior on the weights and Gamma hyperpriors on the noise and weight precisions. Training uses iterative empirical Bayes/evidence-maximization updates of these precisions rather than variational inference; conditional on the precisions, the weight posterior is Gaussian in closed form. When the input is x_λ^{perf}, the coefficients can be interpreted as weights over prior tasks in T_{train}, so target-task adaptation learns which previous tasks are informative for the current one. Implementation details are given in Appendix C.1, and its relation to OBOE is discussed in Appendices B.4 and C.2.

3.4 The maximal marginal relevance (MMR) Acquisition and MeanMMR Baseline

To support surrogates without a calibrated predictive posterior, we adapt MMR (Carbonell & Goldstein, 1998) as a diversity-based acquisition function that selects configurations with high predicted relevance while penalizing similarity to configurations already evaluated on the target task. Let Λ_{obs} be the set of configurations already evaluated on the target task. MMR selects the next configuration by maximizing:

$$\text{MMR}(\lambda) = \alpha \cdot \hat{\mathcal{P}}(t, \lambda) - (1 - \alpha) \cdot \max_{\lambda' \in \Lambda_{\text{obs}}} \cos(z_\lambda, z_{\lambda'}) \tag{1}$$

where $\hat{\mathcal{P}}(t, \lambda)$ is the relevance score, z_λ is the representation used to compute similarity, and α controls the relevance-diversity tradeoff (larger α places more emphasis on relevance, while smaller α encourages greater novelty relative to previously evaluated configurations).

Because MMR only requires a relevance score and a similarity, we can drop the surrogate altogether. MeanMMR replaces the relevance score $\hat{\mathcal{P}}(t, \lambda)$ in Equation 1 with the historical mean performance of λ across T_{train}, and computes similarity directly on the performance vectors P_λ (full expression in Appendix B.5). It turned out to be a surprisingly strong baseline.

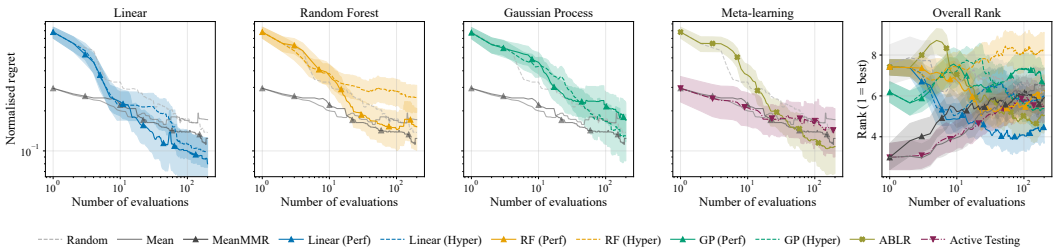

Figure 2: Test normalized regret per model family and overall rank vs. number of evaluations on the CASH benchmark (TabArena). Methods introduced in this work (the Perf variants and MeanMMR) are marked with an upward-pointing triangle (▲). Shaded bands are 95% bootstrap CIs across tasks.

4 Experimental Setup

Baselines. We compare results against dedicated meta-learning methods: the hyperparameter-based MTGP (Swersky et al., 2013), RGPE (Feurer et al., 2018) and ABLR (Perrone et al., 2018), as well as Active Testing (Leite et al., 2012) designed for algorithm selection. These dedicated baselines are included where our compute budget allowed, as the GP-based methods (MTGP, RGPE) do not stay computationally feasible on the high-dimensional CASH benchmark, and MTGP required subsampling modification to apply to HPO (see Appendix B.9 for computational cost analysis).

In addition, we include three more reference baselines. Random search (Bergstra & Bengio, 2012) uses no prior information and suggests configurations in random order. Mean ranking orders configurations by their average performance in P, one of the simplest forms of meta-learning and a well-studied baseline for both HPO and CASH (Brazdil et al., 2022). MeanMMR is the model-free acquisition baseline defined in Section 3.4.

Our models. We compare directly the hyperparameter-based and performance-based representations introduced in Section 3 on three surrogate models: Bayesian ridge regression, the ubiquitous Gaussian process regressor (Balandat et al., 2020), with automatic relevance determination (ARD) enabled to assign an independent lengthscale per input dimension (implementation details in Appendix B.6), and a Random Forest inspired by SMAC (Lindauer et al., 2022). Throughout the experiments, we use the suffixes Hyper and Perf to denote variants using hyperparameter-based and performance-based representations, respectively. Within each surrogate family, the two variants differ only in whether the surrogate receives x_λ^{hyper} or x_λ^{perf}. We use explicit hyperparameter encodings for the hyperparameter-based variants and the full historical performance vector P_λ for the performance-based variants; implementation details are given in Appendix B.4.

Dataset. All experiments use the TabArena benchmark (Erickson et al., 2026) (see Appendix E), which suits our study by combining curated tabular classification and regression tasks with leaderboard results for many modern algorithms and numerous sampled configurations per algorithm, allowing us to construct both CASH and HPO benchmarks from a shared task-configuration performance matrix. Specifically, we construct two finite search spaces: a CASH benchmark over a multi-algorithm configuration set (see Appendix E.1) and an HPO benchmark over an XGBoost-only subset (see Appendix E.2). The candidates in each benchmark define Λ and, on prior tasks, the performance matrix P used by the performance-based methods. Detailed benchmark construction is described in Appendix B.1.

Metrics and Validation Scheme. We use five-fold cross-validation at the task level. Meta-training tasks define the performance matrix and any prior-task training, while held-out tasks are optimized independently as target HPO or CASH problems. Search uses validation-fold feedback only, and final regret is computed on separate test folds for the validation-selected configuration to avoid skewed results due to adaptive overfitting (Schröder et al., 2025). Further details are given in Appendix B.2.

Table 2: Average normalized regret on the test split ($\times 100$) and average rank across tasks, over 200 evaluations amongst the discrete set of configurations for the CASH TabArena benchmark and the HPO TabArena XGBoost-only benchmark. **Bold**: best per metric per benchmark. Underline: best per model family per metric. *: significantly better than others within the family (Wilcoxon signed-rank, $p < 0.05$). Time/task: single-threaded per-task time spent fitting the surrogate and evaluating the acquisition function over the 200 evaluations. ‡: excludes the required one-time pretraining phase shared across tasks (ABLR 3.4m on CASH, 27s on HPO; RGPE 2.0m on HPO).

		CASH			HPO		
Model	**Input**	Regret ↓	Rank ↓	Time/task	Regret ↓	Rank ↓	Time/task
Linear	Perf.	**12.19***	**4.34***	2.7s	9.84*	6.14*	1.1s
	Hyper.	14.59	5.95	24s	12.79	7.04	0.7s
Random Forest	Perf.	18.53*	5.83*	2.0m	11.29	6.51	1.8m
	Hyper.	28.59	8.06	2.7m	13.46	6.59	53s
Gaussian Process	Perf.	23.73	6.99	7.3m	13.40	6.87	1.5m
	Hyper.	20.25	5.93	1.8h	13.61	6.68	7.9m
ABLR	Hyper.	15.47	5.22	1.1s‡	**9.02**	**5.97**	0.6s‡
Active-Testing	Perf.	16.54	5.37	0.9s	16.63	7.56	0.7s
MTGP	Hyper.	—	—	—	13.52	6.98	10s
RGPE	Hyper.	—	—	—	10.57	6.18	9.3m‡
Random	—	19.45	6.61	0.5s	16.86	7.63	0.3s
Mean	Perf.	17.56	5.93	0.7s	25.03	9.37	0.3s
MeanMMR	Perf.	14.60	5.77	6.6s	14.92	7.47	0.5s

5 RESULTS

On the high-dimensional CASH benchmark, the performance-based representation paired with a simple Bayesian ridge surrogate (Linear-Perf) attains the best average regret and rank overall, both in the regret curves (Figure 2) and at the full budget of 200 evaluations (Table 2). Concretely, ABLR needs 149 evaluations to reach the regret Linear-Perf reaches in 73, close to a 2x reduction in trials needed. Linear-Perf is also the fastest of the surrogate models to fit, averaging under three seconds per task against roughly two hours for GP-Hyper on the same benchmark. On the lower-dimensional HPO benchmark, the ABLR method that meta-learns a hyperparameter embedding network performs best, with Linear-Perf following at a close second.

Comparing the performance representation against the usual hyperparameter representation shows statistically significant improvements for the linear and random forest surrogates in the CASH benchmark, but no statistically significant or consistent improvement for the Gaussian Process: the benefit of historical performance as representation is not observed as strongly across all surrogate types.

Rankings of the static baselines that do not fit a surrogate model differ between benchmarks. On the XGBoost HPO setting, all model-based methods consistently achieve better regret than random search after the first few iterations, with mean ranking the worst strategy overall (25.03 versus 16.86 for random search). In the CASH setting, the hyperparameter-based random forest does not clearly improve upon random search, and the mean ranking gives a very strong initial selection of models until being surpassed after ten evaluations, and still beats random search (regret 17.56 versus 19.45). Active testing benefits from the strong initialization on CASH, and is surpassed by Linear-Perf after roughly twenty evaluations. The MeanMMR baseline improves upon mean ranking, such that it beats random search on HPO, and is competitive with non-metalearning methods for roughly the first fifty evaluations, although surpassed earlier by strong meta learners. Surprisingly, MeanMMR outperforms the dedicated meta-learning methods on CASH in terms of average regret.

We measure how the methods depend on the number of tasks for which historical performance is known (Figure 3); as expected, average regret generally improves as the number of prior tasks increases. For CASH, Linear-Hyper is the strongest non-metalearning method whose average regret is not surpassed until at least 16 training tasks are used by Linear-Perf. In the HPO setting, ABLR

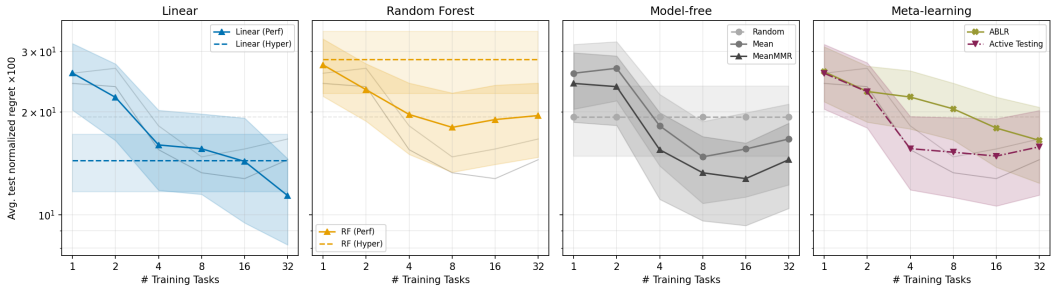

Figure 3: Average normalized regret over 200 configuration evaluations on the CASH benchmark vs. number of training tasks. Methods introduced in this work (the Perf variants and MeanMMR) are marked with an upward-pointing triangle (▲). Dashed horizontal lines show methods that do not make use of results on previous tasks. Shaded bands are 95% bootstrap CIs across tasks.

is immediately competitive with Linear-Hyper and offers the best regret once results for at least one prior training task are available. Methods using mean ranking, including active testing, appear to benefit initially but stop improving after more than 16 tasks are added.

6 Discussion & Limitations

The performance-based representation implements meta-learning well specifically in combination with a linear surrogate model, indicating that a configuration's performance on the target task fits well to a linear combination of its performances on prior tasks: Linear-Perf effectively learns to represent the target task as a weighted average of prior tasks. That aligns with ideas behind related work such as the weighted ensembling over task models of RGPE. We performed an ablation of RGPE that replaces the base-GP predictions with the true performance columns of the prior tasks (RGPE-Oracle), which is still strong on HPO but not as competitive with Linear-Perf on CASH (see Appendix B.9). Freely determining weights per prior task via regression may be more effective and expressive than RGPE's non-negative heuristic ranking agreement; a fixed-precision ablation confirms this is not simply due to adaptive precision estimation (see Appendix C.1). Consistent with this, the weights learned by Bayesian ridge regression tend to be both positive and negative, with roughly half negative on CASH (see Appendix C.3), likely necessary for a good fit on the high-dimensional problem.

Rankings of models differ between the CASH and HPO settings. In the lower-dimensional HPO case, hyperparameter methods all perform better than random search by a larger margin, and the hyperparameter embedding-based ABLR method is best overall. This can be explained by the manageable dimensionality, with only 11 hyperparameters whose effects to learn, versus 475 values after encoding for the high-dimensional CASH problem. This reflects the curse of dimensionality: more data points are needed to obtain the same predictive performance, and since only a subset of hyperparameters is active per selected algorithm, relevant data points per hyperparameter are further reduced. Architectures that exploit this structure efficiently, e.g., separate encoders per algorithm or surrogates designed for conditional, tree-structured search spaces (Jenatton et al., 2017), might have performed better; not pursuing that direction is a limitation of our baselines, but consistent with our motivation that large-scale CASH invites unwanted engineering complexity. One might argue this large-scale task is an unfair comparison, but it also reflects a realistic problem: the choices data scientists face with the full breadth of state-of-the-art tabular models on benchmark leaderboards such as TabArena (Erickson et al., 2026).

The model-free baseline results also support that meta-learning helps more on CASH than on HPO. Within one algorithm's hyperparameter space the choice of configuration might depend more on the specific tasks, while across multiple algorithm types some might be clearly stronger or weaker. That might explain why historical performance gives larger advantages on CASH.

The strong initialisation of the mean best configuration on CASH, as used by active testing, could easily be applied to other methods as well, further strengthening for example Linear-Perf. This was not done for the base model set to keep the experiment controlled by only varying the input

representation, and the other meta-learning baselines did not explicitly include this initialisation step in their algorithm.

MeanMMR benefits from the strong mean signal, while not suffering from the low performance that mean ranking has on HPO. The exploration term based on historical performance diversity bridges part of the gap between static ranking methods and exploration-driven Bayesian optimisation. The low performance of mean ranking on HPO might therefore also be caused by it suggesting low-diversity configurations from a local optimum, suggesting portfolio methods might benefit from similar diversity measures. MMR also extends to surrogate models without a predictive posterior, such as the Two-Tower recommender model; we explored this variant but omit it from the main comparison as its results were competitive but not state-of-the-art (see Appendix B.10).

The biggest limitation of the performance-based methods is requiring a dense matrix with historical performance of every candidate configuration on every prior task. The cost of collecting the matrix scales with both dimensions, number of configurations and number of tasks. Given this initial cost, the method is most applicable to popular search spaces where a centralized matrix can be maintained, shared and reused many times for different applications. For niche domains and architectures where a search space is only used a handful of times, benefits of faster search likely do not outweigh the initial cost of collecting the matrix. Furthermore, unlike factorization methods such as PMF (Fusi et al., 2018), our method would require imputation to support sparse matrices.

Multi-task benchmarks give a more optimistic perspective on this data requirement. In our scalability experiment on CASH, roughly 16 prior tasks were needed before performance representation models outperformed alternatives, while the full TabArena leaderboard (Erickson et al., 2026) already contains results for 51 tasks. Matrices built from benchmark leaderboards are also likely to stay up to date, since authors of new methods contribute the required data whenever they report results on the benchmark.

Scaling of possible performance with respect to the number of configurations is clear: A performance-based method can at best suggest the optimal configuration included in the matrix, when the truly optimal configuration may in reality lie somewhere else in the continuous hyperparameter search space. Future work should examine the continuous case.

Because it requires no hyperparameter encoding at all, our representation also avoids having to define a single unified search space across heterogeneous algorithms, which is especially useful when aggregating results across many different model types, for example from platforms such as OpenML (Bischl et al., 2025).

7 Conclusion

In summary, using historical performance as a configuration representation is an effective and simple approach to meta-learning. Paired with a linear Bayesian ridge regression model, it outperforms dedicated meta-learning baselines on the high-dimensional CASH benchmark, reaching the same tuned performance in roughly half the number of trials, and remains a competitive second on single-model HPO. Simplicity is key: the well-performing Bayesian ridge regression only models linear relationships between tasks and requires no pre-training, hyperparameter preprocessing, or tuning. Using performance representations avoids complexity in combining search space encodings across models with different hyperparameters. For a completely model-free alternative, ranking by the mean and diversity of the performance vectors (MeanMMR) is stronger than pure average ranking and competitive with limited target-task observations.

AI use statement

We have not used generative AI tools for any of the tasks ICLR lists as requiring disclosure: generating synthetic data sets, developing theoretical models or conceptual frameworks, formulating mathematical claims, providing critical ingredients for proving mathematical claims, assisting in the writing of proofs, proposing or refining hypotheses, designing or providing feedback on research methodology or experiments, implementing methods, assisting with translation, cleaning and reformatting data sets, supporting qualitative and thematic data analysis, and interpreting results. Among the tasks for which disclosure is recommended, we used generative AI tools for sourcing and sum-

marising existing literature, for editing software code, limited to refactoring, visualisation scripts, and scripts that aggregate experiment outputs into tables and run the reported significance tests, for creating and modifying figures, for suggesting a structure for this paper, for LaTeX formatting support, and for proofreading to improve readability. We have reviewed all AI-assisted work: all reported results were produced and verified by the authors' own code, AI-suggested references were checked against the primary sources, and all AI-assisted text was read and edited by the authors. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

All reported experiments are simulated by lookups into the public TabArena leaderboard results (Erickson et al., 2026), so no base model has to be retrained to reproduce them. Appendix B.1 describes how the CASH and HPO search spaces are constructed from that data, Appendix E lists the exact base model set and the XGBoost search space, and Appendices B.2 and B.3 define the task-level cross-validation protocol, the validation-test split, and the regret normalization. Implementation details for every surrogate and baseline are given in Appendices B.4–B.10, with the final hyperparameters and random seeds collected in Table 3, and the hardware, software versions, and total compute are reported in Appendix B.11. The anonymized source code, including the configuration files that define each reported run, is provided as supplementary material.

REFERENCES

Sebastian Pineda Arango, Fabio Ferreira, Arlind Kadra, Frank Hutter, and Josif Grabocka. Quicktune: Quickly learning which pretrained model to finetune and how. In *The Twelfth International Conference on Learning Representations*, 2024.

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*, 2020.

James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of machine learning research*, 13(2), 2012.

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.

Pavel Brazdil, Jan N. Van Rijn, Carlos Soares, and Joaquin Vanschoren. *Metalearning: Applications to Automated Machine Learning and Data Mining*. Cognitive Technologies. Springer International Publishing, Cham, 2022. ISBN 978-3-030-67023-8 978-3-030-67024-5.

Eric Brochu, Vlad M. Cora, and Nando de Freitas. A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning, December 2010. arXiv:1012.2599 [cs].

Jaime Carbonell and Jade Goldstein. The use of mmr, diversity-based reranking for reordering documents and producing summaries. In *Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval*, pp. 335–336, 1998.

Wanyu Chen, Pengjie Ren, Fei Cai, Fei Sun, and Maarten De Rijke. Multi-interest diversification for end-to-end sequential recommendation. *ACM Transactions on Information Systems (TOIS)*, 40(1):1–30, 2021.

Paul Covington, Jay Adams, and Emre Sargin. Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM conference on recommender systems*, pp. 191–198, 2016.

Tommaso Di Noia, Jessica Rosati, Paolo Tomeo, and Eugenio Di Sciascio. Adaptive multi-attribute diversity for recommender systems. *Information Sciences*, 382:234–253, 2017.

Juseon Do, Jaesung Hwang, Jingun Kwon, Hidetaka Kamigaito, and Manabu Okumura. Considering length diversity in retrieval-augmented summarization. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pp. 2489–2500, 2025.

Nick Erickson, Jonas Mueller, Alexander Shirkov, Hang Zhang, Pedro Larroy, Mu Li, and Alexander Smola. AutoGluon-Tabular: Robust and Accurate AutoML for Structured Data, March 2020. arXiv:2003.06505 [stat].

Nick Erickson, Lennart Purucker, Andrej Tschalzev, David Holzmüller, Prateek Desai, David Salinas, and Frank Hutter. Tabarena: A living benchmark for machine learning on tabular data. *Advances in Neural Information Processing Systems*, 38, 2026.

Stefan Falkner, Aaron Klein, and Frank Hutter. Bohb: Robust and efficient hyperparameter optimization at scale. In *International conference on machine learning*, pp. 1437–1446. PMLR, 2018.

Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Springenberg, Manuel Blum, and Frank Hutter. Efficient and robust automated machine learning. *Advances in neural information processing systems*, 28, 2015a.

Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Tobias Springenberg, Manuel Blum, and Frank Hutter. Auto-sklearn: Efficient and robust automated machine learning. In *Proceedings of the 29th International Conference on Neural Information Processing Systems - Volume 2*, volume 2 of *NIPS'15*, pp. 2755–2763, Cambridge, MA, USA, December 2015b. MIT Press.

Matthias Feurer, Benjamin Letham, Frank Hutter, and Eytan Bakshy. Practical transfer learning for bayesian optimization. *arXiv preprint arXiv:1802.02219*, 2018.

Matthias Feurer, Katharina Eggensperger, Stefan Falkner, Marius Lindauer, and Frank Hutter. Auto-sklearn 2.0: Hands-free automl via meta-learning. *Journal of Machine Learning Research*, 23 (261):1–61, 2022.

Nicolo Fusi, Rishit Sheth, and Melih Elibol. Probabilistic matrix factorization for automated machine learning. *Advances in neural information processing systems*, 31, 2018.

Po-Sen Huang, Xiaodong He, Jianfeng Gao, Li Deng, Alex Acero, and Larry Heck. Learning deep structured semantic models for web search using clickthrough data. In *Proceedings of the 22nd ACM international conference on Information & Knowledge Management*, pp. 2333–2338, 2013.

Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *International conference on learning and intelligent optimization*, pp. 507–523. Springer, 2011.

Frank Hutter, Holger Hoos, and Kevin Leyton-Brown. An efficient approach for assessing hyperparameter importance. In *International conference on machine learning*, pp. 754–762. PMLR, 2014.

Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren. *Automated machine learning: methods, systems, challenges*. Springer, 2019.

Rodolphe Jenatton, Cédric Archambeau, Javier González, and Matthias Seeger. Bayesian optimization with tree-structured dependencies. In *Proceedings of the 34th International Conference on Machine Learning (ICML)*, volume 70 of *PMLR*, pp. 1655–1664, 2017.

Hadi Samer Jomaa, Sebastian Pineda Arango, Lars Schmidt-Thieme, and Josif Grabocka. Transfer learning for bayesian hpo with end-to-end landmark meta-features. In *Fifth Workshop on Meta-Learning at the Conference on Neural Information Processing Systems*, 2021.

Donald R Jones, Matthias Schonlau, and William J Welch. Efficient global optimization of expensive black-box functions. *Journal of Global optimization*, 13(4):455–492, 1998.

Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, 2009.

Anton Korikov and Scott Sanner. Rebol: Retrieval via bayesian optimization with batched llm relevance observations and query reformulation. *arXiv preprint arXiv:2603.20513*, 2026.

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.

Rui Leite, Pavel Brazdil, and Joaquin Vanschoren. Selecting Classification Algorithms with Active Testing. In Petra Perner (ed.), *Machine Learning and Data Mining in Pattern Recognition*, pp. 117–131, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg. ISBN 978-3-642-31537-4.

Marius Lindauer, Katharina Eggensperger, Matthias Feurer, André Biedenkapp, Difan Deng, Carolin Benjamins, Tim Ruhkopf, René Sass, and Frank Hutter. Smac3: A versatile bayesian optimization package for hyperparameter optimization. *Journal of Machine Learning Research*, 23 (54):1–9, 2022.

Marco Loog, Tom Viering, Alexander Mey, Jesse H Krijthe, and David MJ Tax. A brief prehistory of double descent. *Proceedings of the National Academy of Sciences*, 117(20):10625–10626, 2020.

David JC MacKay. Bayesian interpolation. *Neural computation*, 4(3):415–447, 1992.

Elvis Nava, Mojmir Mutny, and Andreas Krause. Diversified sampling for batched bayesian optimization with determinantal point processes. In *International Conference on Artificial Intelligence and Statistics*, pp. 7031–7054. PMLR, 2022.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Valerio Perrone, Rodolphe Jenatton, Matthias W Seeger, and Cedric Archambeau. Scalable Hyperparameter Transfer Learning. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

Philipp Probst, Anne-Laure Boulesteix, and Bernd Bischl. Tunability: Importance of hyperparameters of machine learning algorithms. *Journal of Machine Learning Research*, 20(53):1–32, 2019.

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

Shaina Raza, Mizanur Rahman, Safiullah Kamawal, Armin Toroghi, Ananya Raval, Farshad Navah, and Amirmohammad Kazemeini. A comprehensive review of recommender systems: Transitioning from theory to practice. *Computer Science Review*, 59:100849, 2026.

Tim Ruhkopf, Aditya Mohan, Difan Deng, Alexander Tornede, Frank Hutter, and Marius Lindauer. Masif: Meta-learned algorithm selection using implicit fidelity information. *Transactions on Machine Learning Research*, 2023.

Sietse Schröder, Mitra Baratchi, and Jan N. van Rijn. Overfitting in Combined Algorithm Selection and Hyperparameter Optimization. In *Advances in Intelligent Data Analysis XXIII: 23rd International Symposium on Intelligent Data Analysis, IDA 2025, Konstanz, Germany, May 7–9, 2025, Proceedings*, pp. 181–194, Berlin, Heidelberg, May 2025. Springer-Verlag. ISBN 978-3-031-91397-6.

Yu Shen, Yupeng Lu, Yang Li, Yaofeng Tu, Wentao Zhang, and Bin Cui. Divbo: Diversity-aware cash for ensemble learning. *Advances in Neural Information Processing Systems*, 35:2958–2971, 2022.

Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. Practical Bayesian Optimization of Machine Learning Algorithms, August 2012. arXiv:1206.2944 [stat].

Kevin Swersky, Jasper Snoek, and Ryan P Adams. Multi-task bayesian optimization. *Advances in neural information processing systems*, 26, 2013.

Chris Thornton, Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Auto-weka: Combined selection and hyperparameter optimization of classification algorithms. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 847–855, 2013.

Michael E Tipping. Sparse bayesian learning and the relevance vector machine. *Journal of machine learning research*, 1(Jun):211–244, 2001.

Jan N Van Rijn and Frank Hutter. Hyperparameter importance across datasets. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 2367–2376, 2018.

Saúl Vargas and Pablo Castells. Rank and relevance in novelty and diversity metrics for recommender systems. In *Proceedings of the fifth ACM conference on Recommender systems*, pp. 109–116, 2011.

Tom Viering and Marco Loog. The shape of learning curves: a review. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(6):7799–7819, 2022.

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

Ying Wei, Peilin Zhao, Huaxiu Yao, and Junzhou Huang. Transferable neural processes for hyperparameter optimization. *arXiv preprint arXiv:1909.03209*, 2019.

Martin Wistuba and Josif Grabocka. Few-shot bayesian optimization with deep kernel surrogates. In *International Conference on Learning Representations*, 2021.

Martin Wistuba, Nicolas Schilling, and Lars Schmidt-Thieme. Scalable gaussian process-based transfer surrogates for hyperparameter optimization. *Machine Learning*, 107(1):43–78, 2018.

Chengrun Yang, Yuji Akimoto, Dae Won Kim, and Madeleine Udell. Oboe: Collaborative filtering for automl model selection. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 1173–1183, 2019.

Xinyang Yi, Ji Yang, Lichan Hong, Derek Zhiyuan Cheng, Lukasz Heldt, Aditee Kumthekar, Zhe Zhao, Li Wei, and Ed Chi. Sampling-bias-corrected neural modeling for large corpus item recommendations. In *Proceedings of the 13th ACM conference on recommender systems*, pp. 269–277, 2019.

APPENDIX

This appendix provides the implementation and evaluation-protocol details, additional figures, and extended discussion referenced from the main paper. Reviewers are not required to read this material; anything critical to evaluating the paper's claims is contained in the main body.

A HPO BENCHMARK FIGURES

Plots showing the test normalized regret against number of evaluations or number of observed prior tasks on the HPO benchmark are shown in Figure 4 and Figure 5.

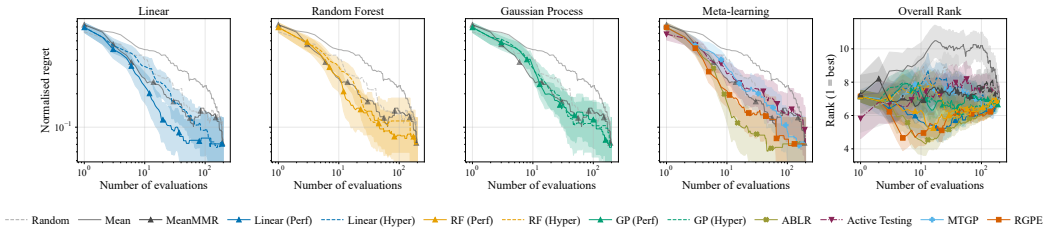

Figure 4: Test normalized regret per model family and overall rank vs. number of evaluations on the HPO benchmark (TabArena XGBoost-only). Methods introduced in this work (the Perf variants and MeanMMR) are marked with an upward-pointing triangle (▲). Shaded bands are 95% bootstrap CIs across tasks.

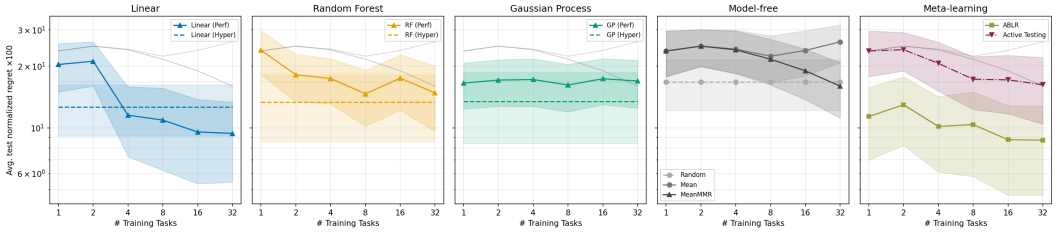

Figure 5: Average normalized regret over 200 model evaluations on the HPO benchmark vs. number of previous tasks observed, grouped by model family. Methods introduced in this work (the Perf variants and MeanMMR) are marked with an upward-pointing triangle (▲). Dashed horizontal lines show methods that do not make use of results on previous tasks. Shaded bands are 95% bootstrap CIs across tasks.

B EXPERIMENTAL SETUP DETAILS

This appendix provides the implementation details behind the experimental protocol, including benchmark construction, encoding choices, acquisition rules, and model settings.

B.1 BENCHMARK CONSTRUCTION

The public TabArena leaderboard dataset contains cross-validated results for evaluating each listed algorithm on each task. For most algorithms this includes not only the default hyperparameter setting, but also 200 randomly selected hyperparameter configurations. From these data we construct the performance matrix P used in our meta-learning models and representations. Using all base algorithms and tasks, the matrix contains 3623 configurations across 23 algorithms and 51 tasks (see Table 6). The algorithm set does not include large ensembles like AutoGluon, as they could have a disproportionate impact through their high performance, while our methods and evaluation framework do not penalize them for disproportionately high training cost. For the HPO benchmark, we use the 201 XGBoost configurations, consisting of one default configuration and 200 random

configurations (see Table 7). The choice of XGBoost is arbitrary; any competitive method with a sufficiently large list of configurations could have been used.

These 3623 configurations for CASH and 201 configurations for HPO fully define the search space Λ in each benchmark. Such a finite set is required for defining the matrix used by performance-based methods. The hyperparameter-based methods do not have that restriction in principle and could suggest configurations from the underlying continuous search space. However, using the same finite search space for all variants isolates the effect of changing the representation. Additionally, staying within this set makes large-scale experimentation possible by simulating configuration evaluations with lookups in the precomputed benchmark results.

B.2 Evaluation Protocol

We employ five-fold cross-validation at the task level, each fold resulting in a *meta-training set* of approximately 40 training tasks and 10 test tasks. The meta-training tasks are used for the performance matrix and representations, and for training the Two-Tower model, though not all methods make use of them. Within the meta-test set, HPO or CASH is run on each test task independently. When a model is updated on one target task, it is reverted to its state after training on the meta-training tasks before continuing with the next target task. Metrics are aggregated using the mean across all out-of-fold tasks, and confidence intervals are computed using bootstrap resampling across tasks.

The same procedure is used when plotting performance with respect to training tasks, with the addition of randomly subsampling tasks within each training fold. There might be strategies for selecting an optimal diverse subset of tasks, we choose randomly for simplicity. The random subsample is without replacement, and seeded such that it is independent across folds but consistent amongst methods.

B.3 Regret and Normalization

As HPO and CASH are iterative processes, we measure regret as a curve over the number of configurations that have been evaluated. As a summary statistic, regret is averaged over this curve. Often, regret is reported as the difference between the optimal value known for an objective function and the optimal value among the configurations selected so far by a search strategy. However, this does not take adaptive overfitting into account, where the selection of the *best so far* may not translate to other test sets (Schröder et al., 2025). To prevent adaptive overfitting from positively skewing the results, we split validation and test folds. The TabArena leaderboard data reports scores for individual folds per task; we set apart half of the folds to form the validation split and the other half as the test split. The validation split defines $\mathcal{P}_{\text{val}}$, which provides the matrix P, all target-task observations seen during search, and the selection of the *best so far* configuration. The held-out split defines $\mathcal{P}_{\text{test}}$, which is used only for final reporting. The final regret calculation therefore evaluates the test performance of the configuration selected by validation performance:

$$r_n(t) = \max_{\lambda \in \Lambda} \mathcal{P}_{\text{test}}(t, \lambda) - \mathcal{P}_{\text{test}}\left(t,\ \arg\max_{\lambda \in \Lambda_{\text{obs}}^{(n)}} \mathcal{P}_{\text{val}}(t, \lambda)\right) \tag{2}$$

where n indexes the number of evaluated models, $\mathcal{P}_{\text{test}}$ is performance on the test folds and $\mathcal{P}_{\text{val}}$ on the validation folds. In practice, we did observe some diverging of validation and test regret, shown in Section D).

Besides adaptive overfitting, we aim to eliminate another type of possible bias observed in related work: normalization leakage. As some tasks may have consistently higher or lower performance across configurations, it is common to report normalized regret. In the case of TabArena this is scaled such that the best score for a task is 1 and the median is 0. This is justified when comparing AutoML methods from an outside perspective, but normalized values should not be available during the search process, as they contain information about the optimal achievable score.

To prevent leakage but keep performance on similar scales across tasks, we use two types of normalization. During search, the performance matrix P and observed target-task values are scaled by the dummy score. The dummy score is the error achieved when constantly predicting the mean

for regression tasks or the majority class for classification, which we consider a zero-cost value in practice that all methods would have access to. When reporting normalized regret in graphs and tables, we use the aforementioned scaling that maps the best performance overall for a task to 1 and the median to 0.

Similarly, while the TabArena leaderboard imputes the few combinations where an algorithm is not capable of a specific task type with the score of a random forest, we score these combinations using the near zero-cost dummy score instead.

B.4 Representation Encoding

For the HPO benchmark, $\phi(\lambda)$ contains the hyperparameters of the single algorithm under consideration. For the CASH benchmark, where $\lambda = (A, h)$, it concatenates the hyperparameter lists of all algorithms with a one-hot algorithm indicator. Categorical choices are one-hot encoded and inactive or non-applicable hyperparameters are encoded consistently across all candidates in Λ. For the vector-based performance variants, namely Linear-Perf, RF-Perf, GP-Perf, and MeanMMR, we use the full historical performance vector P_λ directly and do not apply dimensionality reduction. This differs from OBOE (Yang et al., 2019), which uses low-rank PCA-based representations; in early experiments, our method improved as the representation approached the full vector.

B.5 Acquisition Rules

Expected Improvement is used for surrogate models that provide a predictive mean and uncertainty estimate, while MMR-based acquisition is defined in the main paper. For random forests, the predictive mean and standard deviation used by EI are estimated from the tree ensemble (Lindauer et al., 2022). Given predictive mean $\mu(\lambda)$, standard deviation $\sigma(\lambda)$, and best observed performance $\mathcal{P}^*$, EI is:

$$\mathrm{EI}(\lambda) = (\mu(\lambda) - \mathcal{P}^*)\,\Phi(Z) + \sigma(\lambda)\varphi(Z), \tag{3}$$

$$Z = \frac{\mu(\lambda) - \mathcal{P}^*}{\sigma(\lambda)}, \tag{4}$$

where Φ and φ are the standard normal CDF and PDF, respectively.

For MeanMMR, the general MMR rule from the main paper is instantiated with the historical mean as the relevance score and cosine similarity on raw performance vectors as the diversity term:

$$\mathrm{MeanMMR}(\lambda) = \alpha \cdot \frac{\sum_{s \in T_{\text{train}}} P_{s,\lambda}}{|T_{\text{train}}|} - (1-\alpha) \cdot \max_{\lambda' \in \Lambda_{\text{obs}}} \cos(P_\lambda,\, P_{\lambda'}) \tag{5}$$

Because the first term does not depend on target-task observations, the historical mean ranking is fixed, and only the diversity term updates as configurations are evaluated on the target task.

B.6 Gaussian Process Implementation

Gaussian processes (GP's) (Rasmussen & Williams, 2006) are the classical probabilistic surrogate in BO and provide a predictive posterior for EI. We include a GP surrogate with both representations. The Gaussian process surrogate is implemented using BoTorch's `SingleTaskGP` (Balandat et al., 2020) with a Matérn-$\frac{5}{2}$ kernel (Rasmussen & Williams, 2006). Automatic Relevance Determination (ARD) is enabled, assigning an independent lengthscale to each input dimension so the model can identify which hyperparameters are most predictive. Gamma priors are placed on the lengthscales ($\mathrm{Gamma}(3.0, 6.0)$) and output scale ($\mathrm{Gamma}(2.0, 0.15)$) to regularize the fit when few observations are available. Inputs are min-max normalized to $[0, 1]$ and outputs are standardised per task. Quantile scaling was attempted, which should automatically redistribute hyperparameter values that were originally sampled by TabArena authors at log-scale, but did not lead to an improvement. Kernel hyperparameters are optimized by maximising the marginal log-likelihood via L-BFGS-B at each BO step, warm-started from the previous iteration's solution.

BoTorch's implementation is used across experiments and variants for consistency. Its performance relative to scikit-learn's `GaussianProcessRegressor` (Pedregosa et al., 2011) with the default RBF kernel varies by benchmark: on the high-dimensional CASH benchmark, the simpler ARD-free scikit-learn kernel achieves lower regret for both representations (see Table 2 in the main paper), whereas BoTorch performs better on the lower-dimensional HPO benchmark. Neither variant surpasses Linear-Perf.

B.7 Random Forest Implementation

Both RF-Hyper and RF-Perf use scikit-learn's `RandomForestRegressor` (Pedregosa et al., 2011) with `n_estimators=100` and `random_state=42`. RF-Hyper additionally sets `min_samples_leaf=3`, following SMAC's convention (Lindauer et al., 2022) of preventing overly specific single-observation splits early in the search, when few configurations have been evaluated. RF-Perf uses scikit-learn's default `min_samples_leaf=1` and, like GP-Perf and Linear-Perf (Section B.4), operates on the full performance vector without dimensionality reduction. Predictive mean and standard deviation for EI are estimated from the tree ensemble as described in Section B.5.

B.8 Summary of Final Hyperparameters

For reproducibility, Table 3 lists the final hyperparameter settings used for every surrogate model in the main experiments. Settings not listed follow scikit-learn or BoTorch defaults. Two-Tower settings were tuned per benchmark and are listed separately in Table 4. Every model with a stochastic fitting procedure (Random Forest, GP restarts, Two-Tower initialization/training, and the meta-learning baselines) is seeded with `random_state=42`; Bayesian ridge regression has no internal source of randomness.

Table 3: Final hyperparameters used for every model in the main experiments. Exact configuration files are included in the code release.

Model	Config key	Final hyperparameters
Linear-Hyper / Linear-Perf	`ridge_ei` / `perf_vector_ridge_ei`	BayesianRidge, scikit-learn defaults ($\alpha_1 = \alpha_2 = \lambda_1 = \lambda_2 = 10^{-6}$, `max_iter=300`, `tol=1e-3`); Linear-Perf fits without intercept (Section C.1)
RF-Hyper	`rf_ei`	`n_estimators=100`, `min_samples_leaf=3`, `random_state=42`
RF-Perf	`perf_vector_rf_ei`	`n_estimators=100`, `min_samples_leaf=1` (default), `random_state=42`
GP-Hyper / GP-Perf	`botorch_gp_ei` / `botorch_gp_perf_{cash,hpo}`	BoTorch `SingleTaskGP`, Matérn-5/2 ARD kernel, lengthscale prior $\mathrm{Gamma}(3.0, 6.0)$, outputscale prior $\mathrm{Gamma}(2.0, 0.15)$, L-BFGS-B MLL optimization (Section B.6)
TwoTower-Hyper / TwoTower-Perf	`two_tower_hyper_*` / `two_tower_perf_*`	`random_state=42`; remaining settings tuned per benchmark by random search, see Table 4
ABLR	`ablr`	feature dim 50, hidden layers $[50, 50]$, 300 epochs, lr=0.01, `random_state=42`
Active-Testing	`active_testing`	similarity sharpness=1.0
RGPE	`rgpe`	256 bootstrap samples, 5 target CV folds, base-variance weighting enabled, weight-dilution prevention enabled, `random_state=42`
RGPE-Oracle	`rgpe_oracle`	as RGPE, with base-GP predictions replaced by true performance columns
TST-R	`tstr_discrete`	bandwidth=0.5, base-variance weighting enabled, `random_state=42`
MTGP	`mtgp_{hpo,cash}`	6 auxiliary tasks, 40 (HPO) / 48 (CASH) context flows, refit factor=3.0, `random_state=42`

B.9 Cost of GP-based Transfer and the RGPE-Oracle Ablation

Results of the GP-based meta-learning baselines MTGP (Swersky et al., 2013) and RGPE (Feurer et al., 2018) are reported on the HPO benchmark only. Both fit exact Gaussian processes, whose cost grows cubically with the number of observations n. RGPE maintains one base GP for every prior task, scaling with tn^3 for t tasks and n configurations per task. MTGP includes all observations in one Gaussian process, thus scaling even faster following $(tn)^3$. In the experimental setup for HPO ($n = 201$) this is affordable, on CASH ($n = 3623$) it is not. For both cases, $t = 40$ or $t = 41$ depending on the fold.

To quantify this, we timed BoTorch's `SingleTaskGP` fit on the real CASH hyperparameter features. Single-run fit times were roughly 48, 249, and 1029 seconds at $n = 600$, 900, and 1200 respectively, that roughly matches the $O(n^3)$ cost and confirms that BoTorch does not fall back to a sub-cubic solver in this regime. Extrapolating to the full CASH set, a single base GP fit at $n = 3623$ is on the order of six hours, so RGPE's roughly forty base GPs per fold amount to several days of pretraining per fold, before any target-task search. These are single measurements and intended only as orders of magnitude, but the gap to ABLR's 3.4-minute pretraining, or Linear-Perf's none, is large enough that we exclude the GP-based methods from CASH rather than report downsampled results.

MTGP remains feasible on HPO only after subsampling: each fit conditions on a random subset of 40 context configurations drawn from 6 auxiliary tasks, without which the joint multi-task GP over context and target observations does not fit the HPO time budget. On CASH the fit would scale even more extremely than the several days of pretraining, given that it includes all tasks in one model.

To separate the cost of the GP fitting from the value of the ranking-transfer idea, we also evaluate RGPE-Oracle, a GP-free ablation that replaces each base-GP's predictions with the true performance column of the corresponding prior task. This removes the cubic cost and lets the method run on CASH. On HPO, RGPE-Oracle reaches an average normalized regret of 10.00, competitive with the strongest methods and slightly ahead of the full GP-based RGPE (10.57), which indicates that the GP machinery adds little there. On CASH, where it can run but the full RGPE cannot, it reaches only 17.58, well behind Linear-Perf (12.19). This supports the interpretation in the main paper's Discussion section that the non-negative ranking-agreement weighting over prior tasks is more restrictive than the signed linear combination fit by Linear-Perf.

B.10 Two-Tower Implementation and Tuning

We use a Two-Tower surrogate as a collaborative-filtering-style meta-learning model. Let z_t denote the embedding of task t and z_λ the embedding of configuration λ. The predicted performance of configuration λ on task t is given by their dot product, $\hat{\mathcal{P}}(t, \lambda) = z_\lambda^\top z_t$ (Huang et al., 2013; Raza et al., 2026). It can learn across tasks when trained on the meta-training tasks, even when its configuration tower receives x_λ^{hyper}. The two representation choices differ in how z_λ is obtained: Two-Tower-Hyper maps x_λ^{hyper} to z_λ through a learnable encoder, whereas Two-Tower-Perf uses no encoder and instead learns one free configuration embedding per column of the historical performance matrix by fitting $P_{t,\lambda} \approx z_t^\top z_\lambda$. This makes Two-Tower-Perf closely related to matrix factorization. Since the Two-Tower model does not produce a predictive posterior, EI is unavailable, and we instead use the MMR acquisition defined in the main paper with relevance score $\hat{\mathcal{P}}(t, \lambda) = z_\lambda^\top z_t$ and z_λ the learned configuration embedding. The Two-Tower model offers a more nuanced comparison, being able to meta-learn using both hyperparameter or performance representations.

For Two-Tower-Hyper, the configuration tower is a multilayer perceptron that maps x_λ^{hyper} to z_λ. We do not concatenate separate learned configuration latents to this input, so the variant remains purely hyperparameter-based. The task tower is kept as an identity transformation from learned task latents z_t. For Two-Tower-Perf, both towers are identity transformations over learned latents z_λ and z_t, making it equivalent to a matrix-factorization-style model trained with the same procedure as Two-Tower-Hyper. During target-task adaptation, only the task-side parameters are updated. MMR is used as acquisition rule instead of expected improvement due to the lack of posterior distribution over the output of the architecture.

For a fair comparison, it is valuable to also tune the top-level methods themselves. We expected this to be most essential for the Two-Tower model, as a deep learning system is sensitive to settings such as learning rate and the dimensions of the multilayer perceptron. Both Two-Tower variants were optimized through fifty iterations of random search on the first of the five folds, optimizing average normalized test regret. This introduces a risk of overfitting and positively skewing the results for that model. We consider this acceptable for the present comparison because the main intention is to compare variants within the same model family, both variants are treated the same, and the tuning fold accounts for only a fifth of the final score. Alternatives such as nested cross-validation or a clean split with held-out tuning tasks would have required downscaling all experiments or accepting higher variance due to the smaller number of remaining tasks after holdout.

The shared search space covers three hyperparameters: embedding dimension sampled from $\{8, 16, 32\}$, learning rate sampled log-uniformly from $[0.005, 0.1]$, and batch size sampled from $\{256, 512, 1024, 2048\}$. Two-Tower-Hyper additionally searches over the hyperparameter-tower MLP architecture, with hidden-layer options $\{[16], [32], [64], [128], [16,16], [32,32], [64,64], [128,128]\}$, while Two-Tower-Perf has no MLP in the configuration tower (embeddings feed directly into the dot product). The optimal hyperparameters found per benchmark are listed in Table 4.

Table 4: Best hyperparameters found by random search for each Two-Tower variant and benchmark.

Variant	Benchmark	Emb. dim	LR	Batch size	Hyper. tower
Two-Tower-Hyper	CASH	8	0.007	1024	$[64, 64]$
Two-Tower-Hyper	HPO	8	0.007	256	$[16]$
Two-Tower-Perf	CASH	8	0.005	1024	—
Two-Tower-Perf	HPO	8	0.020	512	—

Figure 6 shows the test regret of both Two-Tower variants against MeanMMR and Linear-Perf on each benchmark. Performance on the HPO benchmark after tuning was almost the exact same between Two-Tower-Perf and Two-Tower-Hyper. This could suggest that training on the entire meta-dataset is sufficient for learning to map hyperparameter vectors to algorithm performance, providing evidence against claims that that mapping would be too complex of a task. However, it is also possible that Two-Tower-Perf does not see an improvement because of an unrelated bottleneck in the architecture that affects both variants. Neither variant clearly improves upon the baseline of MeanMMR, which should be reproducible in theory by a Two-Tower-Perf model with embedding size one where the configuration embedding value is the mean of the configuration. What that suggests is that while the model may be expressive enough, theoretical performance is lost as the training process does not find the optimal solution in practice. Increasing the number of epochs from 500 to 5000 did not yield a clear improvement. The Two-Tower variants were also observed to be highly sensitive to the training procedure parameters, unoptimized variants found during the tuning process sometimes performed worse than random search. So, while the Two-Tower model offers a good meta-learning solution for CASH in high-dimensional hyperparameter search space, the complexity of gradient-descent model training likely is the reason that it does not profit as much as other methods do when performance-based representations can be used directly.

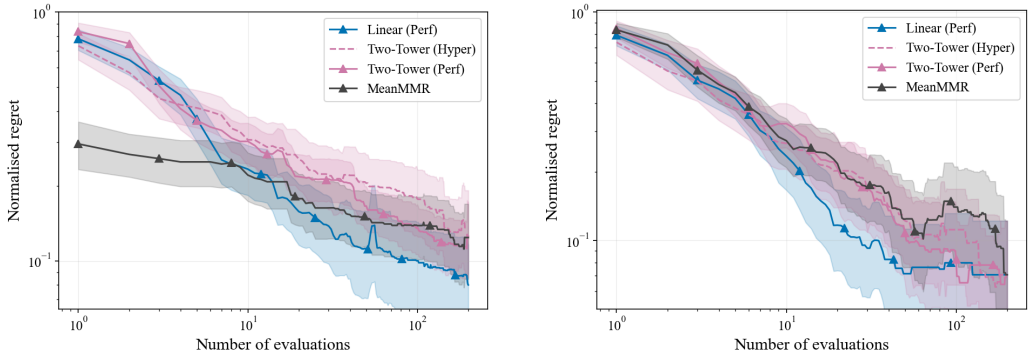

Figure 6: Test normalized regret of the Two-Tower variants on the CASH (left) and HPO (right) benchmarks, with MeanMMR and Linear-Perf as references. The two representations behave almost identically on HPO and similarly on CASH; neither clearly improves on MeanMMR, and both trail Linear-Perf. Shaded bands are 95% bootstrap CIs across tasks.

B.11 Compute

All experiments were run on a single machine using an AMD Ryzen 5 2600 Six-Core CPU, 16 GB of Memory and a GTX 1080 GPU, where only the Two-Tower model uses the GPU, running

Windows with Python 3.11. Key library versions are scikit-learn 1.3.2, PyTorch 2.5.1 (CUDA 11.8), BoTorch 0.16.1, GPyTorch 1.15.2, and Hydra 1.3.2; exact pinned versions for all dependencies are given in the code release's `pyproject.toml` and `uv.lock`. Table 5 reports total wall-clock time per model family across all tasks for both benchmarks including the scalability experiments, for 27h 20m in total.

Table 5: Total wall-clock compute time per model family.

Model family	**Wall-clock time**
Two-Tower	9h 25m
Gaussian Process	7h 43m
Random Forest	5h 40m
Meta-learning baselines	2h 57m
Model-free	0h 54m
Linear	0h 41m
Total	**27h 20m**

C Linear Surrogate Models

C.1 Bayesian Ridge Formulation

We use the Bayesian ridge regression formulation of MacKay (1992), as implemented in scikit-learn's `BayesianRidge` estimator (Pedregosa et al., 2011). Both Linear-Hyper and Linear-Perf use this estimator; for Linear-Perf, we disable the intercept to preserve the weighted-combination interpretation over prior tasks, which also gave a small empirical improvement. For target-task observations $\mathcal{D}_t^{(n)} = \{(x_{\lambda_i}, y_i)\}_{i=1}^n$, the surrogate assumes a linear Gaussian likelihood,

$$y_i = x_{\lambda_i}^\top w + \epsilon_i, \qquad \epsilon_i \sim \mathcal{N}(0, \alpha^{-1}), \tag{6}$$

with an isotropic Gaussian prior over the regression weights,

$$p(w \mid \eta) = \mathcal{N}(w \mid 0, \eta^{-1} I). \tag{7}$$

Here α denotes the noise precision and η the weight precision, corresponding to scikit-learn's coefficient-precision parameter `lambda_`; we use η to avoid overloading the configuration notation λ. Both precisions are themselves assigned Gamma hyperpriors,

$$\alpha \sim \mathrm{Gamma}(\alpha_1, \alpha_2), \qquad \eta \sim \mathrm{Gamma}(\lambda_1, \lambda_2), \tag{8}$$

using the shape-rate parameterization, where we use the scikit-learn defaults $\alpha_1 = \alpha_2 = \lambda_1 = \lambda_2 = 10^{-6}$. Training is therefore not variational inference. For fixed α and η, the posterior over w is available in closed form as a Gaussian. The precision parameters are then updated iteratively by empirical Bayes, also known as type-II maximum likelihood or evidence maximization, using the MacKay-style update rules described by MacKay (1992) and Tipping (2001). We keep the default stopping settings, `max_iter=300` and `tol=1e-3`. Predictions use the resulting Gaussian posterior over w to return both a predictive mean and standard deviation, which are passed to Expected Improvement.

C.2 Choice of the Linear Surrogate

A series of early experiments was done that resulted in the choice of Bayesian ridge regression. Initially, ordinary least squares optimization without regularization was attempted, referencing the existing work of OBOE (Yang et al., 2019). However, tuning the dimensionality reduction factor with PCA resulted in two findings. Firstly, predictive performance was most optimal when the rank equals the number of tasks (the full matrix rank). Secondly, there was a strong peak in prediction error whenever the number of observed target samples was equal to the rank of the dimensionality reduction. The peaking effect was most obvious when measuring prediction error per step, and not as easily seen when viewing regret curves over the entire process, possibly hiding this from previous authors. This peaking behavior is closely related to the classical peaking phenomenon and its

modern discussion under double descent (Loog et al., 2020; Viering & Loog, 2022). The instability of the unregularized model depending on the dimensionality motivated the use of Ridge regression without dimensionality reduction. This consistently provided better performance estimates without peaking behaviour.

Ridge regression was adapted to have an acquisition function either using non-probabilistic ridge regression with MMR, or Bayesian ridge regression using EI. When tuned well, both methods performed comparably. That also confirms the feasibility of using MMR as a replacement for EI. However, Ridge regression with MMR requires tuning the ridge regularization strength and MMR greediness, and Bayesian ridge regression with EI did not require any tuning, beyond always disabling the intercept. Hence Bayesian ridge regression was proposed and used to represent the linear model family.

Bayesian ridge regression re-estimates the noise and weight-prior precisions by marginal-likelihood maximisation on every per-task fit. To test whether this per-fit adaptation is the reason for any significant differences between the Linear-Perf model with respect to ABLR, we compare against a fixed-precision Bayesian linear regression head identical to the one used by ABLR (Perrone et al., 2018). On both benchmarks the two are statistically indistinguishable (average test normalized regret, CASH 12.19 vs. 12.08, $p = 0.76$; HPO 9.84 vs. 9.96, $p = 0.20$; paired Wilcoxon signed-rank over tasks). The adaptive prior precision estimation is therefore not the source of the linear model's performance: the same fixed-precision head used by ABLR performs equivalently when set once to a good value, supporting the use of the simpler Bayesian ridge surrogate that does not require setting the precision.

C.3 Sign of the Learned Task Weights

When Linear-Perf uses the historical performance vector as input, each regression coefficient weights performance on one specific prior task, so the fitted weights reveal how the target task is expressed as a combination of prior tasks. Figure 7 shows these weights for a representative CASH target task. As target-task observations accumulate, the fraction of negative weights grows to roughly half, far from the non-negative weights that a simple averaging of prior tasks would produce. The Bayesian ridge surrogate therefore does not merely average similar prior tasks; it also subtracts dissimilar ones. This signed combination is strictly more expressive than the non-negative ranking-agreement weights used by RGPE, consistent with RGPE-Oracle trailing Linear-Perf on CASH (see Section B.9).

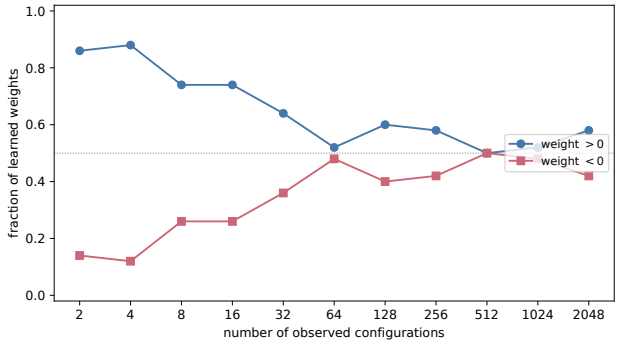

Figure 7: Sign of the learned Bayesian-ridge weights over prior tasks for a representative CASH target task using the performance representation: the fraction of weights that are positive versus negative as the number of observed target-task configurations grows.

D Adaptive Overfitting

As described in Section B.3, we separate validation and test regret, and report only test regret in all main figures and tables. Figure 8 and Figure 9 compare the two metrics per method for the CASH and HPO benchmarks. Validation regret is consistently somewhat better than test regret, although

this pattern is shared across methods and would not have significantly changed the main conclusions. The exception is a slightly larger gap for active testing. None of the methods appear to be overfitting to the extent that early stopping would have been necessary. We retain the validation-test split and reported test regret as it reflects more accurately how well a configuration might work when applied to real-world data after deployment that is not from the split it has been tuned on.

Across methods, validation and test regret track each other closely and decline together, with only minor gaps at higher evaluation counts (for example Active Testing on CASH). We therefore observe little adaptive overfitting in these results. On HPO the two curves stop improving simultaneously, which points to the limited search space, rather than overfitting to the validation split, as the factor that bounds further progress. We nonetheless retain the validation-test split as a conservative safeguard, since selecting the best-so-far configuration on the same data used to guide the search can in principle reward overfitting (Schröder et al., 2025).

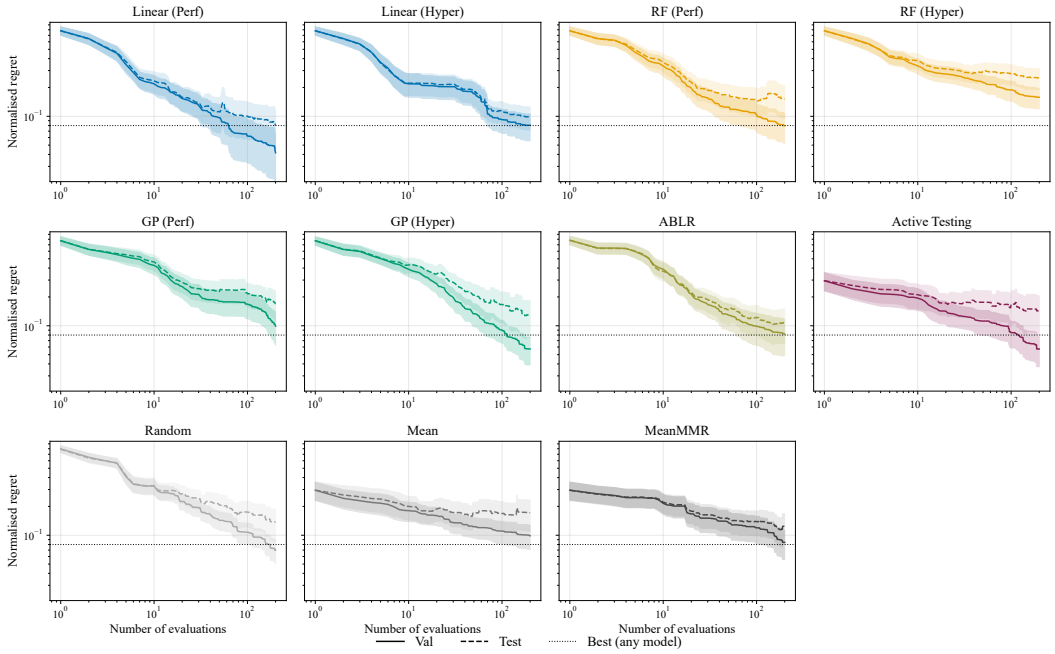

Figure 8: Validation and test regret for all methods on the CASH benchmark, with a horizontal line highlighting the best obtained test regret across methods within the 200 iterations.

E TabArena Leaderboard Dataset

E.1 Base Model Set

All models are taken from the TabArena benchmark leaderboard (Erickson et al., 2026). For most models, the dataset contains results of one default configuration plus 200 randomly sampled configurations from the model's search space (201 total). Foundation models that were only evaluated at their default setting have a single configuration.

E.2 XGBoost Search Space

The XGBoost search space used by the TabArena authors to generate the 200 random configurations is shown below in Table 7.

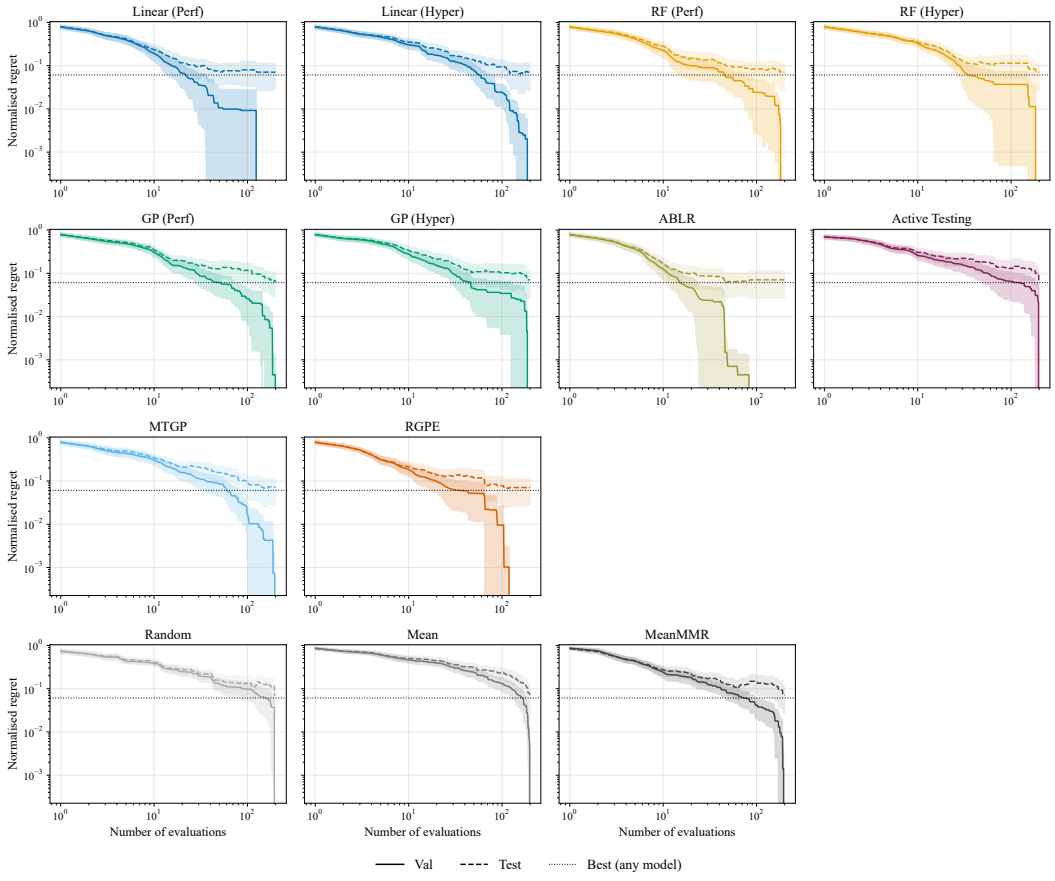

Figure 9: Validation and test regret for all methods on the HPO benchmark, with a horizontal line highlighting the best obtained test regret across methods within the 200 iterations.

Table 6: Base models used in the TabArena meta-dataset and the number of configurations evaluated per model.

Model	Category	Configurations
XGBoost	Gradient boosting	201
LightGBM	Gradient boosting	201
LightGBM (AIO)	Gradient boosting	201
CatBoost	Gradient boosting	201
Extra-Trees	Tree ensemble	201
Random Forest	Tree ensemble	201
k-NN	Classical	201
Linear Model	Classical	201
MLP (Torch)	Neural network	201
MLP (FastAI)	Neural network	201
RealMLP	Neural network	201
ModernNCA	Neural network	201
TabM	Neural network	201
xRFM	Neural network	201
EBM	Interpretable	201
TabPFNv2	Foundation model	201
RealTabPFN v2.5	Foundation model	201
TabDPT	Foundation model	201
TabICL	Foundation model	1
LimiX	Foundation model	1
Mitra	Foundation model	1
BetaTabPFN	Foundation model	1
TabFlex	Foundation model	1
Total		**3623**

Table 7: XGBoost hyperparameter search space. Log scale means the parameter is sampled uniformly in log space.

Hyperparameter	Type	Range	Scale
learning_rate	Float	$[5 \times 10^{-3},\ 10^{-1}]$	Log
max_depth	Integer	$[4,\ 10]$	Log
min_child_weight	Float	$[10^{-3},\ 5.0]$	Log
subsample	Float	$[0.6,\ 1.0]$	Linear
colsample_bylevel	Float	$[0.6,\ 1.0]$	Linear
colsample_bynode	Float	$[0.6,\ 1.0]$	Linear
reg_alpha	Float	$[10^{-4},\ 5.0]$	Linear
reg_lambda	Float	$[10^{-4},\ 5.0]$	Linear
grow_policy	Categorical	{depthwise, lossguide}	—
max_cat_to_onehot	Integer	$[8,\ 100]$	Log
max_leaves	Integer	$[8,\ 1024]$	Log