DYNAMIC HYPERPARAMETER IMPORTANCE FOR EFFICIENT MULTI-OBJECTIVE OPTIMIZATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Choosing a suitable ML model is a complex task that can depend on several objectives, e.g., accuracy, fairness, or energy consumption. In practice, this requires trading off multiple, often competing, objectives through multi-objective optimization (MOO). However, existing MOO methods typically treat all hyperparameters as equally important, disregarding that hyperparameter importance (HPI) can vary significantly across objectives. We propose a novel dynamic optimization approach that prioritizes the most influential hyperparameters based on varying objective trade-offs during the search, thereby accelerating empirical convergence. We advance prior work on HPI for MOO from post-analysis to direct, dynamic integration within the optimization, using HyperSHAP, a recent HPI method that attributes performance gains to hyperparameters via Shapley values. For this, we leverage the objective weightings naturally produced by the MOO algorithm ParEGO and reduce the configuration space by fixing the unimportant hyperparameters, allowing the search to focus on the important ones. We evaluate our method on synthetic tasks from PyMOO and moderate-dimensional HPO tasks from YAHPO-Gym. For HPO, integrating HPI yields up to 42% improvement in final Pareto front quality, while on synthetic data, the improvement is 24% over standard ParEGO.

1 INTRODUCTION

Hyperparameter optimization (HPO) is a critical step in maximizing the performance of machine learning models (Bergstra & Bengio, 2012; Snoek et al., 2012; Lévesque et al., 2016; Feurer & Hutter, 2019; Bischl et al., 2023). Traditionally, HPO has been conducted with a single performance objective in mind, often predictive accuracy. However, real-world applications rarely depend solely on accuracy. Instead, they must balance multiple, often conflicting requirements. For example, large-scale deployments demand low inference latency, while embedded and edge devices require strict limits on memory and energy usage. In socially sensitive applications, fairness constraints may be mandated by regulation (Schmucker et al., 2020; Weerts et al., 2024). This motivates the need to frame HPO as a multi-objective optimization (MOO) problem, where the goal is not a single best configuration but an approximation of the Pareto front, a set of optimal trade-offs across objectives. From that, a developer can make an informed decision about the final model to deploy. We focus on the common setting of HPO for Deep Learning with moderate-dimensional configuration spaces, where fewer trials can save substantial compute due to the high cost of model training.

Compared to single-objective HPO, multi-objective variants are considerably more complex and computationally demanding, making efficiency even more critical. While existing MOO methods (Elsken et al., 2019; Zhao et al., 2021; Morales-Hernández et al., 2022) assume uniform Hyperparameter Importance (HPI), single-objective studies show that HPI varies by task (Bergstra & Bengio, 2012; Hutter et al., 2014; van Rijn & Hutter, 2018; Wever et al., 2026). Leveraging this variability in HPI can improve optimization efficiency (Hutter et al., 2014; Wang et al., 2025; Wever et al., 2026).

Recent work shows that HPI varies across different objective trade-offs in multi-objective settings (Theodorakopoulos et al., 2024). We hypothesize that optimizers with dynamic scalarization, such as ParEGO (Knowles, 2006), can exploit this by adapting their search toward the most relevant hyperparameters. Our approach achieves this by dynamically reducing the configuration space for the current scalarization. Reducing the configuration space to the effective dimensionality enables the surrogate to learn more efficiently and accurately from limited data (Wang et al., 2016; Eriksson

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.

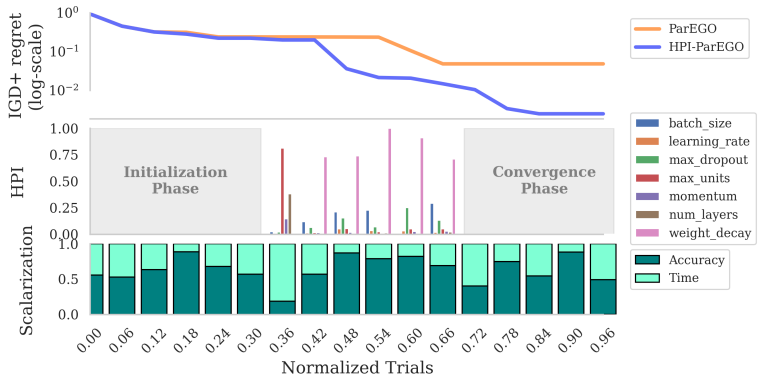


Figure 1: HPO task `lcbench_126025`: accuracy–time trade-off. Top: ParEGO vs. our variant with dynamic HPI on IGD+ regret over trials. Middle: HPI evolution across scalarizations; incl. initialization and consolidation phases with no reduction of the configuration space. Bottom: Scalarizations sampled by ParEGO.

et al., 2019), potentially also mitigating the curse of dimensionality (Bellman et al., 1957). Figure 1 displays the superior performance of our optimizer against the baseline in an illustrative example.

Contributions

1. We present the first efficient MOO method that dynamically incorporates HPI into the optimization loop of scalarization-based MOO approaches. HPI-ParEGO adaptively selects and focuses on the most influential hyperparameters based on the current objective scalarization.
2. We provide insights into the performance of our method with synthetic and HPO tasks, focusing on moderate-dimensional settings from PyMOO (Blank & Deb, 2020) and YAHPO-Gym (Pfisterer et al., 2022). Our HPI-based hyperparameter reduction demonstrates superior performance in single-objective optimization and outperforms standard ParEGO on PyMOO and all our MOO baselines on YAHPO-Gym in convergence speed and Pareto front quality.
3. We demonstrate the impact of key design choices through an ablation study, which includes the amount of randomness, HPI thresholds, when to draw a new scalarization, configuration space reduction, and the informativeness of HyperSHAP vs. random hyperparameter sets.

2 Background and Related Work

Multi-objective optimization (MOO). MOO simultaneously optimizes two or more potentially conflicting objectives. It produces a *set* of Pareto-optimal solutions, instead of a single solution, where no objective can be improved without degrading another (Pareto, 1971). This set, known as the Pareto front, represents the trade-off surface among objectives:

$$\begin{aligned} &\min_{\mathbf{x}\in\mathcal{X}} \mathbf{f}(\mathbf{x}) = (f_1(\mathbf{x}), \ldots, f_m(\mathbf{x})) \\ &\text{subject to: } \neg\exists \mathbf{x}' \in \mathcal{X} \text{ such that } f_i(\mathbf{x}') \leq f_i(\mathbf{x})\ \forall i, \\ &\text{with } f_j(\mathbf{x}') < f_j(\mathbf{x}) \text{ for some } j \end{aligned} \tag{1}$$

MOO is often tackled using either evolutionary algorithms or Bayesian optimization (Morales-Hernández et al., 2022). A well-known evolutionary approach is NSGA-II (Deb et al., 2002), which handles the objectives and maintains solution diversity through non-dominated sorting and crowding distance. ParEGO (Knowles, 2006) is a prominent Bayesian method for HPO that facilitates standard surrogate-based optimization by converting MOO problems into a series of single-objective problems via random scalarization weight sampling:

$$\min_{\boldsymbol{\lambda}\in\Lambda} \left\{ \max_{j=1,\ldots,m} [w_j \cdot f_j(\boldsymbol{\lambda})] + \rho \sum_{j=1}^{m} w_j \cdot f_j(\boldsymbol{\lambda}) \right\} \tag{2}$$

where $\boldsymbol{\lambda} \in \Lambda$ represents a candidate configuration from the configuration space Λ, $f_j(\boldsymbol{\lambda})$ denotes the j-th objective function to be minimized, $\boldsymbol{w} = (w_1, \ldots, w_m)$ is a weight vector sampled randomly, where $w_j \geq 0$ and $\sum_{j=1}^{m} w_j = 1$, and ρ is a small positive scalar (e.g., 0.05) that encourages diversity. While efficient, ParEGO and similar algorithms typically treat all hyperparameters equally, ignoring that some may have more impact on the current objective trade-off. Our work embeds dynamic HPI estimation into ParEGO to prioritize the most relevant hyperparameters during optimization.

Hyperparameter Importance. In single-objective HPO, a variety of HPI methods exist. These are typically applied post-hoc, using the results of completed optimization runs. Surrogate models trained on such optimization data often model performance as a function of hyperparameter configurations. Notable techniques include fANOVA (Hutter et al., 2014) assessing HPI by decomposing performance variance, as well as forward selection (Hutter et al., 2013) and Local Parameter Importance (Biedenkapp et al., 2018). Shapley values (Shapley, 1953) have also been applied to attribute HPI (Adachi et al., 2024; Rodemann et al., 2024; Wever et al., 2026). As a local approach, ablation path analysis (Fawcett & Hoos, 2016; Biedenkapp et al., 2017) measures hyperparameter contributions by comparing the default configuration to optimized ones. Building on these foundations, Theodorakopoulos et al. (2024) extended HPI estimation to the multi-objective setting. Their approach scalarized objective values from the Pareto front and applied fANOVA and ablation analysis to retrospectively evaluate HPI. Our work advances this line of research by directly integrating HPI into the optimization, enabling dynamic and adaptive search guidance. Additionally, we use HyperSHAP as an HPI measure, as it quantifies performance gains.

Dynamic Configuration Space Reductions. Several HPO methods dynamically reduce the configuration space to improve efficiency. Wistuba et al. (2015) prune unpromising regions using prior experiments or early evaluations, Lee et al. (2022) shrink or expand the space as the budget evolves, and Optuna (Akiba et al., 2019) allows users to adjust it during optimization. SEBO (Liu et al., 2023) and BONSAI (Daulton et al., 2026) promote sparsity by resetting low-impact hyperparameters to their defaults for individual configurations, but do not explicitly model HPI to reduce the configuration space globally. ExperienceThinking (Wang et al., 2021), SAASBO (Eriksson & Jankowiak, 2021), and VS-BO (Shen & Kingsford, 2023) focus the search on influential hyperparameters, whereas GSOS (Wang et al., 2025) leverages HPI to group hyperparameters and optimize them in order of importance. However, these methods are designed for single-objective settings; they neither account for HPI varying across objective trade-offs nor explicitly quantify interactions. PriMO (Basu et al., 2025) incorporates human priors per objective in the acquisition function of MOO, but they rely on user-defined beliefs and their priors cover only the extremes of the Pareto front, whereas our approach automatically adapts the search using HPI and explores a broader region of the Pareto front. These efforts reflect a growing interest in adaptive HPO techniques that tailor the configuration space to improve search efficiency. However, to the best of our knowledge, no prior work incorporates dynamic HPI estimation into multi-objective HPO.

3 Dynamic Hyperparameter Reduction

We extend the ParEGO algorithm by incorporating HPI estimation into each optimization step. Our method dynamically identifies the most influential hyperparameters under the current objective scalarization and uses this information to guide the search (see Figure 2). Section 3.1 outlines the approach; Section 3.2 details HPI integration and configuration space reduction; and Section 3.3 presents the thresholding strategy for how aggressively the space is reduced.

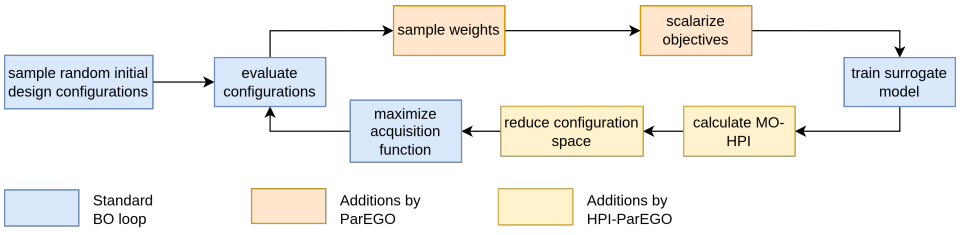


Figure 2: Algorithm overview: Blue boxes represent the standard Bayesian optimization (BO) loop, orange boxes additions made by ParEGO, and yellow boxes those introduced by our proposed HPI-ParEGO.

Algorithm 1: HPI-ParEGO (Blue highlighting the additional parts of our approach)

Input: Objective functions $f(\cdot) = (f_1(\cdot), \ldots, f_m(\cdot))$, configuration space Λ, acquisition function α, budget B, initial size N_{init}, initial weights $\boldsymbol{w} \sim \Delta^{m-1}$, threshold τ, random chance r, weight update every u iterations

Output: Approximated Pareto Set

1 Initialize history $\mathcal{H} = \{(\boldsymbol{\lambda}_i, f(\boldsymbol{\lambda}_i))\}_{i=1}^{N_{\text{init}}}$ with an initial design (e.g., random search)
2 **for** $i = N_{init} + 1$ **to** B **do**
3 **if** $i \bmod u = 0$ **then**
4 Update scalarization weights $\boldsymbol{w} \sim \Delta^{m-1}$ `// For the original ParEGO u = 1`
5 Train surrogate model $\hat{f}_{\boldsymbol{w}}$ on $\mathcal{H}$ using the scalarized objective value of ParEGO (see Equation 2)
6 **if** $1 - r$ *chance* **then**
7 Compute HPI $h(\Lambda^j)$ for each hyperparameter Λ^j `// See Section 3.2`
8 Select important hyperparameters (HPs) by HPI threshold τ based on $h(\Lambda^j)$
9 Reduce configuration space $\Lambda' \subset \Lambda$ by fixing unimportant HPs
10 Select configurations: $\boldsymbol{\lambda}_i \in \arg\max_{\lambda \in \Lambda'} \alpha(\boldsymbol{\lambda} \mid \mathcal{H}, \hat{f}_{\boldsymbol{w}})$
11 **else**
12 Select random configuration $\boldsymbol{\lambda}_i$ from original configuration space $\boldsymbol{\lambda}_i \in \Lambda$
13 Evaluate $f(\boldsymbol{\lambda}_i)$ and add result to history: $\mathcal{H} \leftarrow \mathcal{H} \cup \{(\boldsymbol{\lambda}_i, f(\boldsymbol{\lambda}_i))\}$
14 Update threshold τ `// See Section 3.3`
15 **return** Pareto Set based on $\mathcal{H}$

3.1 General Algorithm: HPI-ParEGO

Our method builds on ParEGO (Knowles, 2006) (Equation 2).[1] By varying the weights across iterations, ParEGO gradually explores diverse tradeoffs along the Pareto front. ParEGO (Algorithm 1) starts from an initial design of sampled configurations (Line 1). At each iteration, scalarization weights $\boldsymbol{w}$ are drawn uniformly from the unit simplex (Lines 3-4), determining the emphasis on each objective. The evaluated configurations $\boldsymbol{\lambda}_i$ and their costs $f(\boldsymbol{\lambda}_i)$ in the optimization history $\mathcal{H}$ are then scalarized with $\boldsymbol{w}$, and a surrogate model $\hat{f} : \boldsymbol{\lambda} \mapsto f(\boldsymbol{\lambda}_i)$ is fitted to approximate performance under the current scalarization (Line 5).

We extend the ParEGO-loop by incorporating HPI (Lines 7-9). Using the surrogate model $\hat{f}$, we estimate how important each hyperparameter Λ^j is for the current scalarized objective, yielding an importance score $h(\Lambda^j)$. We then select the smallest subset of hyperparameters that collectively account for at least a fraction τ (Line 8) of the total performance gain, retaining only the most influential dimensions $\Lambda' \subset \Lambda$ (Line 9) and concentrating the optimization budget where changes are most likely to improve performance. The remaining hyperparameters are fixed to a constant value (i.e., the current incumbent configuration in our implementation[2]). Because both the importance estimates and the scalarization weights jointly determine the reduced space, changing the weights too frequently would discard useful structure before it can be fully exploited. We therefore update $\boldsymbol{w}$ only every u iterations (Line 3), giving the optimizer time to capitalize on a stable reduced space. Note that HPI is re-estimated at every iteration as the surrogate improves with new evaluations.

Next, the acquisition function α, balancing exploration and exploitation, e.g., expected improvement (Jones et al., 1998) or upper confidence bound (Srinivas et al., 2010), is optimized over Λ' to propose new candidates (Line 10), which are evaluated and added to the history $\mathcal{H}$ (Line 13). The threshold τ may then be updated based on the optimization's progress (Line 14).

Following established practice in Bayesian optimization (e.g., (Falkner et al., 2018; Lindauer et al., 2022)), we interleave, with a random chance r, a random configuration drawn from the full space

[1] Our approach only assumes that the objectives are scalarized during the optimization, and thus is applicable to approaches similar to ParEGO (Zhang & Li, 2007; Bradford et al., 2018) with minor modifications.

[2] Concretely, we define the incumbent as the configuration in $\mathcal{H}$ minimizing the current scalarized objective $\hat{f}_{\boldsymbol{w}}$. Using the incumbent as the reference point has two advantages: (i) HyperSHAP quantifies the improvement over the incumbent and thus pushes the optimizer to even better configurations; and (ii) the reference point may change over time, and thus it mitigates the risk of a local optimum.

Λ (Line 12). This guards against premature convergence to early HPI estimates and ensures that hyperparameters deemed unimportant under a scalarization can still be explored under future tradeoffs.

3.2 Calculating MO-HPI and Reducing the Configuration Space

We dynamically estimate HPI during optimization to focus on the most promising dimensions. Since HPI is re-evaluated on every iteration, the choice of HPI method is critical. We adopt HyperSHAP (Wever et al., 2026), which frames hyperparameter optimization as a cooperative game (Fudenberg & Tirole, 1991) and applies Shapley values to quantify tunability (Probst et al., 2019). Concretely, let λ^* denote a reference configuration and $\hat{f}$ the surrogate model trained on the run history $\mathcal{H}$. HyperSHAP treats each hyperparameter Λ^j as a player and computes its Shapley values $h(\Lambda^j)$, measuring the expected marginal gain in $\hat{f}$ from tuning Λ^j away from λ^*, averaged over all possible subsets of co-tuned hyperparameters. Because the Shapley values are defined relative to a reference configuration rather than by variance decomposition, they capture directed performance gains and naturally account for higher-order interactions. In the multi-objective setting, we leverage ParEGO's scalarization to reduce this to a single-objective tunability problem at each iteration.

By the efficiency axiom of Shapley values, the individual contributions sum to the total gain over the reference $\sum_{\Lambda^j \in \Lambda} h(\Lambda^j) = \hat{f}(\lambda^*_\Lambda) - \hat{f}(\lambda^*)$, where λ^*_Λ denotes the configuration obtained by tuning all hyperparameters. We exploit this property to reduce the configuration space by selecting the smallest subset that accounts for at least a τ-fraction of the total gain

$$\arg\min_{\Lambda' \subseteq \Lambda} |\Lambda'| \quad \text{subject to} \quad \frac{\sum_{\Lambda^j \in \Lambda'} h(\Lambda^j)}{\sum_{\Lambda^k \in \Lambda} h(\Lambda^k)} \geq \tau.$$

In practice, we solve this greedily by ordering hyperparameters by their Shapley values and selecting the top contributors until their cumulative share exceeds τ. If an alternative HPI method is used, a simpler selection rule, such as retaining all hyperparameters above the τ-quantile, can be substituted.

We consider only the most influential hyperparameters and reduce optimization complexity by constructing a reduced configuration space $\Lambda' \subset \Lambda$, by diminishing the influence of less important hyperparameters. Based on the importance scores $h(\Lambda^j)$, we retain only the most relevant hyperparameters $\Lambda' \subset \Lambda$ and fix the rest to constants (we use the incumbent). Each reduction starts from the original configuration space Λ. In rare cases, this leads to a too-narrow configuration space, such that no new configurations can be selected; we then fall back to the original space Λ for that iteration.

3.3 Dynamic Threshold

The threshold τ determining the selected hyperparameters changes over the course of the optimization. In principle, one can consider two lines of argument: (i) The optimizer should start with a reduced configuration space to make quick progress, and later the space should not be reduced to allow for also tuning less important dimensions; (ii) Alternatively, the surrogate model (used for optimization and for HPI estimation) requires a sufficient number of trials to provide reliable estimates before the space can be reduced. We argue that both are valid and thus propose a simple combination that leverages their advantages: First, we disable hyperparameter reduction during the initial third of trials, allowing broad exploration and reliable estimation of HPI (initialization phase). Then we apply a threshold of $\tau = 0.8$, corresponding to 80% of the cumulative contribution measured by the Shapley values, for the next third of trials to focus the search on the most relevant hyperparameters and accelerate progress. Finally, in the last third of trials, we reconsider all hyperparameters, allowing the optimizer to tune less important dimensions (consolidation phase). We call this approach "Symmetric-0.8".

4 Empirical Evaluation

In this section, we begin by describing our experimental setup and then evaluate the proposed HPI-ParEGO along the following research questions: (i) How well does the HPI-selection work for single-objective optimization? (Sec. 4.2) (ii) Is our HPI-based approach more effective for MOO than the standard ParEGO algorithm on well-understood synthetic benchmark functions? (Sec. 4.3) (iii) How do the design choices of HPI-ParEGO impact its performance? (Sec. 4.4) (iv) How well do the results translate to real-world HPO benchmarks? (Sec. 4.5)

4.1 Experiment Setup

Datasets. We conduct our experiments on synthetic functions from PyMOO (Blank & Deb, 2020) and HPO tasks from YAHPO-Gym (Pfisterer et al., 2022), spanning a range of configuration space dimensionalities. From PyMOO, we select the `ZDT` suite (Zitzler et al., 2000) (30 dimensions for `ZDT1-3`, 10 for the others), excluding `ZDT5`, which is defined over a discrete bitstring domain and therefore less amenable to conversion into an HPO problem. From YAHPO-Gym, we select 13 tasks each from `LCBench` (Zimmer et al., 2021) (7 dimensions) and `rbv2_ranger` (8 dimensions)[3], chosen because different hyperparameters matter for different objective trade-offs (see Appendix A for details). Of these, three tasks per scenario serve as an initial ablation study, with the remaining 20 reserved for the final evaluation to avoid biasing it.

Baselines. Our primary baseline is the ParEGO implementation in the HPO framework SMAC3 (Lindauer et al., 2022), without our HPI modifications. It uses a random forest as the surrogate model and expected improvement as the acquisition function, a combination that has shown strong performance in prior HPO benchmark studies (Eggensperger et al., 2021). The acquisition function is optimized by sorted random search. The surrogate model is retrained every two evaluations, scalarization weights are updated every ten iterations ($u = 10$), and a random configuration is evaluated with a random chance $r = 10\%$. The default configuration of the configuration space is always evaluated first. We also compare against several established MOO algorithms: multi-objective TPE (Ozaki et al., 2020), NSGA-II (Deb et al., 2002), and EHVI (Daulton et al., 2020), all implemented in Optuna (Akiba et al., 2019)[4], and DE from Nevergrad (Rapin & Teytaud, 2018).

Evaluation Metrics. We track the normalized best-so-far performance using two common multi-objective indicators: inverted generational distance plus (IGD+) and hypervolume (HV). IGD+ quantifies the average distance from each point in a reference Pareto front to its closest point in the solution set (Ishibuchi et al., 2015). The reference Pareto front pools the costs across optimizers and seeds and is normalized per task to a [0,1] scale. The HV indicator is the volume of objective space dominated by the current Pareto front, bounded by a reference point set to the componentwise maximum of all observed cost vectors; we negate the HV to minimize it consistently. For both indicators, we apply per-task min-max normalization, yielding normalized regrets on a [0,1] scale where lower is better. Convergence plots show these regrets over normalized trials (i.e., function evaluations), averaged first across tasks and then across seeds, with shaded regions indicating ± 1 standard error. For the ablation study, we report the mean area under the curve (AUC) of the per-seed convergence curves for each indicator, each averaged across tasks as a summary measure.

Implementation Details. Our HPI-ParEGO variant inherits all settings from the ParEGO baseline and augments them with our HPI extension, ensuring that any performance differences are attributable to the proposed method. All experiments are executed through the CARP-S benchmark suite (Benjamins et al., 2026), using 10 random seeds for the main evaluation and 5 for the ablation study. More details on implementation, hardware, and per-task trial counts are provided in Appendices D, B, and C.

4.2 Single Objective Optimization

We first evaluate whether HPI-based configuration space reduction improves convergence speed in the simpler single-objective optimization (SOO) setting. To this end, we replace ParEGO with SMAC's default HPO optimizer while keeping all other settings identical to those used in the MOO experiments to ensure a fair comparison. We target validation accuracy on the selected `LCBench` tasks and measure performance via normalized mean incumbent regret. As shown in Figure 3, the HPI-reduced configuration space consistently outperforms the full configuration space baseline in SOO, lending confidence that the benefits of HPI transfer to the MOO setting studied in the remainder of this paper.

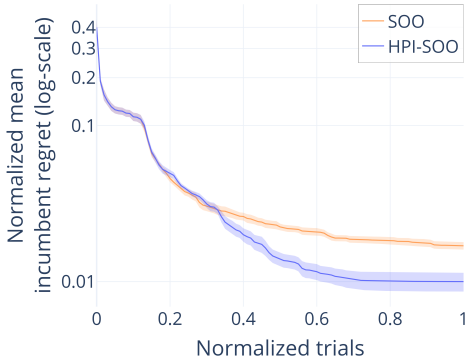
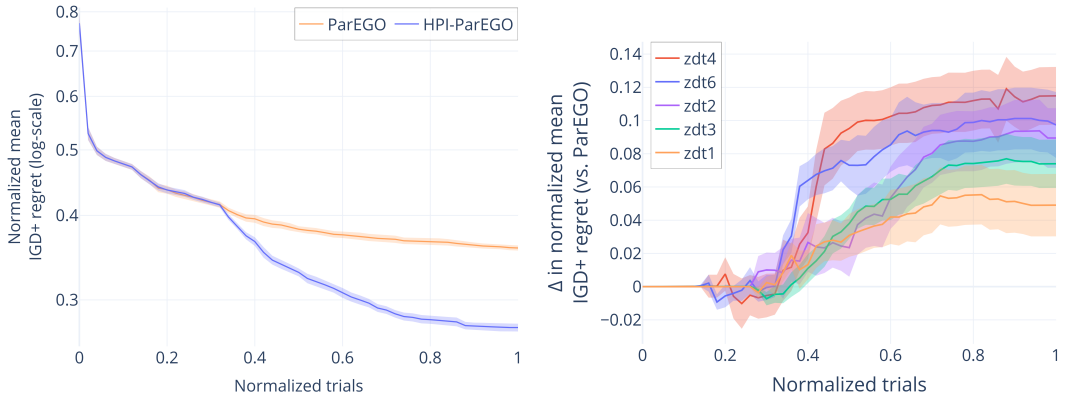


Figure 3: The HPI-based optimizer for single-objective optimization compared to baseline HPO on the selected `LCBench` tasks.

[3]Fidelity hyperparameters are fixed.

[4]For EHVI, we use Optuna's BoTorchSampler with default settings.



(a) The HPI-ParEGO optimizer compared to the ParEGO optimizer on IGD+ regret.

(b) Difference in mean IGD+ regret with standard error between ParEGO and HPI-ParEGO. Values > 0 indicate a better performance of HPI-ParEGO.

Figure 4: Results on the PyMOO tasks.

4.3 HPI-ParEGO on PyMOO

We next evaluate our approach on the selected PyMOO tasks (Figure 4a), which are established synthetic MOO benchmarks that allow for detailed analysis. HPI-ParEGO begins to outperform the ParEGO baseline at roughly 33% of the trial budget, precisely when the HPI-based configuration space reduction begins (see Subsection 3.3). This indicates that the most important hyperparameters are identified early. Per-task breakdowns (Figure 4b) confirm that HPI-ParEGO substantially outperforms ParEGO across all PyMOO functions. Together, these results demonstrate that HPI-ParEGO handles well-known synthetic benchmarks effectively and accelerates convergence throughout the trial budget. We further examine its behavior as dimensionality increases in Appendix F. Appendix E shows the HV and IGD+ results compared to all baselines. We note that, by construction, ParEGO can have slight advantages on HV compared to other baselines, and IGD+ is seen as a fairer metric. Appendix E shows that on these artificial benchmarks, an HV-based acquisition function (EVHI) efficiently exploits the fairly simple Pareto fronts across both metrics. However, as we show later, this advantage does not generalize to more complex real-world problems.

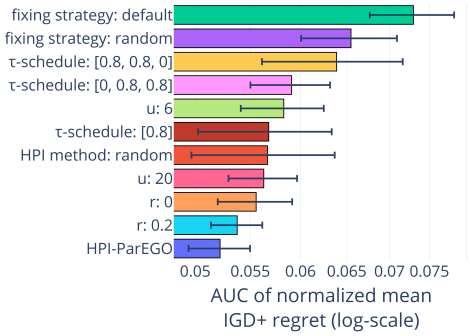
4.4 Ablation Study

To address our third research question of how the design choices of HPI-ParEGO affect its performance, we conduct a systematic ablation study, in which we *vary one component at a time* while holding all others fixed to the configuration of our proposed method (Table 5a). The study is carried out on six held-out tasks, three from `LCBench` and three from `rbv2_ranger`, which are excluded from all subsequent evaluations to guard against overfitting to these design decisions. Full per-task results with IGD+ regret and HV regret are provided in Appendix D. Figure 5b summarizes the findings by reporting the AUC as each component is varied away from the HPI-ParEGO default.

The most impactful design choice is the strategy for fixing unimportant hyperparameters: setting them to their current incumbent values yields a substantial advantage over using defaults or random values. This is intuitive, as HyperSHAP measures tunability relative to the incumbent; fixing low-importance hyperparameters to that reference preserves a strong configuration while concentrating the optimization budget on high-impact dimensions. Replacing HyperSHAP selection with random selection leads to a consistent performance drop, confirming that HPI provides a more informative criterion. Ablating the Symmetric-0.8 schedule shows that combining the consolidation and initialization phases yields better performance than either alone. Finally, the occasional evaluation of the unreduced space and the frequency of weight updates each contribute a small but positive effect.

Component	Values
HPI method:	HyperSHAP, random values
Fixing strategy:	incumbent, default, random
Threshold τ-schedule:	Initialization (0,0.8,0.8), Consolidation (0.8,0.8,0), Constant-0.8, Symmetric-0.8
Random chance r:	0.0, 0.1, 0.2
Weight update iterations u:	6, 10, 20

(a) Explored settings of the ablation study.



(b) AUC of IGD+ regret for ablating over different algorithm components.

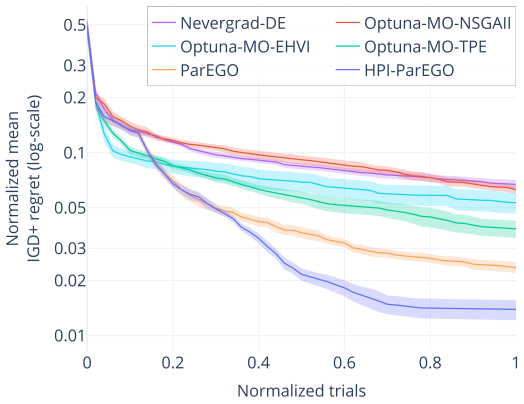
Figure 5: Ablation study results.

4.5 HPI-ParEGO on HPO Benchmarks

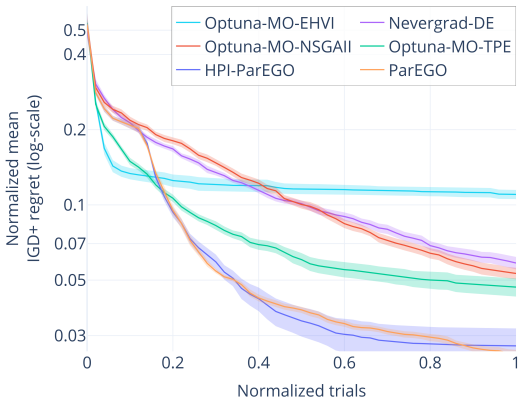
Lastly, we evaluate on real-world HPO problems, comparing HPI-ParEGO against ParEGO and the additional baselines on the selected YAHPO-Gym tasks (Figure 6). For LCBench, HPI-ParEGO emerges as the best optimizer, overtaking EHVI and MO-TPE at around 15-20% of the trial budget, with a large gap to ParEGO. Compared to the synthetic results, learning accurate HPI values proves more challenging on real HPO benchmarks, resulting in a smaller gap between ParEGO and HPI-ParEGO for `rbv2_ranger`, where conditional configuration spaces add further complexity. Interestingly, EHVI stagnates very quickly on both scenarios, showing that EHVI's strong results on PyMOO (Appendix E) cannot be reproduced on complex real-world tasks. In summary, HPI-ParEGO performs strongly across both benchmark suites. HV regret results and per-task differences between HPI-ParEGO and ParEGO for each scenario are shown in Appendix G.

4.6 Computational Overhead

Calculating HPI every iteration adds some computational overhead. On PyMOO, we measured it as $\sim$ 20 seconds per trial (s/trial) and on YAHPO-Gym $\sim$ 7 s/trial relative to ParEGO. Since we use surrogate benchmarks, we do not know the evaluation cost. This overhead is fixed and independent of the evaluation cost, which in HPO settings ranges from minutes to hours. Under a pessimistic 20 s/trial overhead, HPI-ParEGO remains favorable whenever it matches ParEGO's performance with fewer trials (e.g., 400 vs. 500) and the per-trial evaluation cost exceeds 80 s $(400 \times (20 + c) < 500 \times c$, where c denotes evaluation cost). This is normally true for HPO on modern DL.



(a) Results on the selected `LCBench` tasks.



(b) Results on the selected `rbv2_ranger` tasks.

Figure 6: Results of HPI-ParEGO compared to the baseline optimizers measured in IGD+ regret.

5 CONCLUSION

In this work, we proposed a novel hyperparameter optimization method based on ParEGO that incorporates dynamic HPI into the optimization. Our method leverages ParEGO's scalarization mechanism to identify the most influential hyperparameters under varying trade-offs between objectives. Through extensive experiments on PyMOO and YAHPO-Gym, spanning synthetic and real-world multi-objective tasks, we demonstrated that HPI-ParEGO consistently outperforms or matches standard ParEGO and most of our baseline optimizers in low- to moderate-dimensional tasks. This supports our hypothesis: Many hyperparameters contribute little to performance under specific objective trade-offs, and dynamically filtering them during optimization helps the optimizer focus on the relevant subspace. Calculating HPI adds a bit of computational overhead, but it is fixed on a given configuration space and negligible relative to typical evaluation costs in real HPO settings, making the method practical for HPO on modern DL (see Section 4.6 for a detailed analysis).

A full theoretical understanding of when importance-guided filtering benefits MOO remains open. Our preliminary understanding suggests that in applications with many unimportant hyperparameters, HPI-ParEGO is more effective at quickly identifying them. However, a highly variable HPI with many slightly important hyperparameters is harder to learn with a reduced optimization budget. Nevertheless, our results clearly show that HPI should not remain a post-hoc diagnostic tool, but can be used during optimization, as previous work has done for SOO (Wang et al., 2021), even in MOO. This opens up a new class of resource-efficient MOO methods that can be important for Green AI and Green AutoML, emphasizing lower compute costs and reduced energy consumption without sacrificing accuracy (Schwartz et al., 2019; Tornede et al., 2023). By focusing on the most relevant hyperparameters, our method reduces computational costs and makes HPO more accessible, though practitioners should carefully consider objective selection to avoid encoding biases or misuse.

5.1 LIMITATIONS

Our method relies on an accurate surrogate model for reliable HPI estimation, which can be a bottleneck in early iterations (Wever et al., 2026) and data-scarce scenarios, despite our mitigation via interleaving random configurations and delayed reduction (via the initialization phase). However, the three-phase schedule assumes a reasonably set budget; for very few trials, the τ-schedule might not work well because the reduction occurs too early. Moreover, the performance and surrogate model quality may degrade as the dimensionality of the configuration space increases, especially since HyperSHAP can incur excessive overhead, making it too slow in practice for very large-scale HPO problems modeling full ML pipelines, e.g., Kotthoff et al. (2017); Feurer et al. (2022). Our scalability analysis (Appendix F) is limited to synthetic benchmarks; whether the same holds for high-dimensional real-world HPO tasks remains untested. Furthermore, we selected YAHPO-Gym tasks with varying HPI across trade-offs to test the mechanism under the conditions it targets. This may favor our method, but we chose the tasks before any optimization was done.

5.2 FUTURE WORK

There are several opportunities for future work to overcome these limitations: (i) exploration of more robust and uncertainty-aware HPI estimation techniques, for example, using stochastic Shapley values (Chau et al., 2023); (ii) meta-learning of HPI priors from previous HPO tasks; or (iii) reducing τ in a data-driven way. Furthermore, a common approach to reduce the high cost of model training is to use multi-fidelity optimization (Swersky et al., 2014; Falkner et al., 2018; Li et al., 2020; Bohdal et al., 2023), but it is an open problem how much information, e.g., on HPI, can be leveraged from cheap training runs. Results from Zimmer et al. (2021) suggest that HPI might be stable enough at least on simple DNNs. Moreover, our method is currently tailored to scalarization-based optimizers such as ParEGO, which limits generality. Extending the method to other MOO paradigms, such as nondominated-sorting-based evolutionary algorithms, requires rethinking how HPI is applied. Finally, we do not cover many-objective optimization with more than three objectives, which requires large optimization budgets, but the sequential nature and ParEGO's surrogate model are better suited to constrained optimization budgets. Therefore, extensions of our idea to NSGA-III (Deb & Jain, 2014) and the population-based contexts are promising future work.

AI USE STATEMENT

In this work, we used generative AI tools to aid and polish writing and to draft sections of the paper. We have not used generative AI tools for retrieval and discovery, or research ideation and execution. Generating synthetic datasets and proving mathematical claims are not applicable to this work. We have reviewed all AI-assisted work. All authors verified the content and text for correctness. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

HPI-ParEGO is specified in Algorithm 1 and Section 3, and the experimental setup is in Section 4.1 and Appendices A–D. All benchmarks and frameworks are publicly available. Anonymized code to reproduce all experiments is provided as supplementary material.

REFERENCES

M. Adachi, B. Planden, D. A. Howey, M. A. Osborne, S. Orbell, N. Ares, K. Muandet, and S. L. Chau. Looping in the human: Collaborative and explainable Bayesian optimization. In *Proc. of AISTATS'24*, 2024.

T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama. Optuna: A next-generation Hyperparameter Optimization framework. In *Proc. of KDD'19*, pp. 2623–2631, 2019.

S. Basu, F. Hutter, and D. Stoll. Multi-objective hyperparameter optimization in the age of deep learning. *arXiv preprint arXiv:2511.08371*, 2025.

R. Bellman, Rand Corporation, and Karreman Mathematics Research Collection. *Dynamic Programming*. Rand Corporation Research Study. Princeton University Press, 1957. ISBN 9780691079516.

C. Benjamins, H. Graf, S. Segel, D. Deng, T. Ruhkopf, L. Hennig, S. Basu, N. Mallik, E. Bergman, D. Chen, F. Clément, A. Tornede, M. Feurer, K. Eggensperger, F. Hutter, C. Doerr, and M. Lindauer. carps: A framework for comparing n hyperparameter optimizers on m benchmarks. *Transactions of Machine Learning Research (TMLR)*, 2026.

J. Bergstra and Y. Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13:281–305, 2012.

A. Biedenkapp, M. Lindauer, K. Eggensperger, C. Fawcett, H. Hoos, and F. Hutter. Efficient parameter importance analysis via ablation with surrogates. In *Proc. of AAAI'17*, pp. 773–779, 2017.

A. Biedenkapp, J. Marben, M. Lindauer, and F. Hutter. CAVE: Configuration assessment, visualization and evaluation. In *Proc. of LION'18*, 2018.

B. Bischl, M. Binder, M. Lang, T. Pielok, J. Richter, S. Coors, J. Thomas, T. Ullmann, M. Becker, A.-L. Boulesteix, D. Deng, and M. Lindauer. Hyperparameter optimization: Foundations, algorithms, best practices, and open challenges. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, pp. e1484, 2023.

J. Blank and K. Deb. PyMOO: Multi-objective optimization in Python. *IEEE Access*, 8:89497–89509, 2020.

O. Bohdal, L. Balles, M. Wistuba, B. Ermis, C. Archambeau, and G. Zappella. PASHA: Efficient HPO and NAS with progressive resource allocation. In *Proc. of ICLR'23*, 2023.

E. Bradford, A. Schweidtmann, and A. Lapkin. Efficient multiobjective optimization employing Gaussian processes, spectral sampling and a genetic algorithm. *J. Glob. Optim.*, 71(2):407–438, 2018.

S. L. Chau, K. Muandet, and D. Sejdinovic. Explaining the uncertain: Stochastic Shapley values for Gaussian process models. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, Red Hook, NY, USA, 2023. Curran Associates Inc.

S. Daulton, M. Balandat, and E. Bakshy. Differentiable expected hypervolume improvement for parallel Multi-Objective Bayesian optimization. In *Proc. of NeurIPS'20*, pp. 9851–9864, 2020.

S. Daulton, D. Eriksson, M. Balandat, and E. Bakshy. BONSAI: Bayesian optimization with natural simplicity and interpretability. *arXiv*, 2026.

K. Deb and H. Jain. An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, Part I: Solving problems with box constraints. *IEEE Transactions on Evolutionary Computation*, 18(4):577–601, 2014.

K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002.

K. Eggensperger, P. Müller, N. Mallik, M. Feurer, R. Sass, A. Klein, N. Awad, M. Lindauer, and F. Hutter. HPOBench: A collection of reproducible multi-fidelity benchmark problems for HPO. In *Proc. of NeurIPS'21 Datasets and Benchmarks Track*, 2021.

T. Elsken, J. Metzen, and F. Hutter. Efficient multi-objective Neural Architecture Search via lamarckian evolution. In *Proc. of ICLR'19*, 2019.

D. Eriksson and M. Jankowiak. High-dimensional Bayesian optimization with sparse axis-aligned subspaces. In *Proc. of UAI'21*, pp. 493–503, 2021.

D. Eriksson, M. Pearce, J. Gardner, R. Turner, and M. Poloczek. Scalable global optimization via local Bayesian optimization. In *Proc. of NeurIPS'19*, 2019.

S. Falkner, A. Klein, and F. Hutter. BOHB: Robust and efficient Hyperparameter Optimization at scale. In *Proc. of ICML'18*, pp. 1437–1446, 2018.

C. Fawcett and H. Hoos. Analysing differences between algorithm configurations through ablation. *Journal of Heuristics*, 22(4):431–458, 2016.

M. Feurer and F. Hutter. Hyperparameter Optimization. In F. Hutter, L. Kotthoff, and J. Vanschoren (eds.), *Automated Machine Learning: Methods, Systems, Challenges*, chapter 1, pp. 3 – 38. Springer, 2019. Available for free at http://automl.org/book.

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.

D. Fudenberg and J. Tirole. *Game Theory*. MIT press, 1991.

F. Hutter, H. Hoos, and K. Leyton-Brown. Identifying key algorithm parameters and instance features using forward selection. In *Proc. of LION'13*, pp. 364–381, 2013.

F. Hutter, H. Hoos, and K. Leyton-Brown. An efficient approach for assessing hyperparameter importance. In *Proc. of ICML'14*, pp. 754–762, 2014.

H. Ishibuchi, H. Masuda, Y. Tanigaki, and Y. Nojima. Modified distance calculation in generational distance and inverted generational distance. In *Evolutionary Multi-Criterion Optimization*, pp. 110–125. Springer International Publishing, 2015.

D. Jones, M. Schonlau, and W. Welch. Efficient global optimization of expensive black box functions. *Journal of Global Optimization*, 13:455–492, 1998.

J. D. Knowles. ParEGO: a hybrid algorithm with on-line landscape approximation for expensive multiobjective optimization problems. *IEEE Transactions on Evolutionary Computation*, 10(1): 50–66, 2006.

L. Kotthoff, C. Thornton, H. Hoos, F. Hutter, and K. Leyton-Brown. Auto-WEKA 2.0: Automatic model selection and hyperparameter optimization in WEKA. *Journal of Machine Learning Research*, 18:25:1–25:5, 2017.

J. Lee, S. Ahn, H. Kim, and J. Ruth Lee. Dynamic hyperparameter allocation under time constraints for Automated Machine Learning. *Intelligent Automation & Soft Computing*, 31(1):255–277, 2022.

L. Li, K. Jamieson, A. Rostamizadeh, E. Gonina, J. Ben-tzur, M. Hardt, B. Recht, and A. Talwalkar. A system for massively parallel hyperparameter tuning. In *Proc. of MLSys'20*, 2020.

M. Lindauer, K. Eggensperger, M. Feurer, A. Biedenkapp, D. Deng, C. Benjamins, T. Ruhkopf, R. Sass, and F. Hutter. SMAC3: A versatile bayesian optimization package for Hyperparameter Optimization. *Journal of Machine Learning Research*, 23(54):1–9, 2022.

S. Liu, Q. Feng, D. Eriksson, B. Letham, and E. Bakshy. Sparse Bayesian optimization. In *International Conference on Artificial Intelligence and Statistics*, pp. 3754–3774. PMLR, 2023.

J. Lévesque, C. Gagné, and R. Sabourin. Bayesian Hyperparameter Optimization for Ensemble Learning. In *Proc. of UAI'16*, pp. 437–446, 2016.

A. Morales-Hernández, I. Van Nieuwenhuyse, and S. Gonzalez. A survey on multi-objective hyperparameter optimization algorithms for machine learning. *Artificial Intelligence Review*, 56: 8043–8093, 2022.

Y. Ozaki, Y. Tanigaki, S. Watanabe, and M. Onishi. Multiobjective tree-structured parzen estimator for computationally expensive optimization problems. In *Proc. of GECCO'20*, pp. 533–541, 2020.

V. Pareto. *Manual of Political Economy*. Augustus M. Kelley, New York, 1971. Originally published in Italian as *Manuale di Economia Politica*, 1906.

F. Pfisterer, L. Schneider, J. Moosbauer, M. Binder, and B. Bischl. YAHPO Gym – an efficient multi-objective multi-fidelity benchmark for hyperparameter optimization. In *Proc. of AutoML Conf'22*. PMLR, 2022.

P. Probst, A. Boulesteix, and B. Bischl. Tunability: Importance of hyperparameters of machine learning algorithms. *Journal of Machine Learning Research*, 20(53):1–32, 2019.

J. Rapin and O. Teytaud. Nevergrad - a gradient-free optimization platform, 2018.

J. Rodemann, F. Croppi, P. Arens, Y. Sale, J. Herbinger, B. Bischl, E. Hüllermeier, T. Augustin, C. Walsh, and G. Casalicchio. Explaining Bayesian optimization by Shapley values facilitates human-AI collaboration. *CoRR*, abs/2403.04629, 2024.

R. Schmucker, M. Donini, V. Perrone, M. Zafar, and C. Archambeau. Multi-objective multi-fidelity hyperparameter optimization with application to fairness. In *MetaLearn'20*, 2020.

R. Schwartz, J. Dodge, N. A. Smith, and O. Etzioni. Green AI. *arXiv:1907.10597v3 [cs.CY]*, 2019.

L. S. Shapley. *A Value for N-Person Games*, pp. 307–318. Princeton University Press, Princeton, 1953.

Y. Shen and C. Kingsford. Computationally efficient high-dimensional Bayesian optimization via variable selection. In *Proc. of AutoML Conf'23*. PMLR, 2023.

J. Snoek, H. Larochelle, and R. Adams. Practical Bayesian optimization of machine learning algorithms. In *Proc. of NeurIPS'12*, pp. 2960–2968, 2012.

N. Srinivas, A. Krause, S. Kakade, and M. Seeger. Gaussian process optimization in the bandit setting: No regret and experimental design. In *Proc. of ICML'10*, pp. 1015–1022, 2010.

K. Swersky, J. Snoek, and R. Adams. Freeze-thaw Bayesian optimization. *arXiv:1406.3896 [stats.ML]*, 2014.

D. Theodorakopoulos, F. Stahl, and M. Lindauer. Hyperparameter importance analysis for multi-objective AutoML. In *Proc. of ECAI'24*, pp. 1100–1107, 2024.

T. Tornede, A. Tornede, J. Hanselle, F. Mohr, M. Wever, and E. Hüllermeier. Towards green Automated Machine Learning: Status quo and future directions. *Journal of Artificial Intelligence Research*, 77:427–457, 2023.

J. van Rijn and F. Hutter. Hyperparameter importance across datasets. In *Proc. of KDD'18*, pp. 2367–2376, 2018.

C. Wang, H. Wang, C. Zhou, and H. Chen. ExperienceThinking: Constrained hyperparameter optimization based on knowledge and pruning. *Knowledge-Based Systems*, 223:106602, 2021.

R. Wang, I. Nabney, and M. Golbabaee. Grouped sequential optimization strategy - the application of hyperparameter importance assessment in deep learning. In *Proceedings of Machine Learning Research*, 2025.

Z. Wang, F. Hutter, M. Zoghi, D. Matheson, and N. de Freitas. Bayesian optimization in a billion dimensions via random embeddings. *Journal of Artificial Intelligence Research*, 55:361–387, 2016.

H. Weerts, F. Pfisterer, M. Feurer, K. Eggensperger, E. Bergman, N. Awad, J. Vanschoren, M. Pechenizkiy, B. Bischl, and F. Hutter. Can fairness be automated? Guidelines and opportunities for fairness-aware AutoML. *Journal of Artificial Intelligence Research*, 79:639–677, 2024.

M. Wever, M. Muschalik, F. Fumagalli, and M. Lindauer. HyperSHAP: Shapley values and interactions for hyperparameter importance. In *Proc. of AAAI'26*, 2026.

M. Wistuba, N. Schilling, and L. Schmidt-Thieme. Hyperparameter search space pruning - A new component for sequential model-based Hyperparameter Optimization. In *Proc. of ECML/PKDD'15*, pp. 104–119, 2015.

Q. Zhang and H. Li. MOEA/D: A multiobjective evolutionary algorithm based on decomposition. *IEEE Trans. Evol. Comput.*, 11(6):712–731, 2007.

Y. Zhao, L. Wang, K. Yang, T. Zhang, T. Guo, and Y. Tian. Multi-objective optimization by learning space partition. In *International Conference on Learning Representations*, 2021.

L. Zimmer, M. Lindauer, and F. Hutter. Auto-Pytorch: Multi-fidelity metalearning for efficient and robust AutoDL. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 43:3079–3090, 2021.

E. Zitzler, K. Deb, and L. Thiele. Comparison of multiobjective evolutionary algorithms: Empirical results. *Evolutionary computation*, 8(2):173–195, 2000.

A SELECTION OF YAHPO-GYM TASKS

We selected the tasks for `LCBench` and `rbv2_ranger` by randomly drawing 20 different weight vectors and applying HyperSHAP to the surrogate models provided by YAHPO-Gym (Pfisterer et al., 2022), combining multiple objectives as originally defined by Knowles (2006). For `rbv2_ranger`, we consider accuracy and memory as objectives; for `LCBench`, we run tasks based on validation accuracy and time. To this end, we assessed the tunability of these weightings relative to the default configuration as a baseline. Based on these results, we apply our hyperparameter selection method to identify a subset of hyperparameters to tune for a specific weight vector. Afterward, we selected the top 13 tasks for `LCBench` and `rbv2_ranger` that maximize the number of distinct hyperparameter subsets across different weight vectors. Additionally, we paid attention to the fact that the subsets of hyperparameters actually differ across different weightings. Although these criteria may favor HPI-aware methods, tasks were fixed prior to all experiments.

B HARDWARE SETUP

The computations were conducted on a high-performance computer with nodes equipped with 2× AMD Milan 7763 (2×64 cores) and 256 GiB of RAM each, running Red Hat Enterprise Linux Ootpa and Slurm. Individual runs took between 1 and 24 CPU-hours depending on the optimizer and benchmark, with 1 core and 16GB RAM allocated per run. Our main results were evaluated on 25 tasks × 6 optimizers × 10 seeds (1,500 runs), plus an additional 630 runs for the remaining experiments, totaling to 2,130 runs and an estimated 25,560 CPU-hours (at ∼12 CPU-hours/run on average).

C NUMBER OF TRIALS PER TASK

In the YAHPO-Gym paper (Pfisterer et al., 2022), the number of trials is defined as number of trials $= 20 + 40\sqrt{D}$ with D being the number of dimensions. For our `LCBench`, `rbv2_ranger`, and `PyMOO`, we used five times as many trials. We ran the PyMOO tasks `ZDT1-3` only three times as many trials, since the optimizer overhead per iteration is higher for these tasks. Table 1 lists the resulting number of trials per task (rounded up to the nearest multiple of 5).

Benchmark	Task/Scenario	Trials
PyMOO	`ZDT1-3`	720
PyMOO	`ZDT4, ZDT6`	735
YAHPO-Gym	`LCBench`	630
YAHPO-Gym	`rbv2_ranger`	670

Table 1: Number of trials per task.

D ABLATION STUDY VARIANTS

In an ablation study, we tried variants of our algorithm. All options are shown in Table 5a in Section 4.4 and will be described here in more detail. All comparisons were performed with the HPI-ParEGO optimizer setup (HPI method: HyperSHAP, fixing strategy: incumbent, τ-schedule: Symmetric-0.8, r=0.1, u=10). For the ablation experiments, we used three tasks from `LCBench` and three tasks from `rbv2_ranger`, which were not part of the final evaluation. The summary of the results for HV regret is shown in Figure 7.

D.1 HPI IMPLEMENTATION DETAILS

To integrate HyperSHAP, we use the HyperSHAP library (Wever et al., 2026) version 0.0.6, which is available as a PyPI package and on GitHub (https://github.com/automl/hypershap). It generates some random configurations as input to the current surrogate model and uses these to estimate the Shapley

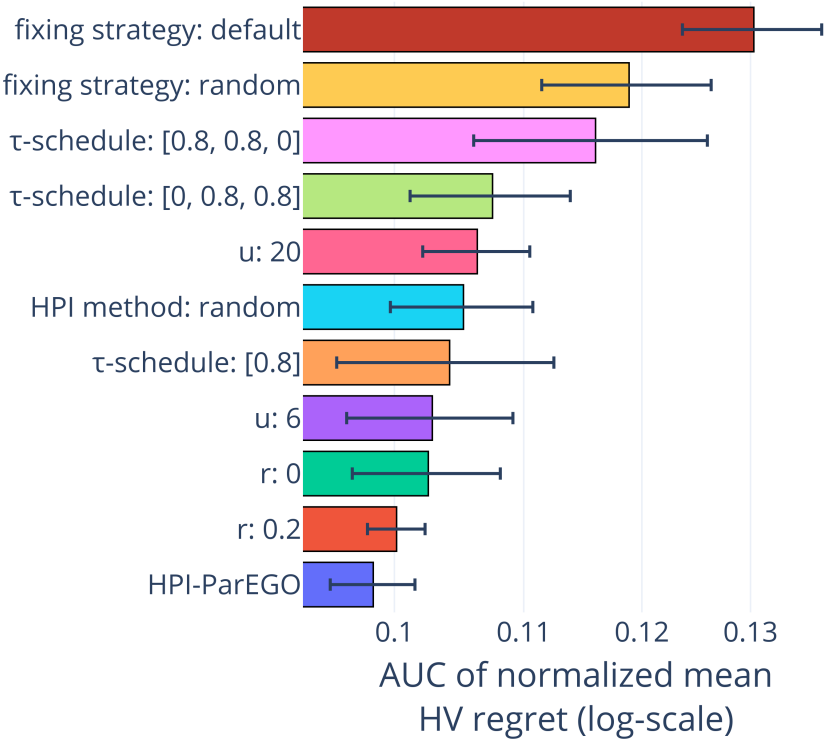
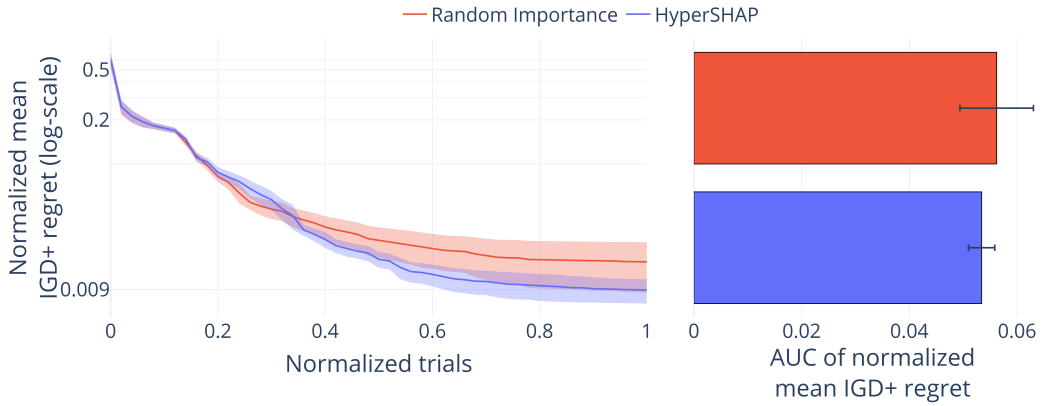


Figure 7: AUC of HV regret for ablating over different algorithm components.

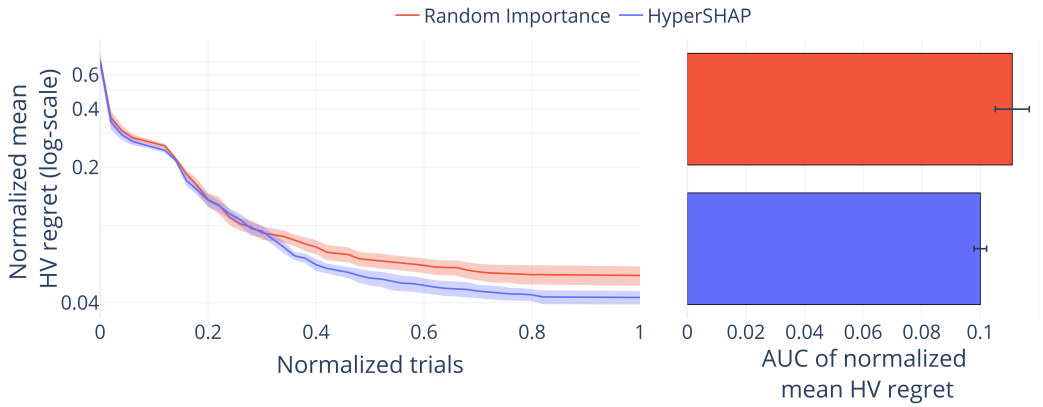
values. We use the tunability estimation with the index set to Shapley values and an order of 1, as calculating higher-order interactions would be too resource-intensive for this method, given that the importance is recalculated many times.

D.2 Comparison of Random Importance against HyperSHAP

To investigate whether the performance gains of HPI-ParEGO are primarily driven by dimensionality reduction, we compare random selection of a set of important hyperparameters with HPI-based selection using HyperSHAP. As shown in Figure 8, HyperSHAP-based selection consistently outperforms random importance. This suggests that HyperSHAP can identify relevant hyperparameters already early in the optimization process, even with limited data.



(a) IGD+ regret.

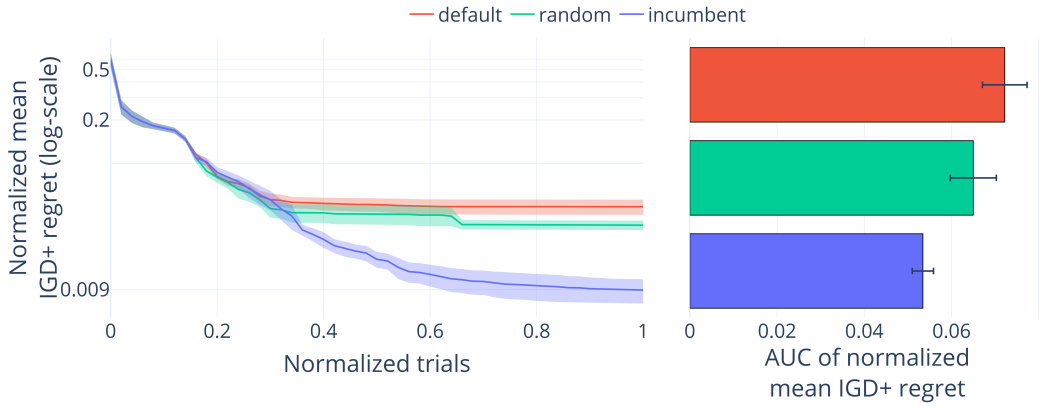


(b) HV regret.

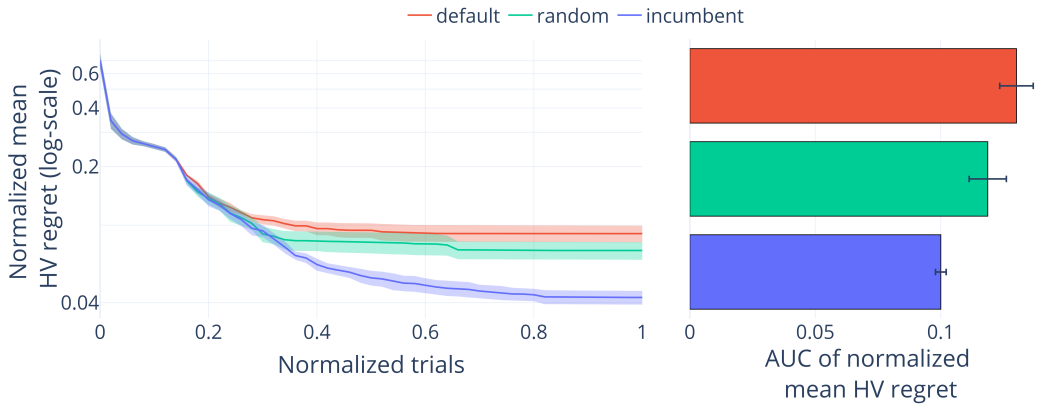
Figure 8: Ablation results for comparing random importance values against HPI-ParEGO (HPI-ParEGO in blue).

D.3 Hyperparameter Fixing Strategy

We tested fixing the constant values of the unimportant hyperparameters to the hyperparameter's default value, the configuration with the current incumbent value in the run history $\mathcal{H}$ for the given scalarization, or a randomly sampled value. Additionally, for HyperSHAP, that choice (incumbent, default, or random) was also used as a reference configuration. Figure 9 displays the difference between the approaches. A large improvement can be seen by using the current incumbent value.



(a) IGD+ regret.

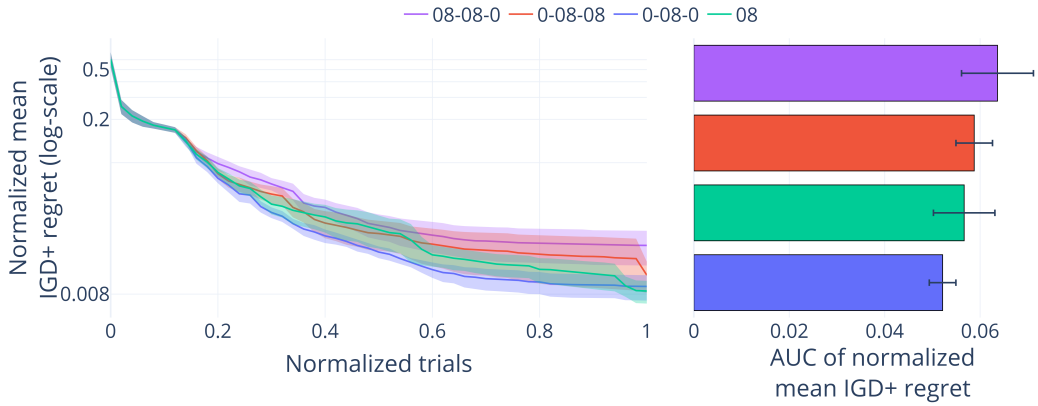


(b) HV regret.

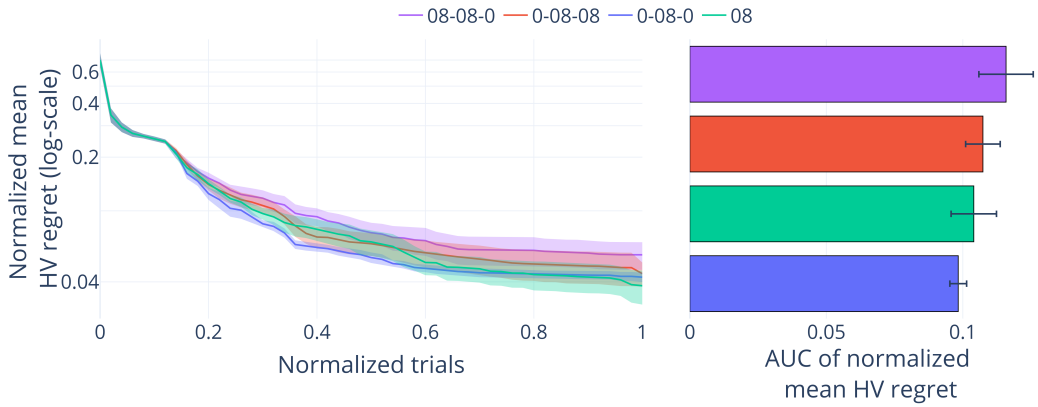
Figure 9: Ablation results for different constant values to reduce the configuration space (HPI-ParEGO in blue).

D.4 THRESHOLD τ-SCHEDULE

We ablated over different τ-schedules. Besides the constant schedule, they make changes after every 33% of the optimization trials. The motivation for the initialization phase is that we start with (almost) all hyperparameters to first learn which ones are important and improve the surrogate model, and then focus more and more on the important ones. The motivation behind the consolidation phase is to allow the optimizer to tune the less important hyperparameters in the end as well. For the symmetric threshold, we combine the intuition by letting the optimizer explore the space initially, then narrowing it down, and finally allowing more exploration at the end to ensure the entire space is covered. Figure 10 shows the results of the threshold comparison, which shows that the symmetric schedule is the best. However, the constant threshold performs similarly at later stages, especially for HV regret. Nevertheless, it is definitely worse until around 60% of the trials. This means that using the full space at the beginning is more important than using it at the end.



(a) IGD+ regret.

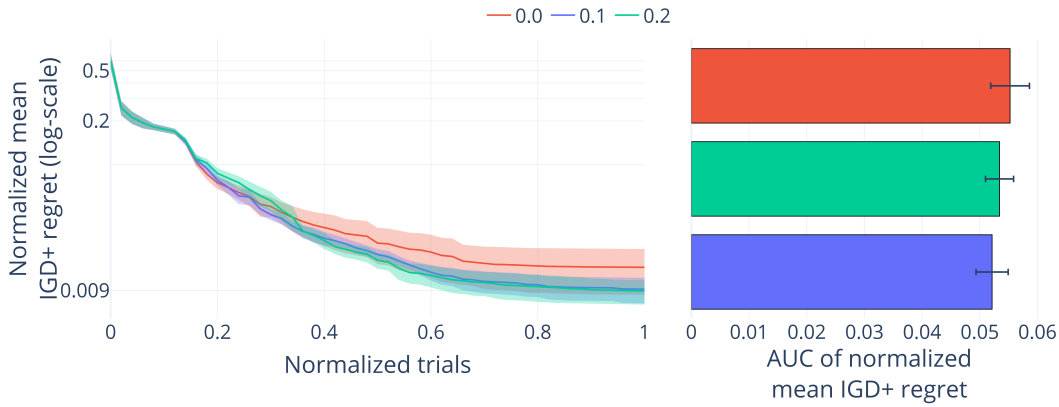


(b) HV regret.

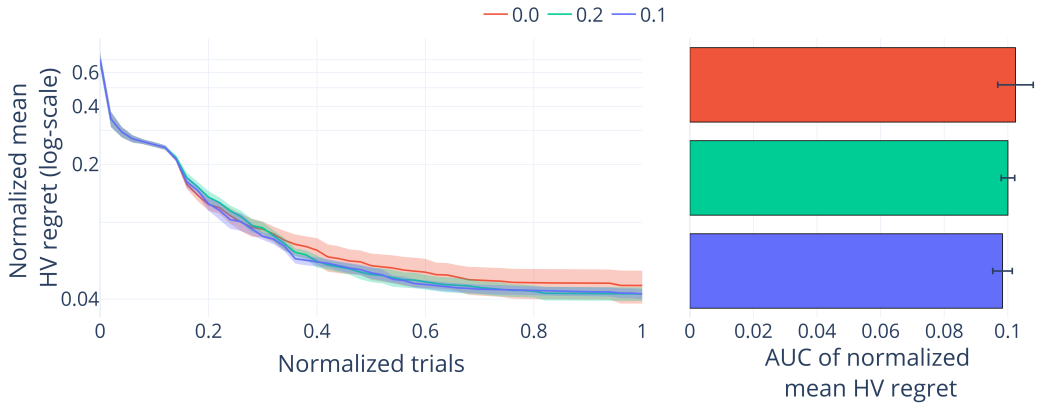
Figure 10: Ablation results for different thresholds. 0 means no reduction (HPI-ParEGO in blue).

D.5 Random Chance for HPI

We also considered different values for the random chance r that a random configuration of the original configuration space will be evaluated. We tested: 0.0, 0.1, 0.2. The results are shown in Figure 11. It can be seen that sometimes using random, full configurations has a small but positive effect.



(a) IGD+ regret.

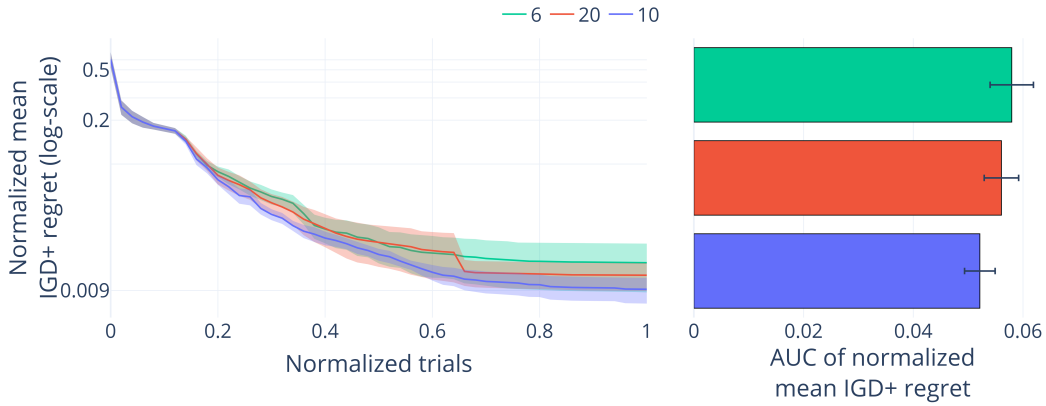


(b) HV regret.

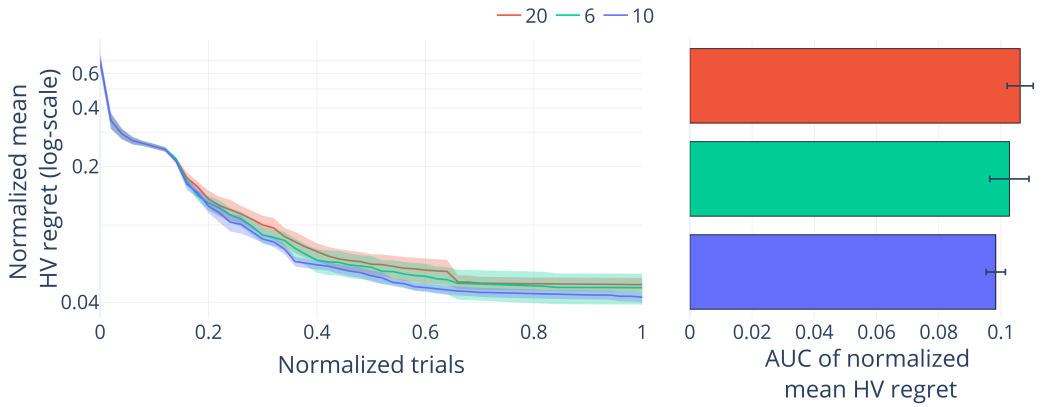
Figure 11: Ablation results for different random chances r (HPI-ParEGO in blue).

D.6 Number of Iterations for a Weight Update

The intuition behind increasing the number of iterations for a weight update, compared to the original ParEGO algorithm, is that the optimization process can focus on a scalarization for more iterations and thereby improve the current scalarization. This also allows for leveraging the reduced configuration space several times before it is reset. We tested different values: 6, 10, and 20. Figure 12 shows that every 10 iterations is a good value.



(a) IGD+ regret.

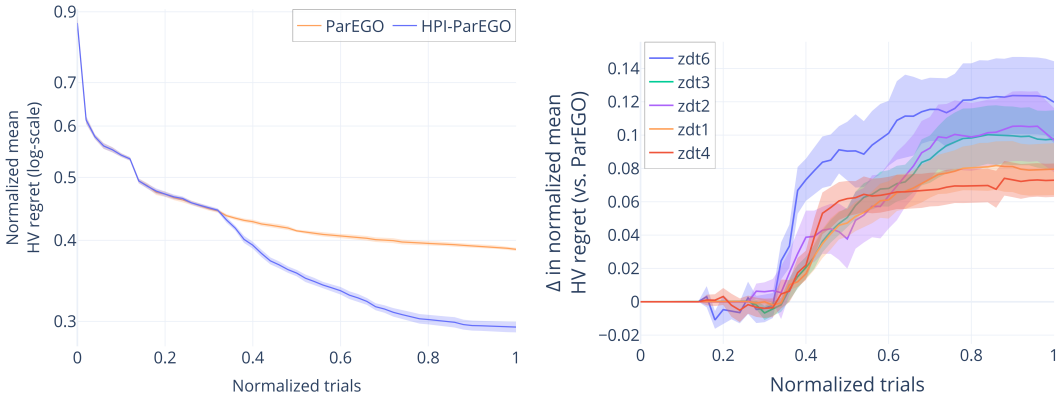


(b) HV regret.

Figure 12: Ablation results for different numbers of iterations for weight updates u (HPI-ParEGO in blue).

E FURTHER RESULTS ON PYMOO

Figure 13 shows the comparison of HPI-ParEGO against the ParEGO optimizer as well as the task difference on the PyMOO tasks for the HV regret indicator. Similar to the IGD+ results, HPI-ParEGO clearly outperforms ParEGO.

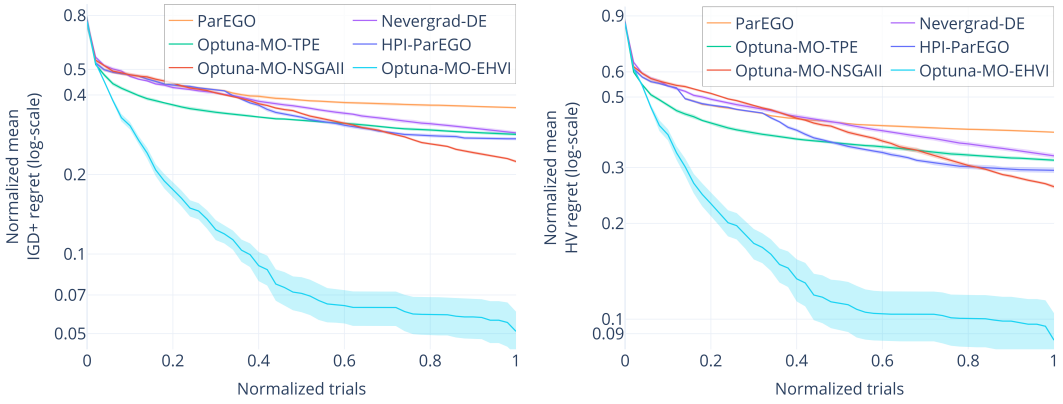


(a) The HPI-ParEGO optimizer compared to the ParEGO optimizer.

(b) Difference in mean HV regret with standard error between ParEGO and HPI-ParEGO. Values > 0 indicate a better performance of HPI-ParEGO.

Figure 13: Results on the PyMOO tasks.

Figure 14 shows the results of HPI-ParEGO against all baselines on the PyMOO tasks. EHVI performs best, likely because the simple, smooth ZDT problems allow its HV-based acquisition function to efficiently exploit the Pareto front by directly optimizing the HV performance measure and, for some tasks, the ZDT problems yield simple solutions (one objective at 0, the other at 1). As the YAHPO-Gym results show, this advantage does not translate to more complex real-world problems. Following EHVI, MO-TPE leads initially, but from around halfway, HPI-ParEGO is competitive with the remaining baselines, with only NSGA-II pulling ahead in the final stages.



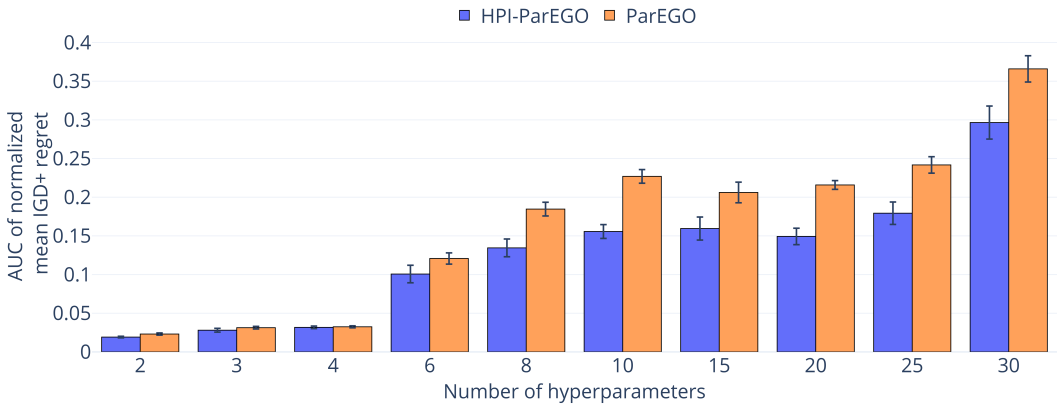
(a) IGD+ regret.

(b) HV regret.

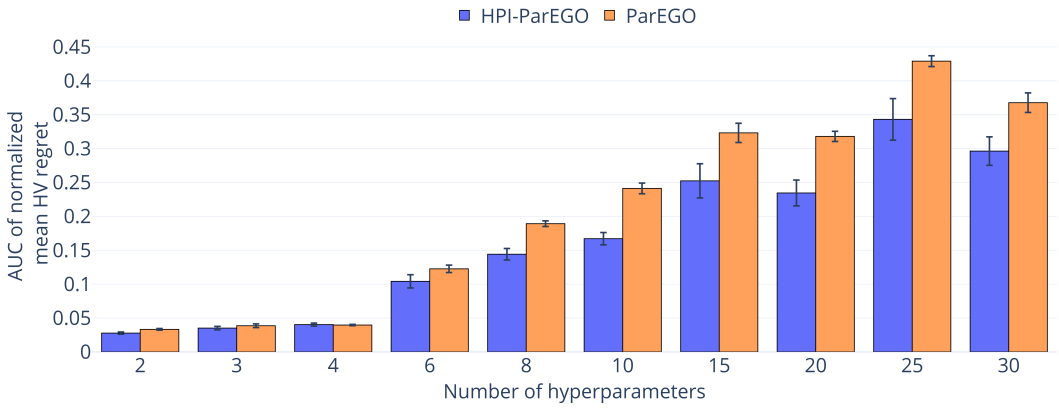
Figure 14: Results on the PyMOO tasks for the HPI-ParEGO optimizer compared to all baselines.

F SCALABILITY

In this experiment, we want to determine how our optimizer behaves across different dimensionalities. We compare ParEGO against HPI-ParEGO on the PyMOO task `zdt2`, which has 30 dimensions. In this task, the first objective $f_1(\mathbf{x}) = x_1$ depends solely on x_1, while the second objective depends on all other variables through a sum $g(\mathbf{x}) = 1 + \frac{9}{(n-1)} \cdot \sum_{i=2}^{n} x_i$. Because each variable $x_2, \ldots, x_{30}$ contributes only a fraction to this sum, their individual influence is small relative to x_1, making `zdt2` a natural testbed for scalability under varying hyperparameter importance. We repeat the experiment and include more irrelevant hyperparameters until we reach all 30 with this schedule: 2, 4, 6, 8, 10, 15, 20, 25, 30. Figure 15 shows that for the first dimensions, the two optimizers perform similarly. As dimensionality scales up, the advantage of HPI-based selection becomes evident.



(a) AUC of mean IGD+ regret.

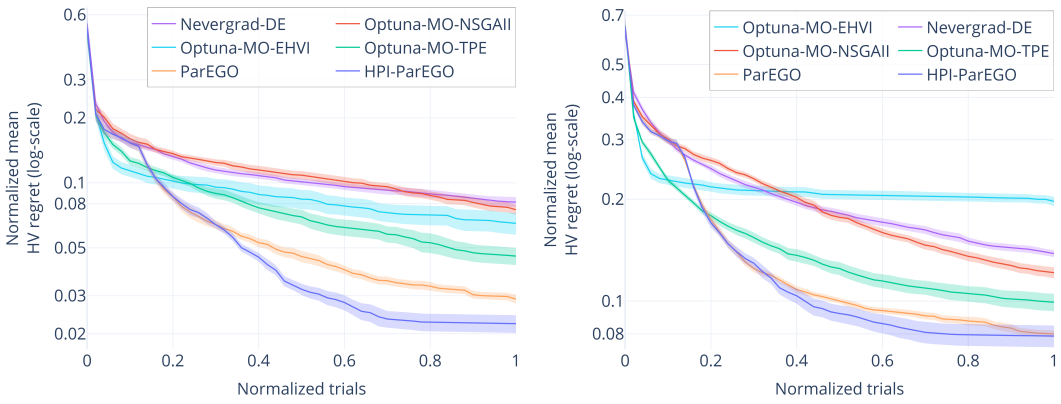


(b) AUC of mean HV regret.

Figure 15: Results of HPI-ParEGO vs ParEGO for different dimensionalities on the PyMOO task `zdt2`.

G ADDITIONAL RESULTS FOR THE SELECTED YAHPO-GYM TASKS

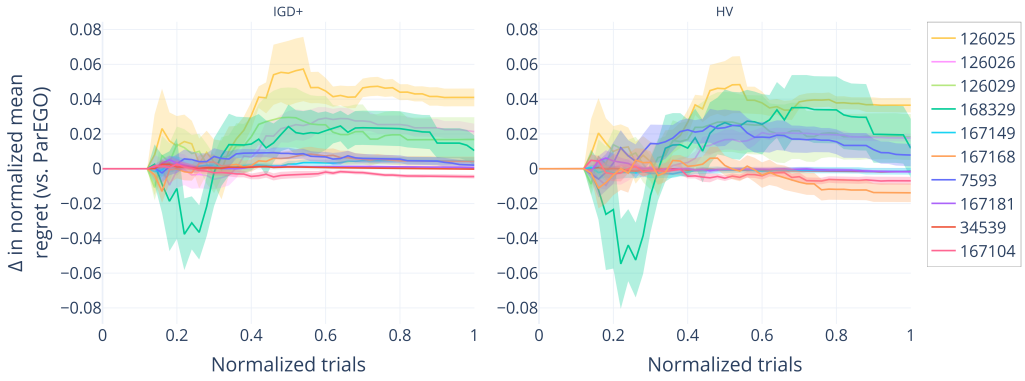
Figure 16 shows the comparison of HPI-ParEGO against the baseline optimizers for the HV regret indicator. The results are very similar to the IGD+ indicator. Figure 17 compares HPI-ParEGO and the ParEGO baseline across the different scenarios from YAHPO-Gym for HV regret and IGD+ regret. The difference between the two metrics is minimal and only visible in a few tasks. The results reveal that the benefits of incorporating HPI are not uniform: while HPI-ParEGO achieves clear improvements on `LCBench`, its advantage is less consistent on `rbv2_ranger`, where, for some tasks, it clearly performs better, while for others ParEGO performs better. This variation highlights that the effectiveness of HPI-guided optimization can depend on task characteristics, such as the dimensionality of the configuration space and the presence of conditions within it. As shown in Figure 6 in Section 4.5, HPI-ParEGO performs overall better than the standard ParEGO.



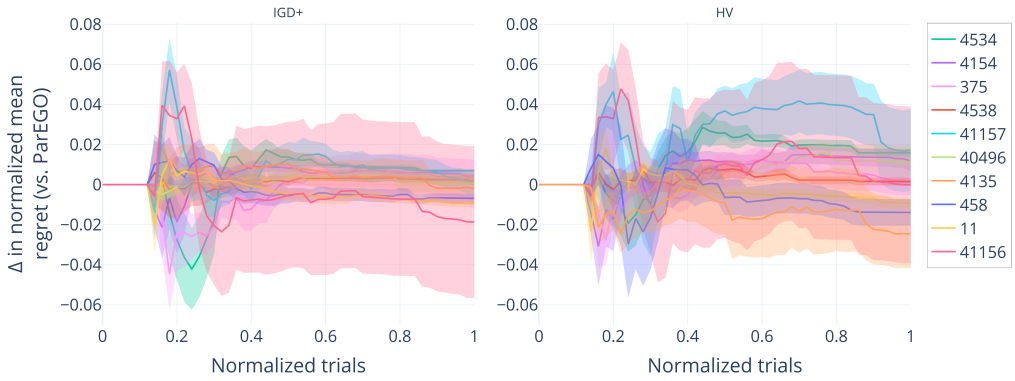
(a) Results on the selected `LCBench` tasks.

(b) Results on the selected `rbv2_ranger` tasks.

Figure 16: Results of HPI-ParEGO compared to the baseline optimizers measured in HV regret.



(a) Difference in mean regret with standard error for the `LCBench` tasks.



(b) Difference in mean regret with standard error for the `rbv2_ranger` tasks.

Figure 17: Difference between ParEGO and HPI-ParEGO on the selected YAHPO-Gym tasks. Values > 0 indicate a better performance of HPI-ParEGO.