A Limitations

Naturally, our approach suffers from limitations, which we describe below together with potential remedies.

First, as we rely on showing Pareto fronts to the user, we make the implicit assumption of optimizing the hyperparameters of MO-ML algorithms which optimize only two loss functions $\mathcal{L}_1$ and $\mathcal{L}_2$. Although our approach would in principle also work with more loss functions, obtaining the corresponding comparison results from users would be much more complicated as showing a user a 3-dimensional, let alone more dimensional, Pareto fronts is infeasible. A possible remedy to this problem would be to deploy dimensionality reduction techniques for multi-objectives settings (Deb and Saxena 2005) such as PCA (Jolliffe and Cadima 2016), LDA (Balakrishnama and Ganapathiraju 1998) mapping higher dimensional Pareto fronts to two dimensions for showing them to the user.

Second, the feature representation as described in Eq. **??** could be improved in general to be independent of the number of loss functions. This would simplify the application of the subsequent object ranking algorithm as the replacement and imputation strategy might be discarded. On the downside, depending on the feature representation, the positional encoding, which we assume to be of importance for the learner, might get lost.

Third, in Sec. **??** we note that, ideally, the sampled Pareto fronts used to generated pairwise comparisons with the help of the user should be sampled such that the space of possible Pareto fronts is covered to avoid generalization issues of the learned utility function. However, our current approach does not actively account for this. In fact, achieving such a coverage is a challenging task as we cannot directly sample the corresponding Pareto fronts, but instead rely on sampling hyperparamter configurations leading to the Pareto front. A potential remedy for this problem could be to meta-learn a model mapping from a hyperparameter $\boldsymbol{\lambda}$ configuration to the feature vector $\boldsymbol{f}_{P_{\mathcal{D}_{val}}(A(\mathcal{D}_{train},\boldsymbol{\lambda}))}$ representing the Pareto front obtained by running A with λ. If we had such a model with a quick evaluation time, we could leverage rejection sampling and sample until we have a sufficient coverage in the feature space.

Lastly, our approach is tailored towards MO-ML. However, single objective ML is much more common in practice, limiting the applicability of our approach. As such, a generalization of our approach to doing MO-HPO of single objective ML algorithms is desirable.

B Formal Definition of Pareto Front Quality Indicators

HV: Hypervolume, it quantifies the volume of the front by merging the hypercubes determined by each of its solutions $h \in H$ and the reference point r (i.e., the worst possible solution), formally:

$$HV(H) = \beta\left(\bigcup_{h\in H}\{x \mid h \prec x \prec r\}\right)$$

where β denotes the Lebesgue measure. This ranges from 0 to 1, the highest the better.

SP: Spacing, it is the most popular uniformity indicator and it gauges the variation of the distance between solutions in a set;

$$SP(H) = \sqrt{\frac{1}{N-1}\sum_{h\in H}(\bar{d} - d_1(h, H/h))^2}$$

where $\bar{d}$ is the mean of all the $d_1(h, H/h) : h \in H$ and $d_1(h, H/h)$ means the L^1 norm distance (Manhattan distance) of h to the set H/h. A SP value of zero indicates all members of the solution set are spaced equidistantly on the basis of Manhattan distance.

MS: Maximum Spread, it is a widely used spread indicator, and it measures the range of a solution set by considering the maximum extent on each objective:

$$MS(H) = \sqrt{\sum_{j=1}^{m}\max_{h,h'\in H}(\mathcal{L}_j(h) - \mathcal{L}_j(h'))^2}$$

where m denotes the number of objectives. MS is to be maximized; the higher the value, the better extensity to be claimed.

R2: this indicator measures the proximity to a specific reference point r via the Chebyshev norm:

$$R2(H) = \min_{h\in H} d_{\text{inf}}(h, r)$$

Given m objectives, $d_{\text{inf}}(h, r)$ corresponds to $\max_{i=1}^{m}(\mid \mathcal{L}_i(h) - \mathcal{L}_i(r))$. Lower values translate into Pareto fronts closer to the reference point, taken as the ideal solution.

C Fisher-Jenks Algorithm

The Fisher-Jenks algorithm addresses the problem of determining the best arrangement of ordered values into different buckets, also known as *natural breaks optimization* because it effectively highlights significant transitions in the underlying distribution. We leverage this algorithm to identify similar Pareto fronts in the set of the Preliminary sampling, i.e., that should be ranked equally in a global ranking. The process starts by trying to divide the samples into two groups, evaluating all possible break points by the Jenks optimization function. This is the core of the algorithm because it drives towards the optimal buckets and consists of a trade-off between minimizing within-bucket variance and maximizing between-bucket variance. Once the optimal breakpoint is found, the data is divided into two groups based on this breakpoint. The process is then repeated recursively for each of these groups until a predetermined number of buckets is reached. By exploiting a simple elbow method, it is possible to determine the optimal number of buckets, i.e., choosing the "knee of a curve" as a cutoff point, where diminishing returns are no longer worth the additional cost. Given n samples to split into k buckets, the complexity of the algorithm would be $O(k \cdot n^2)$. Yet, nowadays, finding breaks for 15 classes for a data array of 7 million unique values now takes 20 seconds on commodity hardware. In these experiments, we have only 40 samples to arrange in – approximately – 30 buckets, depending on the indicator.

D Evaluation

In the following, we provide details about the employed MO-ML algorithm (D.1), implementational set-up (D.2), the LCBench configuration space (D.3), end-to-end performance varying the number of pairwise comparisons (D.4).

D.1 MO-ML Algorithm

Since our aim is to collect Pareto fronts and get preferences on their shapes, we are interested in MO-ML algorithms that provide different Paretos as the hyperparameters change. Fortunately, according to the case study, an extensive literature is readily available. For instance, in the domain of fairness in AI, MO-ML implementations yield a Pareto front through mitigation, i.e., a hyper-parameter with significant impact is varied to achieve fairer outcomes.

In our use case, Green AutoML, Pareto fronts are handed over by observing *accuracy* and *power consumption* during model training of DNNs. In practice, it is known that the *number of epochs* has an obvious influence on the objectives, hence, on the shape of the retrieved Pareto front. Figure 1 shows an example: given a hyperparameter configuration, we perform a grid search over the number of epochs and obtain snapshots of the DNN learning curve (Mohr and van Rijn 2022).

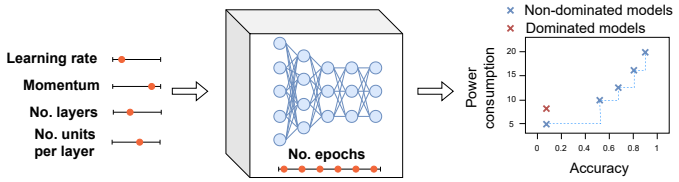

Figure 1: Working example of the DNN wrapper: the grid search over the number of epochs allows us to have a Pareto front as output.

D.2 Implementational Details

The code and results of the performed experiments are available via GitHub[1] with the corresponding documentation to run them. There is no need for the user to locally build the project, we deployed a Docker Image that instantiates a pre-built container with all the needed Python dependencies and runs the experiments. In particular, we leverage: the package cs-rank (Pfannschmidt, Gupta, and Hüllermeier 2018, 2019) as a basis for our preference learning models, the package pymoo(Blank and Deb 2020) for the quality indicator implementations, and jenkspy[2] for the Fisher-Jenks algorithm (Fisher 1958; Jenks 1967). Finally, we exploit the well-known HPO tool SMAC (Hutter, Hoos, and Leyton-Brown 2011; Lindauer et al. 2022) to apply Bayesian optimization.

The experiments were tested varying three different seeds on an AMD Ryzen 5 3600 6-Core Processor with 64 GB. The total computation time was 21984 seconds (around 6 hours), for an overall power consumption of 520 W.

D.3 LCBench Configuration Space

All runs feature funnel-shaped MLP nets and use SGD with cosine annealing without restarts. Overall, 7 parameters were sampled at random (4 float, 3 integer). These are reported in Table 1. More details can be found at https://github.com/automl/LCBench.

[1]https://github.com/automl/interactive-mo-ml

[2]https://github.com/mthh/jenkspy

Hyperparameter	Type	Range	Distribution
Batch size	Integer	[16, 512]	Log
Learning rate	Float	[1e-4, 1e-1]	Log
Momentum	Float	[0.1, 0.99]	Linear
Weight decay	Float	[1e-5, 1e-1]	Linear
No. layers	Integer	[1, 5]	Linear
No. units per layer	Integer	[64, 1024]	Log
Dropout	Float	[0.0, 1.0]	Linear

Table 1: Search space of LCBench.

D.4 Additional Experimental Results

Robustness In the main paper, we reported the performance of the end-to-end approach when the preference learning models are trained atop 28 pairwise comparisons (1 out of the 5 available cross-validation folds). We also tested our approach by varying such available pairwise comparisons. Tables 2 to 5 shows the performance when the models used from 2 to 5 folds, that would be – respectively – 56, 84, 112, and 140 pairwise comparisons.

We can observe negligible fluctuations in performance among the tables, thus the insights presented in the main paper remain consistent. This indicates that our approach is robust; indeed, when leveraging just 28 pairwise comparisons, we are able to get performance as good as when using 140 comparisons.

PB\IB	$HV\uparrow$		$SP\downarrow$		$MS\uparrow$		$R2\downarrow$	
$HV\uparrow$	0.75 (±0.18)	**0.77 (±0.17)**	**0.75 (±0.18)**	0.52 (±0.24)	**0.75 (±0.18)**	0.52 (±0.21)	0.75 (±0.18)	**0.77 (±0.16)**
$SP\downarrow$	**0.02 (±0.02)**	0.03 (±0.02)	0.02 (±0.02)	**0.01 (±0.0)**	**0.02 (±0.02)**	0.04 (±0.03)	**0.02 (±0.02)**	0.04 (±0.02)
$MS\uparrow$	**0.6 (±0.11)**	0.19 (±0.08)	**0.6 (±0.11)**	0.19 (±0.12)	0.6 (±0.11)	**0.65 (±0.06)**	**0.6 (±0.11)**	0.23 (±0.11)
$R2\downarrow$	0.24 (±0.18)	**0.22 (±0.16)**	**0.24 (±0.18)**	0.47 (±0.23)	**0.24 (±0.18)**	0.45 (±0.21)	0.24 (±0.18)	**0.21 (±0.16)**

Table 2: Comparison between indicator-based HPO (i.e., IB, columns) and preference-based HPO (i.e., PB, rows). The preference learning model is trained using 56 pairwise comparisons.

PB\IB	$HV\uparrow$		$SP\downarrow$		$MS\uparrow$		$R2\downarrow$	
$HV\uparrow$	0.76 (±0.17)	**0.77 (±0.17)**	**0.76 (±0.17)**	0.52 (±0.24)	**0.76 (±0.17)**	0.52 (±0.21)	0.76 (±0.17)	**0.77 (±0.16)**
$SP\downarrow$	**0.02 (±0.02)**	0.03 (±0.02)	0.02 (±0.02)	**0.01 (±0.0)**	**0.02 (±0.02)**	0.04 (±0.03)	**0.02 (±0.02)**	0.04 (±0.02)
$MS\uparrow$	**0.6 (±0.09)**	0.19 (±0.08)	**0.6 (±0.09)**	0.19 (±0.12)	0.6 (±0.09)	**0.65 (±0.06)**	**0.6 (±0.09)**	0.23 (±0.11)
$R2\downarrow$	**0.22 (±0.17)**	**0.22 (±0.16)**	**0.22 (±0.17)**	0.47 (±0.23)	**0.22 (±0.17)**	0.45 (±0.21)	0.22 (±0.17)	**0.21 (±0.16)**

Table 3: Comparison between indicator-based HPO (i.e., IB, columns) and preference-based HPO (i.e., PB, rows). The preference learning model is trained using 84 pairwise comparisons.

Comparison to a Multi-Objective HPO algorithm A multi-objective HPO algorithm will need a MO metric. The main contribution of our paper is to show how to get around choosing this MO metric explicitly by using an interactive data-driven preference-based alternative. Therefore, a direct comparison is not fair from our point of view. Nevertheless, we performed a comparison per the reviewers' request showing that we perform competitively or better (Table 6).

PB\IB	$HV \uparrow$	$SP \downarrow$	$MS \uparrow$	$R2 \downarrow$
$HV \uparrow$	0.76 (±0.17) \ **0.77 (±0.17)**	**0.76 (±0.17)** \ 0.52 (±0.24)	**0.76 (±0.17)** \ 0.52 (±0.21)	0.76 (±0.17) \ **0.77 (±0.16)**
$SP \downarrow$	**0.02 (±0.02)** \ 0.03 (±0.02)	0.02 (±0.02) \ **0.01 (±0.0)**	**0.02 (±0.02)** \ 0.04 (±0.03)	**0.02 (±0.02)** \ 0.04 (±0.02)
$MS \uparrow$	**0.6 (±0.1)** \ 0.19 (±0.08)	**0.6 (±0.1)** \ 0.19 (±0.12)	0.6 (±0.1) \ **0.65 (±0.06)**	**0.6 (±0.1)** \ 0.23 (±0.11)
$R2 \downarrow$	0.24 (±0.17) \ **0.22 (±0.16)**	**0.24 (±0.17)** \ 0.47 (±0.23)	**0.24 (±0.17)** \ 0.45 (±0.21)	0.24 (±0.17) \ **0.21 (±0.16)**

Table 4: Comparison between indicator-based HPO (i.e., IB, columns) and preference-based HPO (i.e., PB, rows). The preference learning model is trained using 112 pairwise comparisons.

PB\IB	$HV \uparrow$	$SP \downarrow$	$MS \uparrow$	$R2 \downarrow$
$HV \uparrow$	0.76 (±0.17) \ **0.77 (±0.17)**	**0.76 (±0.17)** \ 0.52 (±0.24)	**0.76 (±0.17)** \ 0.52 (±0.21)	0.76 (±0.17) \ **0.77 (±0.16)**
$SP \downarrow$	**0.01 (±0.02)** \ 0.03 (±0.02)	**0.01 (±0.02)** \ **0.01 (±0.0)**	**0.01 (±0.02)** \ 0.04 (±0.03)	**0.01 (±0.02)** \ 0.04 (±0.02)
$MS \uparrow$	**0.6 (±0.11)** \ 0.19 (±0.08)	**0.6 (±0.11)** \ 0.19 (±0.12)	0.6 (±0.11) \ **0.65 (±0.06)**	**0.6 (±0.11)** \ 0.23 (±0.11)
$R2 \downarrow$	0.23 (±0.17) \ **0.22 (±0.16)**	**0.23 (±0.17)** \ 0.47 (±0.23)	**0.23 (±0.17)** \ 0.45 (±0.21)	0.23 (±0.17) \ **0.21 (±0.16)**

Table 5: Comparison between indicator-based HPO (i.e., IB, columns) and preference-based HPO (i.e., PB, rows). The preference learning model is trained using 140 pairwise comparisons.

Indicator	Us \ ParEGO
$HV \uparrow$	**0.76 (±0.17)** \ 0.75 (±0.16)
$SP \downarrow$	**0.01 (±0.01)** \ 0.04 (±0.04)
$MS \uparrow$	**0.61 (±0.09)** \ 0.20 (±0.16)
$R2 \downarrow$	0.23 (±0.16) \ **0.22 (±0.16)**

Table 6: Comparison with ParEGO, performance are measured w.r.t. the indicator picked by the user.

Non-linear RankSVM We performed the end-to-end approach with a non-linear implementation of RankSVM (Chen et al. 2017) as a preference model, but cannot show the complete results here due to size constraints. Even though the ranking performance results for Kendall's Tau are worse (Figure 2), the end-to-end performance is comparable (Tables 8 to 11). It seems that the larger ranking error does not influence the performance of the HPO process too much. We believe that the more powerful hypothesis class of the non-linear SVM suffers from overfitting leading to the larger ranking error. Overall, this validates our conclusion from Section 5.2.

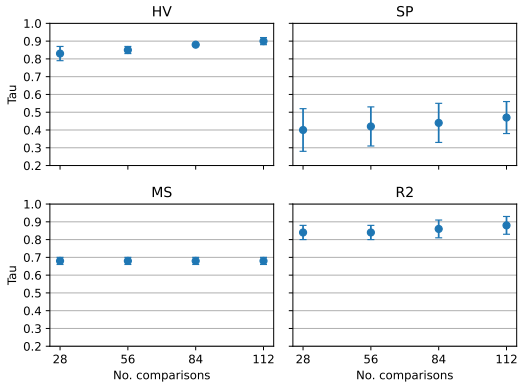

Figure 2: Kendall's Tau of the preference learning models trained with non-linear RankSVM.

PB\IB	$HV \uparrow$		$SP \downarrow$		$MS \uparrow$		$R2 \downarrow$	
$HV \uparrow$	0.75 (±0.17)	**0.77 (±0.17)**	**0.75 (±0.17)**	0.52 (±0.24)	**0.75 (±0.17)**	0.52 (±0.21)	0.75 (±0.17)	**0.77 (±0.16)**
$SP \downarrow$	**0.02 (±0.01)**	0.03 (±0.02)	0.02 (±0.01)	**0.01 (±0.0)**	**0.02 (±0.01)**	0.04 (±0.03)	**0.02 (±0.01)**	0.04 (±0.02)
$MS \uparrow$	**0.62 (±0.09)**	0.19 (±0.08)	**0.62 (±0.09)**	0.19 (±0.12)	0.62 (±0.09)	**0.65 (±0.06)**	**0.62 (±0.09)**	0.23 (±0.11)
$R2 \downarrow$	0.23 (±0.16)	**0.22 (±0.16)**	**0.23 (±0.16)**	0.47 (±0.23)	**0.23 (±0.16)**	0.45 (±0.21)	0.23 (±0.16)	**0.21 (±0.16)**

Table 7: Comparison between indicator-based HPO (i.e., IB, columns) and preference-based HPO (i.e., PB, rows). The preference learning model is trained using the non-linear RankSVM implementation and 28 pairwise comparisons.

PB\IB	$HV \uparrow$		$SP \downarrow$		$MS \uparrow$		$R2 \downarrow$	
$HV \uparrow$	0.73 (±0.19)	**0.77 (±0.17)**	**0.73 (±0.19)**	0.52 (±0.24)	**0.73 (±0.19)**	0.52 (±0.21)	0.73 (±0.19)	**0.77 (±0.16)**
$SP \downarrow$	**0.02 (±0.01)**	0.03 (±0.02)	0.02 (±0.01)	**0.01 (±0.0)**	**0.02 (±0.01)**	0.04 (±0.03)	**0.02 (±0.01)**	0.04 (±0.02)
$MS \uparrow$	**0.63 (±0.07)**	0.19 (±0.08)	**0.63 (±0.07)**	0.19 (±0.12)	0.63 (±0.07)	**0.65 (±0.06)**	**0.63 (±0.07)**	0.23 (±0.11)
$R2 \downarrow$	0.23 (±0.17)	**0.22 (±0.16)**	**0.23 (±0.17)**	0.47 (±0.23)	**0.23 (±0.17)**	0.45 (±0.21)	0.23 (±0.17)	**0.21 (±0.16)**

Table 8: Comparison between indicator-based HPO (i.e., IB, columns) and preference-based HPO (i.e., PB, rows). The preference learning model is trained using the non-linear RankSVM implementation and 56 pairwise comparisons.

PB\IB	$HV \uparrow$		$SP \downarrow$		$MS \uparrow$		$R2 \downarrow$	
$HV \uparrow$	0.76 (±0.16)	**0.77 (±0.17)**	**0.76 (±0.16)**	0.52 (±0.24)	**0.76 (±0.16)**	0.52 (±0.21)	0.76 (±0.16)	**0.77 (±0.16)**
$SP \downarrow$	**0.02 (±0.01)**	0.03 (±0.02)	0.02 (±0.01)	**0.01 (±0.0)**	**0.02 (±0.01)**	0.04 (±0.03)	**0.02 (±0.01)**	0.04 (±0.02)
$MS \uparrow$	**0.61 (±0.1)**	0.19 (±0.08)	**0.61 (±0.1)**	0.19 (±0.12)	0.61 (±0.1)	**0.65 (±0.06)**	**0.61 (±0.1)**	0.23 (±0.11)
$R2 \downarrow$	0.23 (±0.16)	**0.22 (±0.16)**	**0.23 (±0.16)**	0.47 (±0.23)	**0.23 (±0.16)**	0.45 (±0.21)	0.23 (±0.16)	**0.21 (±0.16)**

Table 9: Comparison between indicator-based HPO (i.e., IB, columns) and preference-based HPO (i.e., PB, rows). The preference learning model is trained using the non-linear RankSVM implementation and 84 pairwise comparisons.

PB\IB	$HV \uparrow$		$SP \downarrow$		$MS \uparrow$		$R2 \downarrow$	
$HV \uparrow$	0.76 (±0.17)	**0.77 (±0.17)**	**0.76 (±0.17)**	0.52 (±0.24)	**0.76 (±0.17)**	0.52 (±0.21)	0.76 (±0.17)	**0.77 (±0.16)**
$SP \downarrow$	**0.01 (±0.01)**	0.03 (±0.02)	0.01 (±0.01)	**0.01 (±0.0)**	**0.01 (±0.01)**	0.04 (±0.03)	**0.01 (±0.01)**	0.04 (±0.02)
$MS \uparrow$	**0.62 (±0.09)**	0.19 (±0.08)	**0.62 (±0.09)**	0.19 (±0.12)	0.62 (±0.09)	**0.65 (±0.06)**	**0.62 (±0.09)**	0.23 (±0.11)
$R2 \downarrow$	0.23 (±0.17)	**0.22 (±0.16)**	**0.23 (±0.17)**	0.47 (±0.23)	**0.23 (±0.17)**	0.45 (±0.21)	0.23 (±0.17)	**0.21 (±0.16)**

Table 10: Comparison between indicator-based HPO (i.e., IB, columns) and preference-based HPO (i.e., PB, rows). The preference learning model is trained using the non-linear RankSVM implementation and 112 pairwise comparisons.

PB\IB	$HV \uparrow$		$SP \downarrow$		$MS \uparrow$		$R2 \downarrow$	
$HV \uparrow$	0.75 (±0.18)	**0.77 (±0.17)**	**0.75 (±0.18)**	0.52 (±0.24)	**0.75 (±0.18)**	0.52 (±0.21)	0.75 (±0.18)	**0.77 (±0.16)**
$SP \downarrow$	**0.01 (±0.01)**	0.03 (±0.02)	0.01 (±0.01)	**0.01 (±0.0)**	**0.01 (±0.01)**	0.04 (±0.03)	**0.01 (±0.01)**	0.04 (±0.02)
$MS \uparrow$	**0.62 (±0.09)**	0.19 (±0.08)	**0.62 (±0.09)**	0.19 (±0.12)	0.62 (±0.09)	**0.65 (±0.06)**	**0.62 (±0.09)**	0.23 (±0.11)
$R2 \downarrow$	0.24 (±0.17)	**0.22 (±0.16)**	**0.24 (±0.17)**	0.47 (±0.23)	**0.24 (±0.17)**	0.45 (±0.21)	0.24 (±0.17)	**0.21 (±0.16)**

Table 11: Comparison between indicator-based HPO (i.e., IB, columns) and preference-based HPO (i.e., PB, rows). The preference learning model is trained using the non-linear RankSVM implementation and 140 pairwise comparisons.

References

Balakrishnama, S.; and Ganapathiraju, A. 1998. Linear Discriminant Analysis - A Brief Tutorial. *ISIP*, 18(1998): 1–8.

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

Chen, K.; Li, R.; Dou, Y.; Liang, Z.; Lv, Q.; et al. 2017. Ranking support vector machine with kernel approximation. *Computational intelligence and neuroscience*, 2017.

Deb, K.; and Saxena, D. K. 2005. On Finding Pareto-optimal Solutions Through Dimensionality Reduction for Certain Large-dimensional Multi-objective Optimization Problems. *KanGAL*, 2005011: 1–19.

Fisher, W. 1958. On Grouping for Maximum Homogeneity. *JASA*, 53(284): 789–798.

Hutter, F.; Hoos, H.; and Leyton-Brown, K. 2011. Sequential Model-Based Optimization for General Algorithm Configuration. In *Proc. of LION'11*, 507–523.

Jenks, G. 1967. The Data Model Concept in Statistical Mapping. *IYC*, 7: 186–190.

Jolliffe, I.; and Cadima, J. 2016. Principal Component Analysis: A Review and Recent Developments. *PTRS*, 374(2065).

Lindauer, M.; Eggensperger, K.; Feurer, M.; Biedenkapp, A.; Deng, D.; Benjamins, C.; Ruhkopf, T.; Sass, R.; and Hutter, F. 2022. SMAC3: A Versatile Bayesian Optimization Package for Hyperparameter Optimization. *JMLR*, 23(54): 1–9.

Mohr, F.; and van Rijn, J. 2022. Learning Curves for Decision Making in Supervised Machine Learning–A Survey. *arXiv:2201.12150 [cs.LG]*.

Pfannschmidt, K.; Gupta, P.; and Hüllermeier, E. 2018. Deep Architectures for Learning Context-dependent Ranking Functions. *arXiv:1803.05796 [stat.ML]*.

Pfannschmidt, K.; Gupta, P.; and Hüllermeier, E. 2019. Learning Choice Functions: Concepts and Architectures. *arXiv:1901.10860 [stat.ML]*.