Multi-Objective Hyper-Parameter Optimization for Quality Estimation of Machine Translation

Jakob Hauser
STUDENT NUMBER: 2029657

THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
BACHELOR OF SCIENCE IN COGNITIVE SCIENCE & ARTIFICIAL INTELLIGENCE
DEPARTMENT OF COGNITIVE SCIENCE & ARTIFICIAL INTELLIGENCE
SCHOOL OF HUMANITIES AND DIGITAL SCIENCES
TILBURG UNIVERSITY

Thesis committee:

Supervisor: Dr. Dimitar Shterionov
Second Reader: Dr. Gonzalo Nápoles

Tilburg University
School of Humanities and Digital Sciences
Department of Cognitive Science & Artificial Intelligence
Tilburg, The Netherlands
January 2021

Preface

Thanks to my supervisor and my colleagues for supporting me on this journey and providing great advice. Thanks to my friends Max and Stefano for their advice and support. Thanks to Matteo for providing me not only with the computing hardware required to complete this project but also for just being a great human being. Thanks to my family and friends in Tilburg, back home, and wherever else for supporting me on a journey a lot bigger than this bachelor's thesis and making everything possible, in their own ways.

All in all, Thank you so much, everyone!

Multi-Objective Hyper-Parameter Optimization for Quality Estimation of Machine Translation

Jakob Hauser

Hyper-parameter optimization (HPO) is an essential part of most Machine Learning (ML) pipelines. Currently, the predominant usage of HPO for ML falls in the category of Single Objective Optimization (SOO). While the SOO approach of HPO usually benefits from lower runtime and better convergence, it restricts the tuning to only one requirement. In most cases, such an optimization fails to reflect the real-world criteria of the implementation. It is surprising that SOO is still the predominant approach for tuning ML models, while in many other disciplines (e.g. Aerospace Engineering) optimizing for only one of the requirements would be unacceptable. Usually, when designing a product or a process, the goal is to find an optimum across many conflicting objectives. Also in real-world software deployments, Deep Learning models are supposed to strike a balance between raw predictive performance and computational complexity. To achieve this the hyperparameters of Deep Learning models have to be optimized. Therefore, we choose to explore HPO using Multi-Objective Optimization (MOO) in terms of the aforementioned criteria. We evaluate 4 different (both multi and single-objective) optimization algorithms, and investigate their performance in regards to optimizing a Quality Estimation of Machine Translation model. Our chosen optimization criteria are predictive performance and floating-point operations for model inference (used as a proxy measure for computational complexity). While MOO shows some promise the findings of our investigation are inconclusive.

1. Introduction

Hyper-parameter optimization (HPO) is an essential part of most Machine Learning (ML) pipelines. Currently, the predominant usage of HPO for ML falls in the category of Single Objective Optimization (SOO). While the SOO approach of HPO usually benefits from lower runtime and better convergence, it restricts the tuning to only one. In most cases, such an optimization fails to reflect the real-world criteria of the implementation. It is surprising that SOO is still the predominant approach for tuning ML models, while in many other disciplines (e.g. Aerospace Engineering) optimizing for only one of the requirements would be unacceptable. Usually, when designing a product or a process, the goal is to find an optimum across many conflicting objectives. Also in real-world software deployments, Deep Learning models are supposed to strike a balance between raw predictive performance and computational complexity. To achieve this the hyperparameters of Deep Learning models have to be optimized. Therefore, we choose to explore HPO using Multi-Objective Optimization (MOO) in terms of the aforementioned criteria.

While the approach of using MOO for HPO should be applicable to any variant of Deep Learning, we focus on evaluating Multi-Objective Optimization for Quality

Estimation systems. Quality Estimation is the task of assessing the quality of a machine-translated text with neither human intervention nor a reference (human) translation. The model we use is tasked to estimate sentence level Human-targeted Translation Edit Rate scores (HTER)(Snover et al., 2006) and is based on the approach presented in Shterionov et al. (2019). We evaluate 4 different optimization algorithms (both multi and single-objective),and investigate their performance in regards to optimizing a Quality Estimation of Machine Translation model. Our chosen optimization criteria are predictive performance and floating-point operations for model inference (used as a proxy measure for computational complexity). In regard to this, we optimize our model's hyperparameters in terms of the Pearson correlation coefficient between the predicted HTER and the ground truth HTER score, and an approximation of the floating-point operations (FLOPS) required for inference. We use FLOPS as a proxy measure for the computational complexity of our model.

While MOO has been used to solve related problems such as Neural Architecture Search (Lu et al., 2019; Elsken et al., 2018), Feature Selection, and tuning the hyperparameters of statistical learning algorithms (Binder et al., 2020; Horn & Bischl, 2016), little work has been done to evaluate the utility of MOO for HPO of Deep Learning techniques.

We hope that our work can shed light on the potential of applying MOO for state-of-the-art Deep Learning tasks. Furthermore, we hope that our work will be useful to (i) researchers wanting to further improve QE systems, (ii) researchers interested in applying MOO to other domains in ML, and (iii) researchers searching for a methodological framework for comparing MOO algorithms in the context of HPO for Deep Learning.

In the proposed work we plan to investigate the potential advantages of two Multi-Objective Optimization algorithms (NSGA-II, MOTPE) for Quality Estimation (QE) of Machine Translation (MT). We will attempt to answer the following related question and sub-questions in our work:

- A. **How does multi-objective optimization perform relative to single-objective optimization, for the task of hyper-parameter optimization of quality estimation systems for machine translation?**

  - SQ1. **Which of the two proposed multi-objective optimization algorithms performs better with respect to hyperparameter optimization of quality estimation systems?**:

    We task 2 different MOO algorithms to generate configurations for our system. We then compare the algorithms based on the hypervolume indicator (Zitzler & Thiele, 1999).

  - SQ2. **How do quality estimation system configurations found by multi-objective optimization algorithms compare to systems found by our baseline search algorithms?**:

    We compare the set of solutions found by the MOO algorithms with the random sampling and the random search baseline.

  - SQ3. **How does multi-objective optimization compare with single-objective optimization for quality estimation?**:

> We evaluate the usability of MOO for the QE task. We compare MOO with SOO algorithms both using single-objective and multi-objective comparison methods.

While our results are inconclusive for all but the last sub-question. The methodology we propose - when improved in the shortcomings we identified - should pave the way for greater and deeper investigations of MOO not only for QE systems but also for other types of deep learning models.

2. Background

In this section we first introduce some concepts, metrics, and algorithms for Multi-Objective Optimization. Then we provide a view on the landscape of the Quality Estimation for Machine Translation field and present a state-of-the-art QE architecture. We introduce a practical distinction between Neural Architecture Search (NAS) and hyperparameter optimization. Finally, we present some research related to our study and compare their work with our approach.

2.1 Multi-Objective Optimization

In the context of hyperparameter search, an optimization problem can be defined as in Eggensperger et al. (2015). The goal is to minimize an algorithm's loss function given as:

$$f(\boldsymbol{\lambda}) = \mathcal{L}(\boldsymbol{\lambda}, \mathcal{D}_{train}, \mathcal{D}_{valid}) \tag{1}$$

Where $\lambda_1, \ldots, \lambda_n$ each represent an configuration of a specific hyperparameter of the algorithm and $\boldsymbol{\lambda}$ the complete parameter configuration. $\mathcal{D}_{train}$ and $\mathcal{D}_{valid}$ denote the training and validation data respectively. $\Lambda_1, \ldots, \Lambda_n$ represent the domain of each hyper-parameter setting and $\boldsymbol{\Lambda}$ the complete search space. We can then define a multi-objective optimization HPO problem, with the objectives $f_1, \ldots, f_n$ as:

$$\begin{aligned} &min(f_1(\boldsymbol{\lambda}), f_1(\boldsymbol{\lambda}), \ldots, f_n(\boldsymbol{\lambda})) \\ &for\ \boldsymbol{\lambda} \in \boldsymbol{\Lambda} \end{aligned} \tag{2}$$

In nontrivial cases, where the objectives do not coincide, multi-objective problems have multiple Pareto optimal solutions. A single solution is Pareto optimal or non-dominated, if it cannot be improved in terms of one objective, without it being deteriorated in terms of at least one of the other objectives. When we construct a set of such mutually non-dominated solutions we call it the Pareto front (Ehrgott, 2005; Eggensperger et al., 2015). A common approach to solving multi-objective problems (MOPs) are methods like linear scalarization, where weights for each objective are defined as $\omega_1, \ldots, \omega_n$. Then we can redefine the MOP as a single objective optimization problem:

$$\begin{aligned} &min(\sum_{i=1}^{n} \omega_i f_i(\boldsymbol{\lambda})) \\ &for\ \boldsymbol{\lambda} \in \boldsymbol{\Lambda} \end{aligned} \tag{3}$$

While such an approach simplifies the search process greatly, it can only produce one Pareto optimal solution at a time and the weights for each objective have to be chosen before any optimization takes place. For this reason, we did not consider any of the approaches combining multiple objectives into one for our work.

2.1.1 Metrics for Multi-Objective Optimization. There exists a very diverse range of metrics for MOO. The authors in Riquelme et al. (2015) found that 54 different metrics were used in papers contributed to various Evolutionary Multi-Criterion Optimization (EMO) conferences in the time range from 2005 up until 2013 (Deb et al., 2019). These metrics for multi-objective optimization can be split into roughly three categories: **cardinality** metrics describing how many solutions an algorithm produces, **diversity** metrics describing the range and spread of the obtained solutions, and **convergence** metrics describing how close the produced Pareto front lies to the true Pareto optimal front. The most used metric reported by Riquelme et al. (2015) is the hypervolume indicator (HV). The hypervolume metric takes all three aforementioned categories into account. It requires that a reference point be specified from which it measures the area, volume, or hypervolume (depending on the number of objectives) covered by the proposed set of solutions (Zitzler & Thiele, 1999). One of the strengths of the hypervolume indicator is its Pareto compliance. Pareto compliance requires that the hypervolume of A is guaranteed to be greater than the hypervolume of B if the set of solutions in approximation A completely dominates the set corresponding to approximation B (Zitzler et al., 2007). Also in the scientific research reviewed for the purpose of this work hypervolume was the predominant metric used to compare MOO algorithms. As a consequence of this, establishing comparisons between different approaches becomes easier when the hypervolume indicator is used (provided the Pareto fronts and the reference point are reported). This is especially the case when MOO algorithms are tested on benchmarks like the ones generated by the Walking Fish Group (WFG) toolkit (Huband et al., 2005, 2006).

2.1.2 Algorithms for Multi-Objective Optimization. Multi-Objective Optimization is an important tool in a plethora of fields. It is used for optimizing supply chain management (Trisna et al., 2016), and aiding the design of neuromorphic neural network accelerators (Parsa et al., 2020) and aerospace vehicles (Hebbal et al., 2019). The amount of research conducted comparing different MOO and SOO algorithms with respect to HPO of Deep Learning models is scarce. In other fields, there has been quite some research done on this topic. Chiandussi et al. (2012) review different MOO and SOO methodologies for engineering applications and Hamdy et al. (2016) compare algorithms for the purpose of solving nearly-zero-energy-building design problems. A great deal of work has been done to compare MOO algorithms on non-application-specific benchmark problems, mostly paired with the introduction of new MOO algorithms. We selected the two MOO algorithms we present below for our work since they are readily available in Optuna a popular Python library specialized for hyperparameter tuning. To our knowledge, Optuna has the best MOO support amongst similar libraries(Akiba et al., 2019).

The non-dominated sorting genetic algorithm II (NSGA-II) introduced a few crucial improvements over its predecessor NSGA (Deb et al., 2002; Srinivas & Deb, 1994): The introduction of fast non-dominated sorting enabled the algorithm to have a worst-case complexity of $O(MN^2)$. The crowding distance comparison enabled the algorithm to produce a Pareto front approximation with a higher density than before, without requiring any additional parameters.

NSGA-II involves an iterative process of producing offspring from an initial population, then selecting the fittest individuals from both the parent and daughter generation for the new parent generation.The selection is based firstly on the non-domination rank and secondly on the crowding distance of the individuals. The inclusion of the parent generation in the selection procedure guarantees that no Pareto optimal solution gets discarded (unless the number of optimal solutions is bigger than the population size)(Deb et al., 2002; Srinivas & Deb, 1994). The properties of NSGA-II make it very fast and therefore it has an advantage for problems where the objective function evaluations are cheap. In return, NSGA-II is deemed to require more iterations to converge compared to other approaches (Binder et al., 2020).

The Multiobjective Tree-structured Parzen Estimator (MOTPE) is an extension to of a very prominent SOO algorithm, the Tree-structured Parzen Estimator (TPE) (Bergstra et al., 2011, 2013). The TPE algorithm excels at computationally extensive problems with big and complicated search spaces. It is commonly used for HPO of Deep Neural Networks and also is the default algorithm for Optuna and HyperOpt[1], two prominent HPO Python libraries (Akiba et al., 2019).

In their work, Ozaki et al. (2020) compare MOTPE with multiple surrogate assisted evolutionary algorithms (SAEAs), and also compare different hyperparameter configurations of MOTPE to assess its respective performance and behavior. All their test are carried out on the WFG benchmarks 4-9, in two different configurations with a different number of objectives and variables (Huband et al., 2005, 2006). Ozaki et al. (2020) show that the MOTPE algorithm converges faster and retains a higher diversity than the other evaluated algorithms. Furthermore, they show that besides the WFG 5 benchmark problem, MOTPE outperforms the other algorithms consistently for a variety of different settings for the hyperparameter γ but generally recommend the setting $\gamma = 0.10$.

2.2 Quality Estimation for Machine Translation

Quality Estimation encompasses the tasks related to predicting the quality of machine-generated translations without having access to the gold-standard/reference translations. Until recently feature-based approaches were dominant in the field, but by now they have been mostly superseded by end-to-end neural architectures. Regardless of them being outperformed by neural architectures, feature-based approaches are still used as robust baseline algorithms (Specia et al., 2018; Shterionov et al., 2019).

The Conference on Machine Translation (WMT) annually holds shared tasks in an effort to compare novel QE approaches and push the state-of-the-art further. Most recently the WMT 2020 sub-tasks encompassed: (Task 1) predicting Direct Assessment (DA) scores at the sentence level; (Task 2) predicting post-editing effort scores at both sentence and word level and (Task 3) predicting a score for an entire document.

Task 2 involves predicting the Human-targeted Translation Edit Rate (HTER) at the sentence level, and predicting labels (OK, BAD) for tokens and omission errors. HTER measures the edit distance between a reference translation and a non-edited machine translation (Specia et al., 2018, 2020; Snover et al., 2006).

The state-of-the-art in Quality Estimation is reported in the most recent findings report of the WMT conference. In 2020 there were 1374 systems submitted by 19 participating teams. The best performing model for the sentence-level Task 2 was submit-

1 https://github.com/hyperopt/hyperopt

ted by the HW-TSC team, scoring a Pearson correlation coefficient (PCC) of 0.758 for the English-German (En-De) language pairing (Wang et al., 2020; Specia et al., 2020). Their architecture follows the Predictor-Estimator approach (Kim et al., 2017). In this approach, the predictor is used for feature extraction and predicts the following words, while the estimator can be a task-specific classification or regression model component. As their Predictor component, Wang et. al use a standard Transformer trained on data from WMT translation tasks. As their Estimator component, they use a unified QE model, made up out of 3 classifiers for word-level QE and 1 regressor for the sentence-level. They use the weighted cross-entropy loss function for the classifications tasks and the mean squared error (MSE) for the regression task. The model is trained using a multi-task learning framework and the loss of each task is weighed and summed for backpropagation (Wang et al., 2020).

The approach to QE introduced by the HW-TSC team is very modern compared to the system we use in the present work. While we choose our QE system for greater ease of implementation, for future work it could be interesting to apply our methodology to more modern network architectures.

2.3 Practical distinction between NAS and HPO

Since some related works focus on the integration of MOO in Neural Architecture Search (NAS) pipelines, we find it important to shed some light on the differences between NAS and HPO. Neural Architecture Search and Hyperparameter Optimization are closely related. While NAS focuses on building complete neural architectures from a set of predefined building blocks, HPO is used to tune selected parameters of already existing architectures. True NAS is still infeasible for anything besides research purposes and some niche cases since the search spaces are vast and evaluations are very computationally expensive. While the present work focuses more on HPO, it also integrates some aspects of NAS: The search space includes a selection of 3 different attention mechanisms and we also tune the size of the embedding and hidden dimensions (Elsken et al., 2019).

2.4 Related Research

As stated before, there has been some research conducted on applying multi-objective optimization conducted to problems like Neural Architecture Search (NAS), Feature Selection, and the tuning of hyperparameters for statistical learning algorithms. In this section, we present the ones that are the most closely related to our work and briefly mention other notable ones.

In their work titled "Multi-Objective Hyperparameter Tuning and Feature Selection using Filter Ensembles" Binder et al. (2020) investigate simultaneous feature selection and hyperparameter tuning. They use two different approaches, a model-based approach and an evolutionary approach (based on NSGA-II), to train three different classifiers. Namely: A support vector machine (SVM) with a Gaussian kernel, an extreme gradient boosting model, and a kernelized k-nearest-neighbor classifier. The numeric-categorical hyperparameter search spaces for these models contain 3-5 variables. Comparing MOO with SOO the authors find that while MOO does not perform as well as SOO. Their MOO methods often find solutions with a much smaller number of features and only a small loss in predictive performance compared to Soo. The MOO algorithms are compared based on their hypervolume indicator scores (Zitzler & Thiele, 1999). Furthermore, Binder et al. (2020) report that the Evolutionary Algorithm (EA) does not perform as well as the Bayesian optimization-based approach, in both SOO and MOO scenarios. While the EA is more computationally efficient this would only be of any benefit when the evaluations of the objective functions are cheap (Binder et al., 2020). The approach of Binder et al. (2020) resembles the methodology of our work quite closely, while their optimization process is different and they report feature sparseness instead of FLOPS, the manner in which we investigate our results is based partly on their work.

Lu et al. (2019) propose NSGA-Net a MOO approach to NAS. Similar to our approach they use classification error and computational complexity as their performance metrics. They only report that their search algorithm finds networks with a lower error rate compared to other MOO approaches for NAS, but are unable to compare hypervolume measurements to other approaches. They lack the trade of frontiers of other implementations and some others use different objectives (Lu et al., 2019).

Notable other approaches are Elsken et al. (2018) who utilize a lamarckian evolution process to carry out NAS with predictive performance and number of parameters as objectives, Hsu et al. (2018) who propose a framework for MOO NAS with reinforcement learning. While these approaches still investigate MOO for ML techniques, both their objectives and optimization techniques are very different from the ones proposed in our work.

3. Methods

To evaluate the different optimization algorithms we let them select hyperparameter configurations of a neural network and collect information on how the trained network performs with respect to its predictive performance and the number of floating-point operations required for inference.

3.1 Quality Estimation Dataset

Our project relies on a dataset concatenated from multiple sources. We choose to combine the data released for the shared task of the WMT conference from the years 2016, 2017, 2018/2019, 2020, 2021. Specifically, we used the data provided for the shared tasks of sentence-level quality estimation via the prediction of HTER scores, in the language pairing English to German. Since our training set is quite small and the test set contained no gold-standard HTER labels, we split the data from the development set of each year to generate a labeled test set and appended the remaining data points to the training set. The complete breakdown of our dataset can be found in table 1.

3.2 SiameseQE System

For our experiments, we rely on the SiameseQE network as proposed in (Shterionov et al., 2019), a Siamese network with a bidirectional LSTM layer. The model is implemented in PyTorch and the code was provided to us by Dimitar Shterionov (Paszke et al., 2019).

The network encodes two sentences, the source, and target sentence, sequentially. It then calculates either the Euclidean or Cosine distance between the embedded information created in the two forward passes. We use the mean squared error to obtain the loss with respect to the predicted and the expected HTER score. The network can be configured with three different attention mechanisms: soft dot attention as used in the

Year	Domain	Translation Model	Total Amount	train/dev/test
2016	IT	SMT	15k	12k/1k/(blind)
2017	IT	SMT	25k	13k/1k/(blind)
2018/19	IT	NMT	~14k	13k/1k/(blind)
2020	Wikipedia	NMT	8k	7k /1k/(blind)
2021	Wikipedia	NMT	10k	7k /1k/(blind)
Total	Mixed	Mixed	59k	57k/1k/1k

Table 1
The datasets used in this research and the final distribution of our data. Translation models are either Statistical Machine Translation models (SMT) or Neural Machine Translation models (NMT).

original implementation, word-by-word attention (rte), and another implementation of dot attention taken from the torchnlp library (Petrochuk, 2018).

The vocabulary of the model was processed using byte pair encoding (BPE) model with a vocabulary size of 88500 (Sennrich et al., 2016; Shibata et al., 1999). We used the BPE implementation in the sentencepiece Python library by Google (Kudo & Richardson, 2018).

The network was configured to train to up to a maximum of 100 epochs and would stop earlier if the best score on the development dataset would not change for more than 5 epochs.

3.3 Estimating Computational Complexity

Initially, we considered to using Watt consumed by the GPU as our proxy measure for computational complexity. While we were aware of the problems with external consistency and inter-rater consistency (different GPUs and different computer environments), during testing we noticed that the measurement was not even internally consistent, this falls in line with the findings of (Lu et al., 2019). Furthermore, measuring the power-draw of a GPU requires a complicated experiment setup and adds significant computational overhead.

Instead of Watts, we decided to estimate the amount of required Floating Point Operations for inference. For this, we make use of the flop-counting utility in the fvcore[2] collection. The utility relies on the inputs and outputs of each operator in the form of `list(torch._C.Value)` variables. For our custom flop counts to integrate with the other estimates by fvcore we constructed them in the same manner, according to the fvcore documentation. We choose fvcore over other third-party tools for measuring flops because its operator-level way of counting operations enables it to handle customs modules gracefully without implementing custom functions, and over the PyTorch profiling implementation for its easier extensibility. Since fvcore is mostly used for computer vision most, of the attention operators, have not been implemented. We needed to create a custom handle that calculates the operations based on the input and output that each LSTM cell receives.

2 https://github.com/facebookresearch/fvcore

Algorithm	Hyperparameters	Type
Random Sampling	NA	NA
RandomSampler	NA	NA
MOTPESampler	n_startup_trials = 65	MOO
NSGAII	population_size = 25	MOO
TPESampler	multivariate = True	SOO

Table 2
The investigated search algorithms, their used hyperparameters and the category of search algorithm.

3.4 Overall Experiment Design

Our work relies on the optimization algorithms implemented in the Optuna Python library (Akiba et al., 2019). In table 2 we present an overview of the hyperparameters for each algorithm. For a single experiment, an algorithm is configured to run 100 trials/objective function evaluations, i.e. it schedules the training of 100 QE models. Based on the Pearson Correlation Coefficient of the HTER scores, and in the case of MOO, Floating Point operation approximation reported, the samplers suggest new hyperparameter configurations to be trained and evaluated. We repeat these runs with three different seeds for the random number generators of PyTorch, Numpy, and Python. When the experiments are completed we calculate the mean objective values from the solution of the random search (RandomSampler, the name given to the random search algorithm in Optuna is confusing when the conceptually different random sampling technique is also used) and use these to approximate the expected value of randomly sampling the search space. We use random sampling as our lowest level baseline and random search as our second level baseline. The methodology utilizing random search and random sampling as baselines, and executing the search using different easily reproducible seeds (integers from 0 to 2) for the random number generators is shaped by the considerations in (Lindauer & Hutter, 2020).

The search space description in table 3 outlines the set of all possible hyperparameter combinations. We tasked the aforementioned algorithms to find solutions from this space that best satisfy our optimization criteria. Once an algorithm has selected a configuration to evaluate it sends these to the objective function. During each run of the objective function we train a QE model until we reach 100 epochs or until the early stopping condition triggers. After every epoch of training, we evaluate the algorithm's performance on the development set and the test set. When the training is completed we evaluate the model's computational complexity using our customized flop_count function. We report the FLOPS and the best performance on the development set back to the sampler. Furthermore, we save the test set performance from the epoch in which we achieved the highest development set performance for evaluation of the models.

We run 3 sets of experiments one for each seed in which we evaluate each respective search algorithm. The experiments were executed on a machine running Manjaro Linux version 21.0.6, PyTorch version 1.8.1, Optuna version 2.8.0, Python version 3.9.5, using an RTX 2060 with CUDA version 11.3. Each experiment with 100 trials took roughly 8 hours to evaluate.

integers	start/stop/step
embedding dim.	48/248/8
hidden dim.	48/248/8
batch size	8/248/8
floating-point numbers	start/stop
learning_rate	1.5e-3/1.5e-2
dropout_rate	0.1/0.3
categorical variables	options
Attention mechanism	'dot', 'rte', 'nlpAttdot'

Table 3
The hyperparameter search space.

3.5 Evaluation

We evaluate the performance of our models based on the performance of our test set. Even though we calculate Pearson's correlation every epoch, we do not report this score to the optimizer. Instead, when the model terminates the training, we report the performance on the test set from the epoch with the highest score on our development set. The theoretical best possible score a model could achieve, also called the utopia point, is a Pearson correlation of one and zero Flops for inference. We invert the Pearson correlation coefficient values to convert the directions of our optimization from maximizing the Pearson correlation coefficient and minimizing the Flops, to minimizing both. Now our utopia point lies at the coordinates $(0, 0)$ in our objective space and calculating performance metrics for multi-objective optimization becomes substantially easier.

To assess the difference in quality for the multi-objective solutions we compare them based on the hypervolume indicator (Zitzler & Thiele, 1999; Zitzler et al., 2007). We make use of the pygmo2 Python library to calculate these metrics(Biscani & Izzo, 2020).

4. Results

In table 4 we see that the mean hypervolume of all the samplers is quite similar and that all they have a very high standard deviation. When we conduct the Kruskal-Wallis H-test based on the results from all the different algorithms, we find that the reported hypervolumes don't differ from each other in a statistically relevant way (p-value = 0.91). The close similarity in terms of the hypervolume indicator for all search procedures is also visible in figure 1 where we compare the mean hypervolume over each iteration. One iteration is equivalent to one objective function evaluation, i.e. the training of one QE model.

4.1 MOO vs. MOO

The mean hypervolume of both MOO algorithms investigated is reported in table 4. Because of the result of the previously mentioned Kruskal-Wallis H-test we have to

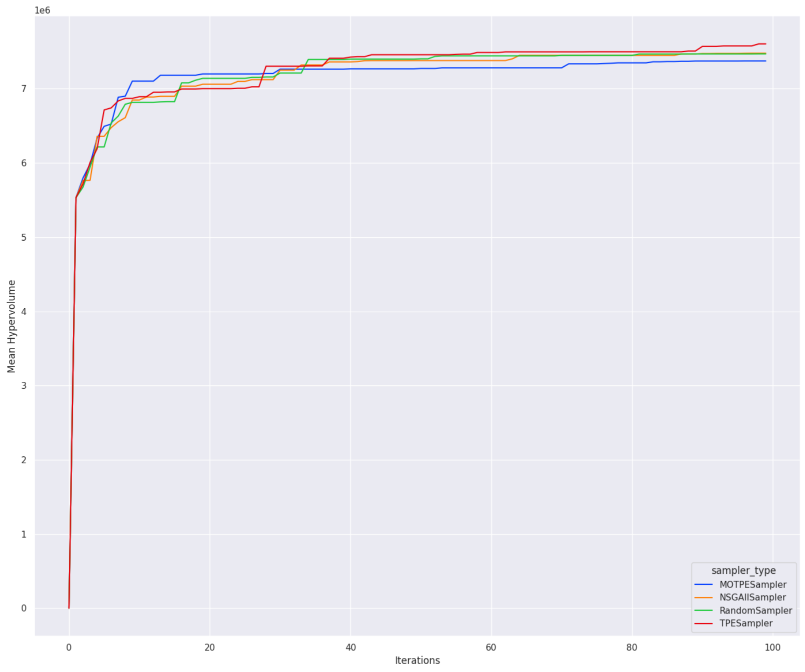

Figure 1
The mean hypervolume at every step of the tested optimizers.

Algorithm	mean ± std.
Random Search	$7.47 \times 10^6 \pm 5.93 \times 10^5$
MOTPE	$7.37 \times 10^6 \pm 6.90 \times 10^5$
NSGAII	$7.48 \times 10^6 \pm 5.39 \times 10^5$
TPE	$7.60 \times 10^6 \pm 4.79 \times 10^5$

Table 4
The mean hypervolume with standard deviation of each respective algorithm.

conclude that the differences in mean hypervolume between the MOO algorithms are not statistically significant.

4.2 MOO vs. Baseline

As visible in table 5 Both NSGA-II and MOTPE have a mean solution that strictly dominates the expected value of the random sampling method. While this shows that the average of their suggested solutions was better as stated before the results in respect to the mean hypervolume are not significant and therefore we cannot judge whether

Algorithm	PCC	Flops
Random Sampling	0.226	8.254×10^5
MOTPE	0.260	6.322×10^5
NSGAII	0.285	6.183×10^5
TPE	0.290	9.348×10^5

Table 5
Our approximation of the expected values from the random sampling method and the average scores produced by the other search methods.

any MOO algorithm generates a better or worse set of Pareto optimal solutions than our second baseline the random search algorithm.

4.3 MOO vs. SOO

To compare the MOO with the SOO algorithms we chose to separately compare the performance on each objective. First, we analyze the results in terms of the PCC. According to the Kruskal-Wallis H-test we can reject the null hypothesis that the populations of the search algorithms are the same ($p = 0.001$). When we conduct Wilcoxon signed-rank test between the TPE algorithm and the other algorithms. We find that the TPE algorithm performs significantly better than the MOTPE algorithm ($p = 0.007$, bonferroni-corrected $\alpha = 0.01$), but not significantly better than any of the others. When we perform the same analysis in terms of floating-point operations we reject the null hypothesis of the Kruskal-Wallis H-test with ($p = 3.739 \times 10^-5$) and find that the TPE algorithm performs significantly worse, compared to the NSGA-II and MOTPE algorithms.

Another interesting comparison between the TPE sampler and our multi-objective methods can be seen in figure 2. The figure shows the mean of the best solutions (in terms of PCC) produced by the TPE algorithm at the tail of the arrows. The head of each respective arrow represents the next best solution by one of the other algorithms that has a lower or equal value of FLOPS. This figure is based on the reporting in (Binder et al., 2020).

5. Discussion

The goal of the presented research was to identify differences in performance in-between the two MOO algorithms, between the MOO and SOO algorithms, and between the MOO algorithms and our baselines. A great deal of our results remains inconclusive. We think the biggest contribution of our study lies in the developed methodology for evaluating optimization algorithms with respect to HPO of deep learning models. Our work combines the insights on how to evaluate a model's complexity in Lu et al. (2019) with the insightful evaluation methodology in Binder et al. (2020). Furthermore, we follow some the suggestions in Lindauer & Hutter (2020) and thus create a useful precedent for evaluating HPO algorithms for DL. In the following sections, we will discuss our result with respect to our research questions. Then we will analyze the limitations of our work and end with suggestions on how to solve the problems we encountered.

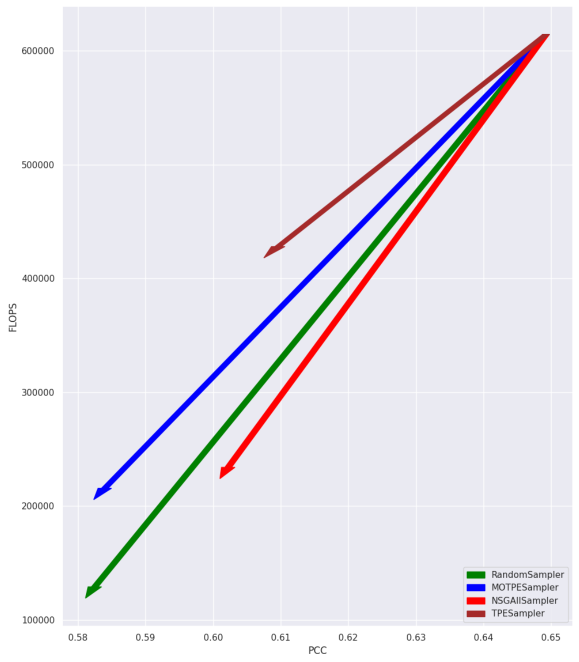

Figure 2
A comparison of the mean best solution by the TPE algorithm (tail of the arrows) and mean of the next best but more efficient solutions by the other algorithms (head of arrows).

5.1 MOO vs. MOO

As shown in section 4, the data we collected is not substantial enough to draw any conclusions about the performance in terms of the reported hypervolume. Therefore, we are unable to give definitive answer to our first sub question.

5.2 MOO vs. Baseline

With respect to the baselines, both of our MOO algorithms scored better than the random sampling baseline. The performance of random sampling is quite poor and this baseline does not provide much information on the suitability of MOO algorithms for tuning QE systems.

5.3 MOO vs. SOO

We can confidently suggest that the TPE algorithm optimizing for the PCC finds significantly better solutions than the MOTPE algorithm. For the FLOPS measurement both MOTPE and NSGA-II find solutions with a significantly lower complexity than the TPE algorithm. As illustrated in `figure cool arrows` the difference in FLOPS between the solutions is quite substantial. These results are quite similar to the ones reported in the work of Binder et al. (2020). It should be stated that our results with respect to the best HTER scores are not comparable to the ones reported in the findings of the 2020 WMT shared task since our test set is presumably much easier than the one of the competition.

5.4 Limitations

In this section, we will take the space to try and analyze what might have caused our results to be this inconclusive. Since all the algorithms performed similarly to the random search algorithm, the ability of the algorithms to model a relationship might be inhibited by some part of our methodology. We identify a few possibly problematic effects of our methodology:

1. Our initial assumption was that for a model to have a high predictive performance it requires a high computational complexity. To test whether this is relationship is reflected in the data we collected we calculate Spearman's rho between the Flops and the PCC reported by the random search algorithm. While we only see a weakly positive correlation between the two variables ($r_s = 0.115$, $p = 0.023$), this test does not consider the other hyperparameters which clearly have a big influence on the predictive performance.
2. Another possible explanation comes in regards to the dataset and the way we adapted it to our experiments. The methodology for training and optimizing our models relies on the generally accepted standard of using a training, a development and a test set. But when training the models it became quite clear that the scores on our development set and the scores on our test set did not align very well. This suggests that in our selection procedure for these sets was poor. As a consequence of this problem, we often identified a training epoch as the best performing epoch when indeed in that epoch our model was performing quite poorly on the test set. Our multi-objective algorithms were tasked to explore a trade-off between development set PCC and Flops but we evaluated them in terms of test set PCC. This offers a plausible explanation for the inability of the algorithms to converge on the theoretically optimal Pareto front.
3. It appears that the performance of models is highly dependent on the initialization of weights. Therefore small changes in architecture might have an unexpectedly big influence on the predictive performance.
4. It might be the case that the model architecture is quite robust in terms of its hyperparameters and that many configurations lead to a similar performance.
5. The experiments we executed were potentially too short in terms of iterations and most likely we conducted a too small amount of experiments. Since this reduced the number of samples available to use in our statistical tests, this most likely had a big impact on our results.

Of the presented aspects we identify item 2 as the most problematic. Item 2 directly impacts the quality of results we report to the search algorithms. This in turn would influence the search algorithms internal models of the solution space and compromise their ability to converge. This problem also has an influence on the data we use to judge the performance of the investigated algorithms, further compromising our results. Item 5 undoubtedly had a great impact on the insignificance of our results. Item 3, especially in combination with item 4 and item 1 could decrease the ability of the algorithms to identify a relationship between the hyperparameter configurations and the results of the model evaluations. Though on their own, item 1 and item 4 don't seem very problematic. While the relationship between PCC and FLOPS might have not been as strong as expected, we still expect the evaluated search algorithms to be able model this relationship.

5.5 Remedies and Future Research

In order to avoid the problems detailed in the previous section, we suggest paying great attention in forming representative development and test sets. We suspect that in our case they were too small. Also, we suggest avoiding our method of splitting the original development sets and using them to populate training and development and test data. We think that this caused our development and test set to lose their ability of evaluating the model's ability to generalize. The remedies for item 5 are easy to identify: conduct a greater amount of experiments with a greater amount of iterations. In regards to item 4 we recommend paying attention on how architectures considered for future research respond to small changes in hyperparameters. In general, we think that applying our methodology to deeper neural network architectures could yield better results. Furthermore, we suggest future studies to evaluate the optimization algorithms on multiple different model architectures and datasets.

6. Conclusion

While MOO has shown promising results in other studies (Binder et al., 2020; Lu et al., 2019; Horn & Bischl, 2016), we were unable to produce similarly positive results. We think this was mostly due to the methodological shortcomings of our approach. The goal of our study was to compare multi-objective and single-objective optimization in the use case of quality estimation for machine translation. While we did not succeed at finding answers for most of our research questions, the methodology we propose - when improved in the shortcomings we identified - should pave the way for greater and deeper investigations of MOO not only for QE systems but also for other types of deep learning models.

The self reflection can be found in appendix section 6.

References

Akiba, T., Sano, S., Yanase, T., Ohta, T., & Koyama, M. (2019). Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

Bergstra, J., Bardenet, R., Bengio, Y., & Kégl, B. (2011). Algorithms for hyper-parameter optimization. In *25th annual conference on neural information processing systems (NIPS 2011)*, volume 24. Neural Information Processing Systems Foundation.

Bergstra, J., Yamins, D., & Cox, D. (2013). Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In Dasgupta, S. & McAllester, D. (Eds.), *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, (pp. 115–123)., Atlanta, Georgia, USA. PMLR.

Binder, M., Moosbauer, J., Thomas, J., & Bischl, B. (2020). Multi-objective hyperparameter tuning and feature selection using filter ensembles. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*, (pp. 471–479).

Biscani, F. & Izzo, D. (2020). A parallel global multiobjective framework for optimization: pagmo. *Journal of Open Source Software*, *5*(53), 2338.

Chiandussi, G., Codegone, M., Ferrero, S., & Varesio, F. E. (2012). Comparison of multi-objective optimization methodologies for engineering applications. *Computers & Mathematics with Applications*, *63*(5), 912–942.

Deb, K., Goodman, E. D., Coello, C. A. C., Klamroth, K., Miettinen, K., Mostaghim, S., & Reed, P. M. (Eds.). (2019). *Evolutionary Multi-Criterion Optimization - 10th International Conference, EMO 2019, East Lansing, MI, USA, March 10-13, 2019, Proceedings*, volume 11411 of *Lecture Notes in Computer Science*. Springer.

Deb, K., Pratap, A., Agarwal, S., & Meyarivan, T. (2002). A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE transactions on evolutionary computation*, *6*(2), 182–197.

Eggensperger, K., Hutter, F., Hoos, H., & Leyton-Brown, K. (2015). Efficient benchmarking of hyperparameter optimizers via surrogates. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 29.

Ehrgott, M. (2005). *Multicriteria optimization*, volume 491. Springer Science & Business Media.

Elsken, T., Metzen, J. H., & Hutter, F. (2018). Efficient multi-objective neural architecture search via lamarckian evolution. *arXiv preprint arXiv:1804.09081*.

Elsken, T., Metzen, J. H., & Hutter, F. (2019). Neural architecture search: A survey.

Hamdy, M., Nguyen, A.-T., & Hensen, J. L. (2016). A performance comparison of multi-objective optimization algorithms for solving nearly-zero-energy-building design problems. *Energy and Buildings*, *121*, 57–71.

Hebbal, A., Brevault, L., Balesdent, M., Talbi, E.-G., & Melab, N. (2019). Multi-objective optimization using deep gaussian processes: Application to aerospace vehicle design. In *AIAA Scitech 2019 Forum*, (pp. 1973).

Horn, D. & Bischl, B. (2016). Multi-objective parameter configuration of machine learning algorithms using model-based optimization. In *2016 IEEE symposium series on computational intelligence (SSCI)*, (pp. 1–8). IEEE.

Hsu, C.-H., Chang, S.-H., Liang, J.-H., Chou, H.-P., Liu, C.-H., Chang, S.-C., Pan, J.-Y., Chen, Y.-T., Wei, W., & Juan, D.-C. (2018). Monas: Multi-objective neural architecture search using reinforcement learning.

Huband, S., Barone, L., While, L., & Hingston, P. (2005). A scalable multi-objective test problem toolkit. In *International Conference on Evolutionary Multi-Criterion Optimization*, (pp. 280–295). Springer.

Huband, S., Hingston, P., Barone, L., & While, L. (2006). A review of multiobjective test problems and a scalable test problem toolkit. *IEEE Transactions on Evolutionary Computation*, *10*(5), 477–506.

Kim, H., Jung, H.-Y., Kwon, H., Lee, J.-H., & Na, S.-H. (2017). Predictor-estimator: Neural quality estimation based on target word prediction for machine translation. *ACM Transactions on Asian and Low-Resource Language Information Processing (TALLIP)*, *17*(1), 1–22.

Kudo, T. & Richardson, J. (2018). SentencePiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. (pp. 66–71). Association for Computational Linguistics.

Lindauer, M. & Hutter, F. (2020). Best practices for scientific research on neural architecture search. *Journal of Machine Learning Research*, *21*(243), 1–18.

Lu, Z., Whalen, I., Boddeti, V., Dhebar, Y., Deb, K., Goodman, E., & Banzhaf, W. (2019). Nsga-net: neural architecture search using multi-objective genetic algorithm. In *Proceedings of the Genetic and Evolutionary Computation Conference*, (pp. 419–427).

Ozaki, Y., Tanigaki, Y., Watanabe, S., & Onishi, M. (2020). Multiobjective tree-structured parzen estimator for computationally expensive optimization problems. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*, (pp. 533–541).

Parsa, M., Mitchell, J. P., Schuman, C. D., Patton, R. M., Potok, T. E., & Roy, K. (2020). Bayesian multi-objective hyperparameter optimization for accurate, fast, and efficient neural network accelerator design. *Frontiers in Neuroscience*, *14*, 667.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., & Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, & R. Garnett (Eds.), *Advances in Neural Information Processing Systems 32* (pp. 8024–8035). Curran Associates, Inc.

Petrochuk, M. (2018). Pytorch-nlp: Rapid prototyping with pytorch natural language processing (nlp) tools. https://github.com/PetrochukM/PyTorch-NLP.

Riquelme, N., Von Lücken, C., & Baran, B. (2015). Performance metrics in multi-objective optimization. In *2015 Latin American Computing Conference (CLEI)*, (pp. 1–11). IEEE.

Sennrich, R., Haddow, B., & Birch, A. (2016). Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, (pp. 1715–1725)., Berlin, Germany. Association for Computational Linguistics.

Shibata, Y., Kida, T., Fukamachi, S., Takeda, M., Shinohara, A., Shinohara, T., & Arikawa, S. (1999). Byte pair encoding: A text compression scheme that accelerates pattern matching. Technical report, Citeseer.

Shterionov, D., Do Carmo, F., Moorkens, J., Paquin, E., Schmidtke, D., Groves, D., & Way, A. (2019). When less is more in neural quality estimation of machine translation. an industry case study. In *Proceedings of Machine Translation Summit XVII Volume 2: Translator, Project and User Tracks*, (pp. 228–235).

Snover, M., Dorr, B., Schwartz, R., Micciulla, L., & Makhoul, J. (2006). A study of translation edit rate with targeted human annotation. In *Proceedings of the 7th Conference of the Association for Machine Translation in the Americas: Technical Papers*, (pp. 223–231)., Cambridge, Massachusetts, USA. Association for Machine Translation in the Americas.

Specia, L., Blain, F., Fomicheva, M., Fonseca, E., Chaudhary, V., Guzmán, F., & Martins, A. F. T. (2020). Findings of the WMT 2020 shared task on quality estimation. In *Proceedings of the Fifth Conference on Machine Translation*, (pp. 743–764)., Online. Association for Computational Linguistics.

Specia, L., Scarton, C., & Paetzold, G. H. (2018). Quality estimation for machine translation. *Synthesis Lectures on Human Language Technologies*, *11*(1), 1–162.

Srinivas, N. & Deb, K. (1994). Muiltiobjective optimization using nondominated sorting in genetic algorithms. *Evolutionary computation*, *2*(3), 221–248.

Trisna, T., Marimin, M., Arkeman, Y., & Sunarti, T. (2016). Multi-objective optimization for supply chain management problem: A literature review. *Decision Science Letters*, *5*(2), 283–316.

Wang, M., Yang, H., Shang, H., Wei, D., Guo, J., Lei, L., Qin, Y., Tao, S., Sun, S., Chen, Y., & Li, L. (2020). Hw-tsc's participation at wmt 2020 quality estimation shared task. In *WMT@EMNLP*.

Zitzler, E., Brockhoff, D., & Thiele, L. (2007). The hypervolume indicator revisited: On the design of pareto-compliant indicators via weighted integration. In *International Conference on Evolutionary Multi-Criterion Optimization*, (pp. 862–876). Springer.

Zitzler, E. & Thiele, L. (1999). Multiobjective evolutionary algorithms: a comparative case study and the strength pareto approach. *IEEE transactions on Evolutionary Computation*, *3*(4), 257–271.

Appendix A: Self Reflection

The writing of this thesis has been a big learning process for me. Since there was no good precedent of research that I could follow and base my methodology on, every time I learned something new it usually also meant that I had to go back and rethink the steps I took before. This iterative process of improvement was hard to cope with mentally. A good example of this is the initial attempts at approximating model complexity by measuring the power draw of the GPU during inference. In the end, this methodology was discarded in favor of the present approach utilizing FLOPS counting. While this switch of methodology might have helped to make this work more relevant it cost me a great amount of time and compromised my ability to focus on the writing. Overall I would summarize that this thesis taught me that at some point I have to stop with these iterative improvements and focus on finishing the work.

Appendix B: Plot of all the found model configurations

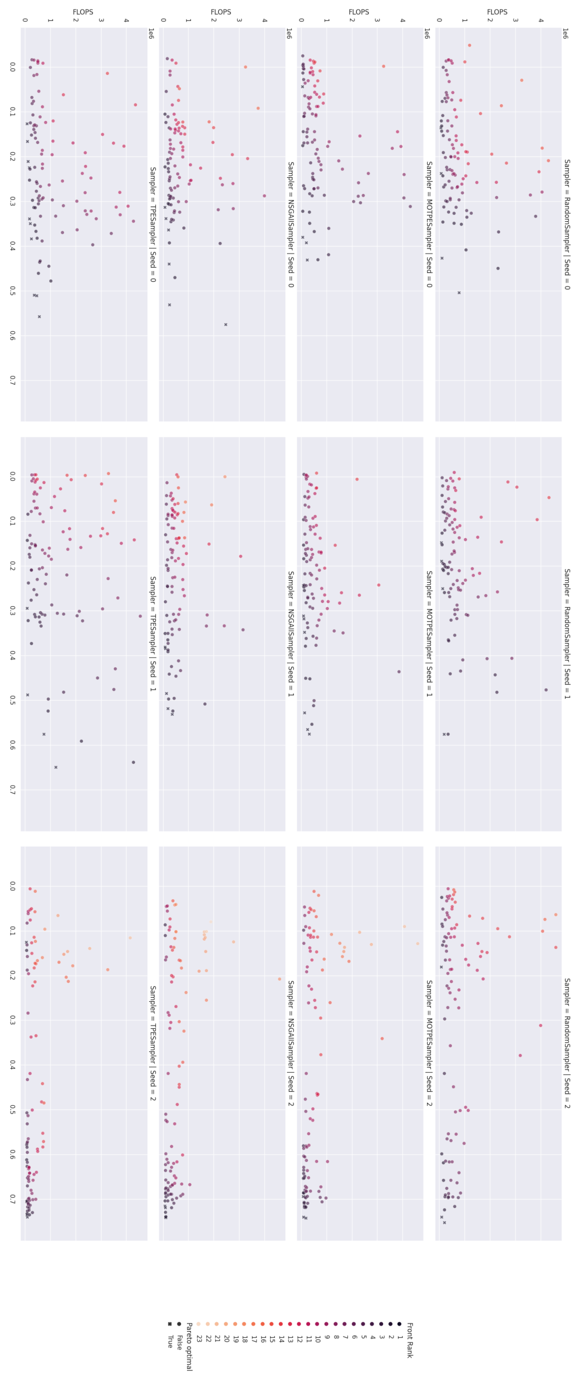

Figure 1
All the found model configurations sorted by seeds (columns) and samplers (rows).