Reproducible and Efficient Benchmarks for Hyperparameter Optimization of Neural Machine Translation Systems

Xuan Zhang and **Kevin Duh**

Johns Hopkins University
xuanzhang@jhu.edu, kevinduh@cs.jhu.edu

Abstract

Hyperparameter selection is a crucial part of building neural machine translation (NMT) systems across both academia and industry. Fine-grained adjustments to a model's architecture or training recipe can mean the difference between a positive and negative research result or between a state-of-the-art and underperforming system. While recent literature has proposed methods for automatic hyperparameter optimization (HPO), there has been limited work on applying these methods to neural machine translation (NMT), due in part to the high costs associated with experiments that train large numbers of model variants. To facilitate research in this space, we introduce a lookup-based approach that uses a library of pre-trained models for fast, low cost HPO experimentation. Our contributions include (1) the release of a large collection of trained NMT models covering a wide range of hyperparameters, (2) the proposal of targeted metrics for evaluating HPO methods on NMT, and (3) a reproducible benchmark of several HPO methods against our model library, including novel graph-based and multiobjective methods.

1 Introduction

Choosing effective hyperparameters is crucial for building strong neural machine translation (NMT) systems. Although some choices present obvious trade-offs (e.g., more and larger layers tend to increase quality at the cost of speed), others are more subtle (e.g., effects of batch size, learning rate, and normalization techniques on different layer types). Optimal versus suboptimal hyperparameters can lead to dramatic swings in system performance; consider the wide range of BLEU scores for variants of the same base system in Figure 1 (left). In practice, these hyperparameters are often tuned manually based on intuition and heuristics, a tedious and error-prone process that can lead to unreliable experimental results and underperforming shared task or production systems. The difficulty is compounded when system builders must jointly optimize *multiple objectives*, such as translation accuracy (BLEU) and decoding speed, which are largely uncorrelated as shown in Figure 1 (right).

In the past decade, various hyperparameter optimization (HPO) methods have emerged in the machine learning literature under the labels of ''AutoML'' (Bergstra et al., 2011; Hutter et al., 2011; Bardenet et al., 2013; Snoek et al., 2015) and ''neural architecture search'' (Zoph and Le, 2016; Liu et al., 2018a,b; Cai et al., 2018; Real et al., 2019). However, it is unclear how they perform on NMT; we are not aware of any prior work with comprehensive evaluation. One challenge is that the state-of-the-art NMT models (Sutskever et al., 2014; Bahdanau et al., 2015; Gehring et al., 2017; Vaswani et al., 2017) require significant computational resources for training. Secondly, they usually have large hyperparameter search spaces. Thus, it is prohibitively expensive in practice to compare HPO methods on NMT tasks.

In order to *enable reproducible HPO research on NMT tasks*, we adopt a benchmark procedure based on ''table-lookup''. This approach was recently introduced to neural architecture search by Ying et al. (2019), and to hyperparameter optimization by Klein and Hutter (2019). First, we train an extremely large number of NMT models with diverse hyperparameter settings and record their performance metrics (e.g., BLEU, decoding time) in a table. Then, we constrain our HPO methods to sample from this finite set of models. This allows us to simply ''look-up'' their precomputed performance metrics, and amortizes the burden of computation: As long as we ensure that

Transactions of the Association for Computational Linguistics, vol. 8, pp. 393–408, 2020. https://doi.org/10.1162/tacl_a_00322
Action Editor: Alexandra Birch. Submission batch: 11/2019; Revision batch: 2/2020; Published 7/2020.

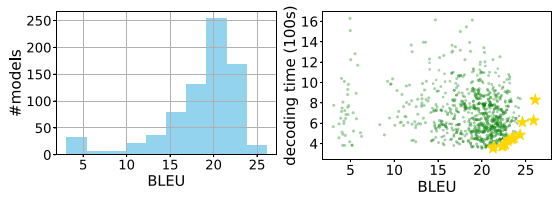

Figure 1: Left: Histogram of BLEU scores that show wide variance in performance for a base NMT system (transformer) with different hyperparameters (e.g., BPE operations, # of layers, initial learning rate). Right: Scatterplot of BLEU and decoding time with different hyperparameters. Gold stars represent the Pareto-optimal systems.

we have trained and pre-computed a large number of representative NMT models beforehand, HPO algorithm developers no longer need to deal with the cost of training NMT. Importantly, this kind of benchmark significantly speeds up the HPO experiment turnover time, enabling fast repeated trials for rigorous tests and facilitates detailed error analysis.

The main contributions of this work are:

1. **Dataset**: We release a benchmark dataset[1] for comparing HPO methods on NMT models. This ‘‘table-lookup’’ HPO dataset supports both single-objective and multiobjective optimization of translation accuracy and decoding time (Section 3). Specifically, we trained a total of 2,245 Transformers (Vaswani et al., 2017) on six different corpora (with a cost of approximately 1,547 GPU days), and collected all pairs of hyperparameter settings and corresponding performance metrics.

2. **Evaluation protocols**: We provide three kinds of metrics for evaluating HPO methods, based on different computational budgets (Section 4). We also demonstrate error analysis techniques that are enabled by this ‘‘table-lookup’’ framework, which provide insights into algorithm behavior (Section 7).

3. **HPO method benchmarks**: We benchmark the performance of several HPO methods on our dataset (Section 6). These include Bayesian optimization as well as a novel graph-based method that exploits the structure of the hyperparameter space (Section 5). We also extend these methods to handle the multiobjective optimization of both BLEU and decoding time. These experiments illustrate how to utilize the dataset to rigorously evaluate HPO for NMT.

[1] https://github.com/Este1le/hpo_nmt.

2 HPO Problem Definition

Given a machine learning algorithm with H hyperparameters, we denote the domain of the h-th hyperparameter by Λ_h and the overall hyperparameter configuration space as $\mathbf{\Lambda} = \Lambda_1 \times \Lambda_2 \times \ldots \Lambda_H$. When trained with a hyperparameter setting $\boldsymbol{\lambda} \in \mathbf{\Lambda}$ on data $\mathcal{D}_{train}$, the algorithm's performance metric on some validation data $\mathcal{D}_{valid}$ is $f(\boldsymbol{\lambda}) := \boldsymbol{\mathcal{V}}(\boldsymbol{\lambda}, \mathcal{D}_{train}, \mathcal{D}_{valid})$. In the context of NMT, $f(\cdot)$ or $\boldsymbol{\mathcal{V}}(\cdot)$ could be the perplexity, translation accuracy (e.g., BLEU score), or decoding time on $\mathcal{D}_{valid}$. In general, $f(\cdot)$ is computationally expensive to obtain; it requires training a model to completion, then evaluating some performance metric on a validation set. For purposes of exposition, we assume that lower $f(\cdot)$ is better, so we might define $f(\cdot)$ as $1 - \text{BLEU}$.

The goal of hyperparameter optimization is then to find a $\boldsymbol{\lambda}_\star = \arg\min_{\boldsymbol{\lambda} \in \mathbf{\Lambda}} f(\boldsymbol{\lambda})$, with as few evaluations of $f(\cdot)$ as possible. An HPO problem can be challenging for three reasons: (a) $\mathbf{\Lambda}$ may be a combinatorially large space, prohibiting grid search over hyperparameters. (b) $f(\cdot)$ may be expensive to compute, so there is a tight budget on how many evaluations of $f(\cdot)$ are allowed. (c) f is not a continuous function and no gradient information can be exploited, forcing us to view the $\arg\min$ as a blackbox discrete search problem. NMT HPO search exhibits all these conditions.

One class of algorithms that tackles the HPO problem is sequential model-based optimization (SMBO), illustrated in Figure 2. SMBO approximates f with a cheap-to-evaluate surrogate model $\hat{f}$ (Feurer and Hutter, 2019; Luo, 2016; Jones et al., 1998). SMBO starts by querying f with initial hyperparameters $\{\boldsymbol{\lambda}_{init}\}$ and recording the resulting $(\boldsymbol{\lambda}_{init}, f(\boldsymbol{\lambda}_{init}))$ pairs. Then, it iteratively (1) fits the surrogate $\hat{f}$ on pairs observed so far; (2) gets the predictions $\hat{f}(\boldsymbol{\lambda}_i)$ for unlabeled/unobserved hyperparameters; and (3) selects a promising $\boldsymbol{\lambda}_p$ to query next based on these predictions and an acquisition function, whose role is to trade off exploration in $\mathbf{\Lambda}$ with high model uncertainty and exploitation in $\mathbf{\Lambda}$ with low $\hat{f}(\cdot)$.

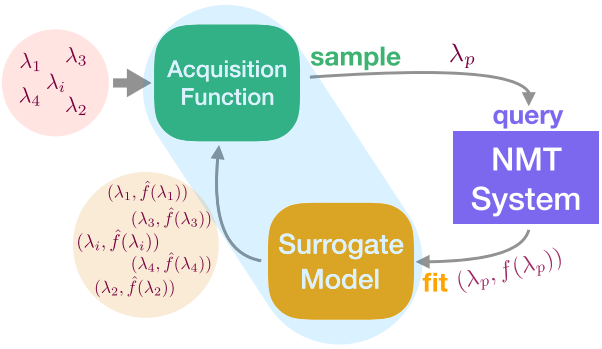

Figure 2: SMBO framework. The part shaded in light blue contains two ingredients required for implementing a SMBO method: the surrogate model and the acquisition function, for which we will present two choices in Section 5.

Evolutionary algorithms (Eberhart and Shi, 1998; Simon, 2013) are also used to solve HPO problems. Unlike SMBO, they do not approximate f with a surrogate $\hat{f}$; rather, they directly sample hyperparameters with high $f(\cdot)$ from a population and recombine them to form the next query.[2]

3 Table-Lookup HPO Datasets

3.1 Table-Lookup Framework

To evaluate a newly devised HPO algorithm, one needs to run each component of the loop in Figure 2. However, the ''query'' step is computationally expensive: We need to train a new NMT system each time we sample a new hyperparameter.

The idea of table lookup is to simply pre-train a large set of I NMT systems and record the pairs $\{\boldsymbol{\lambda}_i, f(\boldsymbol{\lambda}_i)\}_{i=1,\ldots,I}$ in a table. Thus, when running the loop in Figure 2, the HPO algorithm developer can look up $f(\boldsymbol{\lambda}_i)$ whenever necessary, without having to train a NMT model from scratch. This significantly speeds up the experimental process. The advantages are:

1. One can perform multiple random trials of the same algorithm, to test robustness.

2. One can perform comparisons with more baseline algorithms, to make stronger claims.

3. One can perform the same experiment under different budget constraints, to simulate different real-world use cases.

[2]We focus on SMBO methods in this paper, but note that our dataset is amenable to any HPO method.

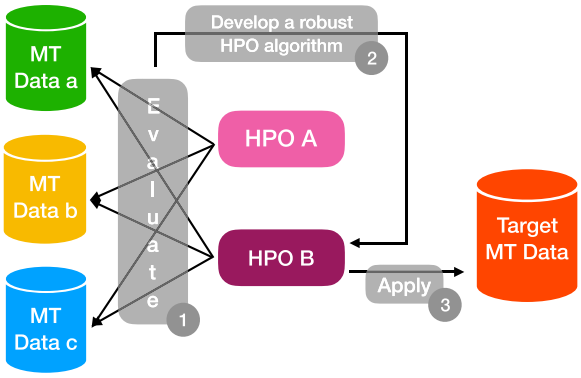

Figure 3: The workflow of HPO algorithm selection/development. HPO algorithm candidates are first evaluated on lookup tables built from multiple MT datasets. Promising candidates may be further developed and evaluated. The most robust one will be selected to apply to the target MT data.

4. One can track the progress of an experiment with respect to oracle results, allowing for more detailed error analysis of HPO.

To be effective, table lookup depends on two important assumptions: First, the table has to be sufficiently large to cover the space of hyperparameters $\boldsymbol{\Lambda}$. Second, the HPO algorithm needs to be modified to sample from the finite set of hyperparameters in the table; this is usually easy to implement but the assumption is that finite-sample results will generalize.

3.2 HPO Algorithm Selection/Development

There exist many choices of HPO algorithm, which can be evaluated or further developed on our lookup tables. Figure 3 illustrates this process. The performance of HPO algorithm candidates on various MT datasets serves as the basis for HPO selection. The selected HPO algorithm can then be applied on new MT datasets.

There are two kinds of generalization effects at play: (1) generalization of an HPO algorithm across MT datasets, and (2) generalization of MT models and their associated hyperparameters across MT datasets. We mainly care about (1) in the algorithm development process, which is why we opt to provide six distinct datasets described in Section 3.3 (as opposed to, e.g., 1 dataset trained on large MT data). If an HPO algorithm performs efficiently in finding good hyperparameter configurations on many MT datasets, then we can more reasonably believe that it will run quickly on a new dataset, regardless of

the underlying MT data characteristics. Even if the best configuration on one MT dataset does not transfer to another, a robust HPO algorithm should still be capable of finding good hyperparameters because the algorithm learns from scratch on each dataset independently.

3.3 MT Data and Setup

To create a robust HPO benchmark, we trained NMT models on six different parallel corpora, which exhibit a variety of characteristics:

TED Talks: We trained Chinese–English (**zh-en**) and Russian–English (**ru-en**) models on the data-split of Duh (2018). This is a mid-resource setup, where $\mathcal{D}_{train}$ consists of 170k lines for zh-en and 180k lines for ru-zh. $\mathcal{D}_{valid}$ has 1,958 sentences and is multiway parallel for both language-pairs.

WMT2019 Robustness task (Li et al., 2019): We trained models on Japanese–English data, in both directions (**ja-en**, **en-ja**). $\mathcal{D}_{train}$ has 4 M lines from a mix of domains. $\mathcal{D}_{valid}$ is a concatenation of 4k mixed-domain sentences and 1k Reddit sentences, for a total of 5,405 lines. The goal of the Robustness task is to test how NMT systems perform on non-standard and noisy text (e.g., Reddit).

Low Resource tasks: We trained models using the IARPA MATERIAL datasets for Swahili–English (**sw-en**) and Somali–English (**so-en**). $\mathcal{D}_{train}$ consists of only 24k lines for both language pairs (BUILD set), and $\mathcal{D}_{valid}$ consists of 2675 lines (ANALYSIS2 set).

Although there are many potential MT datasets we could choose from, we believe these six datasets form a good representative set. It ranges from high-to-low resource; it contains both noisy and clean settings. These datasets also have different levels of similarity—for example, zh-en and ru-en TED talks use the same multiway parallel $\mathcal{D}_{valid}$, so one could ask whether the optimal hyperparameters transfer.

The text is tokenized by Jieba for Chinese, by Kytea for Japanese, and by the Moses tokenizer for the rest. Byte pair encoding (BPE) segmentation (Sennrich et al., 2016) is learned and applied separately for each side of bitext. We train Transformer NMT models with Sockeye[3] (Hieber et al., 2017), focusing on these hyperparameters:

- **preprocessing configurations**: number of BPE symbols[4] (bpe)
- **training settings**: initial learning rate (init_lr) for the Adam optimizer
- **architecture designs**:[5] number of layers (#layers), embedding size (#embed), number of hidden units in each layer (#hidden), number of heads in self-attention (#att_heads).

These hyperparameters are chosen because they significantly affect both accuracy and speed of the resulting NMT. Other hyperparameters are kept at their Sockeye defaults.[6] Table 1 shows our overall hyperparameter space $\mathbf{\Lambda}$; in total among all six datasets, we have 1,983 models; Table 2 shows the exact number of models per dataset, along with the best models and their hyperparameter settings.[7]

Rationale for Hyperparameter Values: There are various design trade-offs in deciding the range and granularity of hyperparameter values. First, we might expand on a wider range of values (e.g., change #hidden = {1024, 2048} to {512, 1024, 2048, 4096}). The effect of this is that we test the HPO algorithm on a wider range of inputs, with potentially more variability in metrics like BLEU and inference speed. Second, we might expand on a more finegrained range of values (e.g., change #hidden = {1024, 2048} to {1024, 1536, 2048}). This might result in smoother metrics, making it easier for HPO algorithms to learn. Although wider range and finer granularity are desirable properties for a HPO dataset, each additional value causes an exponential increase in the number of models because of the cross-product of all values. In general, we think Table 1 represents a reasonable set of values used in the literature. Nevertheless, it should be clarified that empirical findings from table-lookup datasets should be interpreted in light of the limits of hyperparameter range and granularity.

[3] https://github.com/awslabs/sockeye.

[4] Same number of BPE operations is used for both sides.

[5] Same values are used for encoder and decoder.

[6] In this paper, we only focused on integer and real-valued hyperparameters. Categorical hyperparameters need special treatment for most HPO algorithms, thus are not considered.

[7] Note that not all possible hyperparameter configurations are included in the dataset: We excluded ones where training failed or clearly did not learn (e.g., achieved ≈ 0 BLEU).

dataset	bpe (1k)	#layers	#embed	#hidden	#att_heads	init_lr (10^{-4})
zh, ru, ja, en	10, 30, 50	2, 4	256, 512, 1024	1024, 2048	8, 16	3, 6, 10
sw	1, 2, 4, 8, 16, 32	1, 2, 4, 6	256, 512, 1024	1024, 2048	8, 16	3, 6, 10
so	1, 2, 4, 8, 16, 32	1, 2, 4	256, 512, 1024	1024, 2048	8, 16	3, 6, 10

Table 1: Hyperparameter search space for the NMT systems.

Dataset	#models	Best BLEU	bpe	#layers	#embed	#hidden	#att_heads	init_lr
zh-en	118	14.66	30k	4	512	1024	16	3e-4
ru-en	176	20.23	10k	4	256	2048	8	3e-4
ja-en	150	16.41	30k	4	512	2048	8	3e-4
en-ja	168	20.74	10k	4	1024	2048	8	3e-4
sw-en	767	26.09	1k	2	256	1024	8	6e-4
so-en	604	11.23	8k	2	512	1024	8	3e-4

Table 2: For each language pair, we report the number of NMT systems trained on it, the oracle best BLEU we obtained, and its corresponding hyperparameter configuration.

3.4 Objectives: Accuracy and Cost

We train all models on $\mathcal{D}_{train}$ until they converge in terms of perplexity on $\mathcal{D}_{valid}$. We then record various performance measurements:

- **Translation accuracy**: BLEU (Papineni et al., 2002) and perplexity on $\mathcal{D}_{valid}$.
- **Computational cost**: GPU wall clock time for decoding $\mathcal{D}_{valid}$, number of updates for the model to converge, GPU memory used for training, total number of model parameters.

In this paper, we use BLEU on $\mathcal{D}_{valid}$ for single-objective experiments; we use BLEU and decoding time for multiobjective experiments.

3.5 Hyperparameter Importance/Correlation

We might be interested in seeing whether good configurations are always good across datasets. This can be done by ranking configurations by BLEU for each dataset, then measuring correlation between rankings. We show the Spearman's correlation coefficient in Figure 4. NMT systems with same language pairs (ja-en vs. en-ja) are highly correlated. On the contrary, other pairs show low correlation (0.084 for ja-en vs. so-en), implying the need to run HPO on new datasets separately.

The table-lookup approach also enables in-depth analyses of how hyperparameters generally affect system performance. Following Klein and Hutter (2019), we assess the importance of

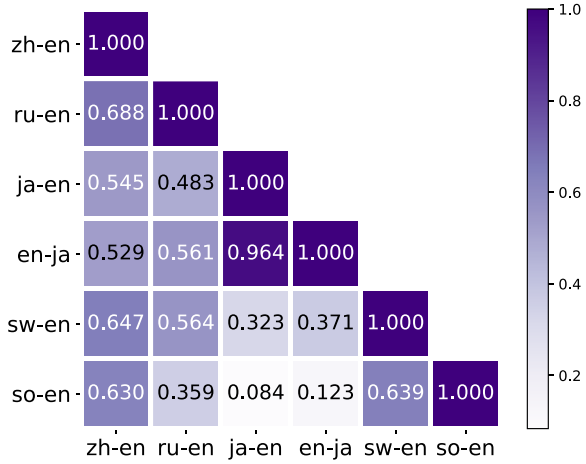

Figure 4: Correlation of hyperparameter rankings across MT datasets.[8]

hyperparameters with fANOVA, which computes the variation in BLEU when changing a specific hyperparameter with values of all the other hyperparameters fixed. In Figure 5, on en-ja, when considering only the top performing NMT models (top left), #att_heads, init_lr, and #embed impact BLEU the most, over the entire configuration space (top middle), #embed is the distinguishing factor. The analysis can be extended to pairs of hyperparameters, where we observe the interaction of init_lr and #embed being important (Figure 5 bottom left).

[8]The ranking is computed only on the subset of MT systems common in all datasets. For this, we consider 30k bpe (for zh, ru, ja, en) to be equivalent to 32k bpe (for sw, so).

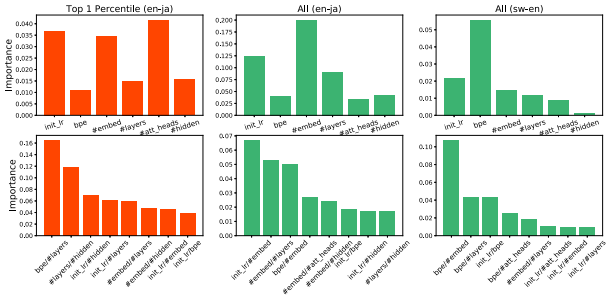

Figure 5: The importance of each hyperparameter (top) and the eight most important hyperparameter pairs (bottom) for top 1% (left), and all NMT models ranked by BLEU on en-ja (middle) and sw-en (right).

Questions may arise over whether the results on en-ja can be taken as general conclusions. We find that it is dataset-dependent—hyperparameter importance ranking differs across language pairs, and is dependent on the range and granularity of hyperparameters considered. As shown in the right column of Figure 5, bpe is the most important hyperparameter for sw-en, instead of #embed. This shows the diversity of our selected MT datasets and the hyperparameter importance analysis is a good tool for probing the search space characteristics of these datasets.

3.6 Reproducible and Efficient Benchmarks

Our table-lookup dataset enables reproducible and efficient benchmarks for HPO of NMT systems. Li and Talwalkar (2019) introduce two notions of reproducibility: exact reproducibility (the reproducibility of reported experimental results); and broad reproducibility (the generalization of the experimental results).[9] Our benchmarks are exact reproducible in the sense that we provide the tables that record all model results (Section 3.3) and the code to run and evaluate our HPO algorithms (Section 6). However, they are not guaranteed to be broad reproducible, because the generalizability of the results might be restricted due to fixed collections of hyperparameter configurations, the variance associated with multiple runs, and the unknown best representative set of MT data. As a result, in this work, we should be careful to not make general conclusions from the observations, but to show how the dataset can be potentially used in facilitating HPO research.

[9]They comment: ‘‘Of the 12 papers published since 2018 at NeurIPS, ICML, and ICLR that introduce novel Neural Architecture Search methods, none are exactly reproducible.’’

4 Evaluation Protocols

To assess HPO method performance, we measure the **runtime** to reach a *quality indicator* (e.g., BLEU) target value. The **runtime** is defined as the number of NMT models trained, or equivalently the number of function evaluations $f(\boldsymbol{\lambda})$ in Figure 2. We consider two ways to measure the HPO performance: **fixed-target** and **fixed-budget**.

4.1 Single-Objective Evaluation Metrics

For single-objective optimization, we have:

- **fixed-target best (ftb)**: We fix the quality indicator value to the best value in the dataset and measure runtime to reach this target.
- **fixed-target close (ftc)**: We measure the runtime to reach a target that is slightly less than the oracle best. This is useful when one can tolerate some performance loss.
- **fixed-budget (fb)**: We fix the budget of function evaluations and measure the difference between the oracle best quality indicator value (e.g., oracle best BLEU) in the dataset vs. the best value achieved by systems queried by the HPO method.

The fixed-budget metric asks what is the best possible system assuming a hard constraint on training resources. The fixed-target metrics ask how much training information is needed to find the best (or approximate best) system in the dataset.

4.2 Multiobjective Evaluation Metrics

In practice, one might desire to optimize multiple objectives, such as translation accuracy and speed. Suppose we have J objectives, and they can be jointly represented as $F(\boldsymbol{\lambda}) = [f^1(\boldsymbol{\lambda}), f^2(\boldsymbol{\lambda}), \cdots, f^J(\boldsymbol{\lambda})]$. As it is unlikely that any one $\boldsymbol{\lambda}$ will optimize all objectives simultaneously, we adopt the concept of *Pareto optimality* (Godfrey et al., 2007). In the context of minimization, $\boldsymbol{\lambda}$ is said to dominate $\boldsymbol{\lambda}'$, that is, $\boldsymbol{\lambda} \prec \boldsymbol{\lambda}'$, if $f^j(\boldsymbol{\lambda}) \leq f^j(\boldsymbol{\lambda}')\ \forall j$ and $f^j(\boldsymbol{\lambda}) < f^j(\boldsymbol{\lambda}')$ for at least one j. If nothing dominates $\boldsymbol{\lambda}$, we call it the *Pareto optimal* solution. The set of all Pareto solutions is referred to as the *Pareto front*, that is, $\{\boldsymbol{\lambda} \mid \nexists \boldsymbol{\lambda}' \in \boldsymbol{\Lambda} : \boldsymbol{\lambda}' \prec \boldsymbol{\lambda}\}$. Intuitively, these are solutions satisfying all possible trade-offs in the multiobjective space. Figure 1 shows an

example of Pareto solutions that maximize BLEU and minimize speed.

For multiobjective optimization, the quality indicator becomes the Pareto front, thus we have:

- **fixed-target all (fta)**: We measure the runtime to find all points on the Pareto front.
- **fixed-target one (fto)**: We measure the runtime to get at least one Pareto point.
- **fixed-budget (fbp)**: We fix the budget of function evaluations and measure the number of Pareto-optimal points obtained.

In the literature, a common way to compare HPO methods is to plot quality indicator value as a function of runtime on a graph (e.g., see Figure 6). The proposed metrics can be viewed as summary statistics drawn as line thresholds on such graphs (Hansen et al., 2016), where the budget/target is set to a value appropriate for the use case.

4.3 Repeated Trials

Some HPO methods may be sensitive to randomness in initial seeds $\{\boldsymbol{\lambda}_{init}\}$ (Feurer et al., 2015). We suggest that repeated randomized trials are important for a rigorous evaluation, and this is only feasible with a table-lookup dataset. In our experiments, we average results of HPO runs across 100 trials, where each trial is seeded with a different set of 3 random initial hyperparameter settings.

5 Methods

We now describe two HPO/SMBO methods used in our experiments: Bayesian optimization[10] is a popular method. Graph-based SMBO is a novel method that adapts ideas in graph-based semi-supervised learning to the HPO problem.

5.1 Bayesian Optimization (BO)

Given a target function $f : \boldsymbol{\Lambda} \rightarrow \mathbb{R}$, Bayesian optimization (Brochu et al., 2010; Shahriari et al., 2015; Frazier, 2018) aims to find an input $\boldsymbol{\lambda}_\star \in \arg\min_{\boldsymbol{\lambda}\in\boldsymbol{\Lambda}} f(\boldsymbol{\lambda})$. It models f with a posterior probability distribution $p(f \mid \boldsymbol{\mathcal{L}})$, where $\boldsymbol{\mathcal{L}}$ is a set of observed points. This posterior distribution is updated each time we observe f at a new point $\boldsymbol{\lambda}_p$. The *utility* of each candidate point is quantified by an acquisition function $a : \boldsymbol{\Lambda} \rightarrow \mathbb{R}$, and $\boldsymbol{\lambda}_p \in \arg\max_{\boldsymbol{\lambda}\in\boldsymbol{\Lambda}} a(\boldsymbol{\lambda})$. In practice, a prominent choice for $p(f \mid \boldsymbol{\mathcal{L}})$ is Gaussian process regression, and a common acquisition function is Expected Improvement (EI).

5.1.1 Gaussian Process Regression

A Gaussian Process (GP) (Rasmussen, 2003) $\mathcal{G}(m(\boldsymbol{\lambda}), k(\boldsymbol{\lambda}, \boldsymbol{\lambda}'))$ is a collection of random variables such that any finite subset of them follows a multivariate Gaussian distribution. A GP is fully specified by a mean $m(\boldsymbol{\lambda})$ and a covariance function or a *kernel* $k(\boldsymbol{\lambda}, \boldsymbol{\lambda}')$, and the sufficient statistics of the posterior predictive distribution, $\mu(\cdot)$[11] and $\Sigma(\cdot)$, can be computed with

$$\mu(\boldsymbol{\lambda}) = \mathbf{K}_\star^T \mathbf{K}^{-1} \mathbf{y}, \quad (1)$$

$$\Sigma(\boldsymbol{\lambda}) = k(\boldsymbol{\lambda}, \boldsymbol{\lambda}) - \mathbf{K}_\star^T \mathbf{K}^{-1} \mathbf{K}_\star, \quad (2)$$

where $\mathbf{y} = [\ldots; f(\boldsymbol{\lambda}); \ldots,]$, $\mathbf{K}_\star = k(\boldsymbol{\Lambda}_{observed}, \boldsymbol{\lambda})$ and $\mathbf{K} = k(\boldsymbol{\Lambda}_{observed}, \boldsymbol{\Lambda}_{observed})$. In the case of HPO, the kernel $k()$ measures the similarity between hyperparameter configurations and $\mu()$ is a prediction of the $f()$ values of not-evaluated hyperparameters.

5.1.2 Expected Improvement (EI)

The EI score (Schonlau et al., 1998) is defined as:

$$a_{EI}(\boldsymbol{\lambda}) = \mathbb{E}[max(\hat{f}(\boldsymbol{\lambda}) - f_{min}, 0)], \quad (3)$$

where f_{min} is the best observed value thus far, and $\hat{f}(\boldsymbol{\lambda}) = \mu(\boldsymbol{\lambda})$. When the prediction $\hat{f}(\boldsymbol{\lambda})$ follows a normal distribution as in the GP, EI can be computed in a closed form. Our acquisition function computes EI for each point in the grid of hyperparameters, and queries the one with largest value.

5.2 Graph-Based SMBO (GB)

Semi-supervised learning addresses the question how to utilize a handful of labeled data and a large amount of unlabeled data to improve prediction accuracy. Graph-based semi-supervised learning (GBSSL, Zhu et al., 2003; Zhu, 2005) describes the structure of data with a graph, where each vertex is a data point and each weighted edge reflects the similarity between vertices. It makes a *smoothness* assumption that neighbors connected

[10] There are works adopting Bayesian optimization for HPO of statistical machine translation systems (Miao et al., 2014; Beck et al., 2016).

[11] For simplicity, we assume a mean of 0 for the prior.

by edges tend to have similar labels, and labels can propagate throughout the graph.

In SMBO surrogate modeling, we hope to make predictions for the unlabeled or not-evaluated points in the hyperparameter space based on the information of labeled or evaluated points. If we pre-define the set of all potential points, then this becomes highly related to semi-supervised learning. From this point of view, we propose GBSSL equipped with suitable acquisition functions as a new SMBO method for searching over a grid of representative hyperparameter configurations.

5.2.1 Graph-Based Regression

Suppose we have a graph $G = (V, E)$ with nodes V corresponding to n points, of which $\boldsymbol{\mathcal{L}}$ denotes the set of labeled points $\{(\boldsymbol{\lambda}_1, f(1)), \cdots, (\boldsymbol{\lambda}_l, f(l))\}$, where $f(i)$ is short for $f(\boldsymbol{\lambda}_i)$, and $\boldsymbol{\mathcal{U}}$ denotes the set of unlabeled points $\{\boldsymbol{\lambda}_{l+1}, \cdots, \boldsymbol{\lambda}_{l+u}\}$, where $n = l + u$. The edges E are represented by a $n \times n$ weight matrix W. For instance, W can be given as the radial basis function (RBF):

$$w_{ij} = \exp\left(-\frac{1}{2\sigma^2}\sum_{d=1}^{H}(\boldsymbol{\lambda}_{id} - \boldsymbol{\lambda}_{jd})^2\right). \quad (4)$$

Note that G is not necessarily fully connected; in practice, kNN graphs with a small k turn out to perform well, where nodes i, j are connected if i is in j's k-nearest-neighborhood or vice versa.[12]

Because closer points are assumed to have similar labels, we define the *energy* function as:

$$E(f) = \frac{1}{2}\sum_{i,j} w_{ij}(f(i) - f(j))^2, \quad (5)$$

and we constrain $f(i), i \in L$ or f_L to be true labels and aim to find $f(i), i \in U$ or f_U that minimizes the energy.

We define a diagonal matrix D, where $D_{ii} = \sum_j W_{ij}$ and the *combinatorial Laplacian* $\boldsymbol{\Delta} = D - W$, Equation (5) can then be rewritten to $E(f) = \boldsymbol{f}^T\boldsymbol{\Delta}\boldsymbol{f}$. If we partition the Laplacian matrix into blocks:

$$\boldsymbol{\Delta} = \begin{bmatrix} \Delta_{LL} & \Delta_{LU} \\ \Delta_{UL} & \Delta_{UU} \end{bmatrix}, \quad (6)$$

we can predict the $f()$ values for unlabeled points by:

$$f_U = -\boldsymbol{\Delta}_{UU}^{-1}\boldsymbol{\Delta}_{UL}f_L. \quad (7)$$

5.2.2 Expected Influence (EIF)

We propose a novel acquisition function called *expected influence* that exploits the graph structure. The idea is to query the point such that, if its $f()$ is observed, has the highest potential to change the $f()$ of all other points as we re-run label propagation through the graph.

We first scale the labels on the graph $f(i) \in \mathbb{R}$ to be between 0 or 1. The best labeled point is set to 1; for the other labeled points, we first compute the probability that a random walk starting at 1 reaches it, then set the label to be 1 if the probability is larger than 0.5 and 0 otherwise.

If we were to query an unlabeled point k, there are two scenarios: Its label is either 1 with probability $f(k)$ or 0 with probability $1-f(k)$. For each scenario, we then consider including k as a newly added ''labeled'' point and re-running label propagation. $f^{+(\boldsymbol{\lambda}_k,1)}(i)$ are the new predictions for points i in the scenario where k is added with label 1. If k is an influencer in the positive direction, this means that many points i will now have large $f^{+(\boldsymbol{\lambda}_k,1)}(i)$; otherwise, $f^{+(\boldsymbol{\lambda}_k,1)}(i)$ might be small on average in magnitude. On the other hand, suppose we add k with label 0 and run label propagation again to obtain new predictions $f^{+(\boldsymbol{\lambda}_k,0)}(i)$. If k is an influencer in the negative direction, this means that $f^{+(\boldsymbol{\lambda}_k,0)}(i)$ will be small (or conversely $1 - f^{+(\boldsymbol{\lambda}_k,0)}(i)$ will be large).

We can now define an influence score and have the acquisition function seeking point p that maximizes the following:

$$\begin{aligned} a_{EIF}(\boldsymbol{\lambda}_k) &= (1 - f(k))\sum_{i=1}^{n}(1 - f^{+(\boldsymbol{\lambda}_k,0)}(i)) \\ &\quad + f(k)\sum_{i=1}^{n} f^{+(\boldsymbol{\lambda}_k,1)}(i) \end{aligned} \quad (8)$$

Intuitively, we try adding each unlabeled point as either a desirable point (label 1) or undesirable point (0). We measure whether this addition changes the result of GB regression, and finally query the hyperparameter that is expected to cause the most significant change.

5.3 BO vs. GB

There is a connection between the BO and GB due to the link between GPs and graphs. The

[12] In experiments, based on initial tuning, we set kNN so that each point has on average $\frac{n}{7}$ neighbors.

Algorithm 1: Multiobjective SMBO

Input : Initial seeds $\{\boldsymbol{\lambda}_{init}\}$, Budget B
Output: Pareto-front approximation $\mathcal{P}$

$\mathcal{L} \leftarrow \{\cdots(\boldsymbol{\lambda}_{init}, F(\boldsymbol{\lambda}_{init}))\cdots\}$
while $b <= B$ **do**
  $\mathcal{P} \leftarrow$ Compute the Pareto front of $\mathcal{L}$
  Fit surrogate models $\hat{f}^1, \cdots, \hat{f}^J$ on $\mathcal{L}$
  Select a new point $\boldsymbol{\lambda}_p$ based on an infill criterion and surrogate model predictions
  $\mathcal{L} \leftarrow \mathcal{L} \cup \{(\boldsymbol{\lambda}_p, F(\lambda_p))\}$
end
return $\mathcal{P}$

GB method defines a Gaussian random field on the graph, which is a multivariate Gaussian distribution on the nodes. This is equivalent to ''finite set'' GPs. Zhu (2005) showed that the kernel matrix K of the finite set GP is equivalent to the inverse of a function of the graph Laplacian $\boldsymbol{\Delta}$, that is, $K = (2\beta(\boldsymbol{\Delta} + \frac{I}{\sigma^2}))^{-1}$[13]. The difference between the finite set GP and GP is that the kernel matrix of the former is defined on $\boldsymbol{\mathcal{L}} \cup \boldsymbol{\mathcal{U}}$, while the latter is defined on $\boldsymbol{\Lambda}$. As a semi-supervised method, the label propagation rule of GB (Equation (7)) shows that all the nodes on the graph contribute to the prediction of a single unlabeled node, whereas for GP, the posterior predictive distribution of a new point does not depend on other unlabeled points as shown by Equation (1).

The main advantage of GB is that it offers flexibility to build graphs over the search space. For instance, one can build a graph with configurations from different model architectures, for example, RNN, CNN, and Transformers. Nodes of the same architecture might gather into a cluster, and clusters can be connected with each other. One can also manipulate the edge weights by manually defined heuristics. One example of such rules could be Euclidean distance scaled by hyperparameter importance. We leave this as future work.

The theoretical caveat of the GB method is that it is restricted to a discrete search space defined by a graph. If a dense grid is desired to mimic a continuous search space, increasing time and space complexity would make it a less efficient method.

5.4 Multiobjective Optimization

For multiobjective optimization, we can use the same surrogate models to estimate each $\hat{f}^j$ independently; but we need a new acquisition function that considers the Pareto front. Various methods have been proposed (Zitzler and Thiele, 1998; Ponweiser et al., 2008; Picheny, 2015; Shah and Ghahramani, 2016; Svenson and Santner, 2016). Here, we adopt the expected hypervolume improvement (EHVI) method (Emmerich et al., 2011), which is a generalization of EI. EHVI as an *infill criterion* and can be combined with different surrogate models. Algorithm 1 provides pseudo-code for the framework.

6 Experiments and Results

We evaluate HPO methods on six NMT tasks with the provided benchmark dataset and report their performance measured by three runtime-based assessment metrics mentioned in Section 4. The code base is provided to ensure reproducibility.[14]

6.1 Single-Objective Optimization

For single-objective optimization, our goal is to find a hyperparameter configuration giving the highest BLEU score over a predefined grid.

6.1.1 Experimental Comparison

We run the comparison with two surrogate models, two kernels,[15] and two acquisition functions, leading to the following HPO systems, where all the GB systems are introduced by this work:

- **RS**: random search (Bergstra and Bengio, 2012), which uniformly samples hyperparameter configurations at random over the grid.
- **BO_EI_M**: GP-based BO with Matérn52 covariance function and expected improvement as acquisition function.
- **BO_EI_R**: GP-based BO with RBF kernel and EI as acquisition function.
- **GB_EI_M**: GB with Matérn52 kernel and EI as acquisition function.[16]

[13] β and σ are adjustable parameters.

[14] https://github.com/Este11e/gbopt.

[15] We choose Matérn52 and RBF kernel because they exhibit different properties and are both frequently used in literature. As shown in Rasmussen (2003), a parameter ν of the Matérn class of covariance functions can affect the smoothness of the functions drawn from GP. For $\nu = \frac{1}{2}$, the process becomes very rough, and for $\nu \to \infty$, the covariance function converges to RBF kernel.

[16] We can make an equivalence between the covariance matrix in multivariate Gaussian distribution and the inverse of a function of the graph Laplacian $\boldsymbol{\Delta}$ (see Section 5.3 for details), so EI can also be applied to GB models.

	zh-en			ru-en			ja-en		
	ftb	**ftc**	**fb**	**ftb**	**ftc**	**fb**	**ftb**	**ftc**	**fb**
RS	61±34	14±11	0.26±0.25	79±47	20±17	0.42±0.29	71±43	16±15	0.40±0.24
BO_EI_M	29±19	13±9	0.24±0.24	41±19	26±17	0.51±0.36	27±17	16±15	0.39±0.45
BO_EI_R	24±15	11±8	0.22±0.26	40±26	20±13	0.44±0.37	20±11	13±9	0.33±0.44
GB_EI_M	84±15	13±8	0.35±0.21	50±34	18±17	0.35±0.25	23±7	**6**±3	0.14±0.11
GB_EI_R	86±15	12±7	0.33±0.20	51±32	18±17	0.35±0.28	21±6	**6**±3	0.10±0.12
GB_EIF_M	19±21	8±5	0.11±0.17	32±18	22±13	0.46±0.31	**13**±4	**6**±2	**0.01**±0.04
GB_EIF_R	**13**±20	**6**±4	**0.06**±0.15	**28**±17	**17**±12	**0.33**±0.30	**13**±3	**6**±2	**0.01**±0.05
	en-ja			**sw-en**			**so-en**		
	ftb	**ftc**	**fb**	**ftb**	**ftc**	**fb**	**ftb**	**ftc**	**fb**
RS	71±46	12±10	0.71±0.37	334±201	186±152	2.45±0.97	301±161	39±39	0.63±0.32
BO_EI_M	60±29	15±17	0.86±0.60	**33**±17	**29**±17	1.60±1.41	65±62	19±21	0.41±0.36
BO_EI_R	62±36	13±12	0.79±0.58	55±47	33±24	**1.42**±1.33	52±70	**13**±11	**0.24**±0.30
GB_EI_M	**22**±20	11±11	**0.42**±0.57	63±37	62±36	3.56±0.95	187±99	61±28	1.17±0.44
GB_EI_R	24±21	13±12	0.47±0.59	56±26	55±26	3.39±0.95	201±104	62±29	1.16±0.44
GB_EIF_M	47±22	**9**±7	0.63±0.32	58±24	57±24	3.13±0.51	**42**±30	26±8	0.48±0.13
GB_EIF_R	45±22	10±7	0.69±0.39	59±25	58±25	3.15±0.52	**42**±30	28±7	0.49±0.12

Table 3: Evaluation on NMT models trained with different language pairs for single-objective (BLEU) optimization. Results are averaged over 100 trials and standard deviations are also reported. Fixed-target best (ftb) and fixed-target close (ftc) are measured by number of model evaluations, and fixed-budget (fb) is measured by BLEU difference. For ftc, the tolerance of performance degradation is set to 0.5 BLEU.[17] For fb, the runtime budget is set to 20.[18]

- **GB_EI_R**: GB with RBF kernel and EI.
- **GB_EIF_M**: GB with Matérn52 kernel and expected influence as acquisition function.
- **GB_EIF_R**: GB with RBF and EIF.

We use the George library (Ambikasaran et al., 2014) for GP implementation. For all the methods, configurations are sampled without replacement.

6.1.2 Results

Results for single-objective optimization are summarized in Table 3:

- RS always needs to explore roughly half of all the NMT models to get the best one (ftb).
- The effectiveness of BO is confirmed: On sw-en, BO_EI_M only takes 10% of the runtime used by RS to achieve the optima.
- For ftb, the best GB outperforms the best BO on four of the six datasets: on en-ja, GB_EI_M reduces the ftb runtime of BO_EI_M by 38. GB_EIF often works better than GB_EI.
- Matérn kernel and RBF kernel are almost equally good for both BO and GB.
- Adjusting initialization can result in a noticeable variance on performance. We suggest that researchers experiment with enough random trials when evaluating HPO systems.

6.2 Multiobjective Optimization

We now show benchmarks for multiobjective optimization. Our goal is to search for configurations achieving higher BLEU and less decoding time.

6.2.1 Experimental Comparison

We run the comparison on the following systems, where GB systems are introduced by this work:

- **RS**: random search, uniformly samples the configurations at random.
- **BO_M**: GP-based BO equipped with Matérn kernel and EHVI as the infill criterion.
- **BO_R**: GP-based BO with RBF kernel and EHVI.
- **GB_M**: GB equipped with Matérn kernel and EHVI as the infill criterion.
- **GB_R**: GB with RBF kernel and EHVI.

[17] Except for en-ja, where tolerance is set to 1 BLEU, because BLEU difference between top two models is > 0.5.

[18] Including three initial evaluations.

	zh-en			ru-en			ja-en		
	fto	**fta** (J=3)	**fbp** (B=50)	**fto**	**fta** (J=4)	**fbp** (B=50)	**fto**	**fta** (J=5)	**fbp** (B=50)
RS	30±24	88±22	1.3±0.8	33±26	139±28	1.3±0.9	21±18	129±20	1.7±1.0
BO_M	24±16	81±16	1.7±0.7	**16**±14	**80**±26	**2.4**±0.9	17±13	**77**±28	**3.3**±1.3
BO_R	**20**±13	**75**±15	**1.8**±0.5	17±15	84±32	**2.4**±1.0	18±14	94±32	2.8±1.2
GB_M	24±16	85±16	**1.8**±0.6	17±14	102±30	1.9±0.9	**16**±12	103±21	2.4±1.1
GB_R	24±15	90±12	1.7±0.6	17±12	103±30	2.0±0.9	19±12	107±20	2.2±1.0
	en-ja			**sw-en**			**so-en**		
	fto	**fta** (J=8)	**fbp** (B=50)	**fto**	**fta** (J=14)	**fbp** (B=200)	**fto**	**fta** (J=7)	**fbp** (B=200)
RS	17±16	150±17	2.5±1.4	54±51	719±47	3.4±1.7	88±73	534±55	2.1±1.3
BO_M	**15**±10	100±34	**4.6**±1.7	**26**±20	**344**±201	**12.0**±2.8	**30**±21	**321**±113	**5.1**±1.2
BO_R	17±13	**93**±30	4.3±2.0	28±27	454±153	10.0±2.2	31±25	399±129	4.7±1.4
GB_M	17±13	121±28	4.0±1.5	59±75	469±198	7.8±4.3	61±63	447±99	2.9±1.4
GB_R	17±14	119±24	3.6±1.5	58±75	509±193	7.4±4.1	66±58	426±102	2.9±1.4

Table 4: Evaluation on NMT models trained with different language pairs for multiobjective (BLEU & decoding time) optimization. Fixed-target one (fto) and fixed-target all (fta) are measured by number of model evaluations, and fixed-budget (fbp) is measured by number of Pareto-optimal points. J is the size of the true Pareto set and B is the runtime budget.[19]

6.2.2 Results

The multiobjective optimization evaluation results are summarized in Table 4:

- RS is a bad choice for multiobjective optimization, if one aims to quickly collect as many Pareto-optimal configurations as possible: To get all the true optima, RS usually needs to go through the whole search space (fta), and with fixed budget it obtains much fewer Pareto points than other methods (fbp).
- BO is generally superior across datasets. On sw-en, it only spends less than half of the time that RS takes to get the Pareto set (344 vs. 719), and can find 8.6 more Pareto points than RS with 200 NMT models evaluated.
- GB provides comparable performance as BO on four datasets, whereas on sw-en and so-en, BO noticeably outperforms GB, which might not be a perfect solution for a multiobjective task.

7 Analysis

7.1 HPO Algorithm Behavior

Section 6 shows how to rigorously compare HPO methods based on various performance metrics. Here we illustrate examples of how to obtain deeper insights into HPO algorithm behavior using the table-lookup framework.

For single-objective optimization, we compare the best BLEU and mean squared error (MSE), which is the averaged squared difference between ground-truth BLEU and predictions, achieved by different HPO methods across time. We can see from Figure 6 (left) that BO and GB converge much faster than RS, and GB is superior over time. This could be partly explained by Figure 6 (right), GB can already fit the data well in the beginning, while BO starts from a much larger MSE and decreases gradually.

For multiobjective optimization, we show the evolution of Pareto-optimal fronts in Figure 7. There is a trend that Pareto fronts are moving towards the lower right corner at each iteration, verifying the effectiveness of our HPO methods.

7.2 Effect of Random Initialization

NMT training might not be deterministic due to the random initialization of model parameters. All the experimental results so far are obtained by a single run using one random seed. In order to explore the variance of the model performance induced by initialization effects, we fix the hyperparameter configurations and train models initialized with various random seeds. Specifically, we select five hyperparameter configurations,[20] and re-trained them for additional five times each with different random initializations. We did this for two datasets: the low-resource sw-en task and the larger WMT2019 ja-en task.

[19] Budget is adjusted based on the size of search space.

[20] Four of these are randomly selected. We also include the configuration that achieved the best BLEU in Table 2.

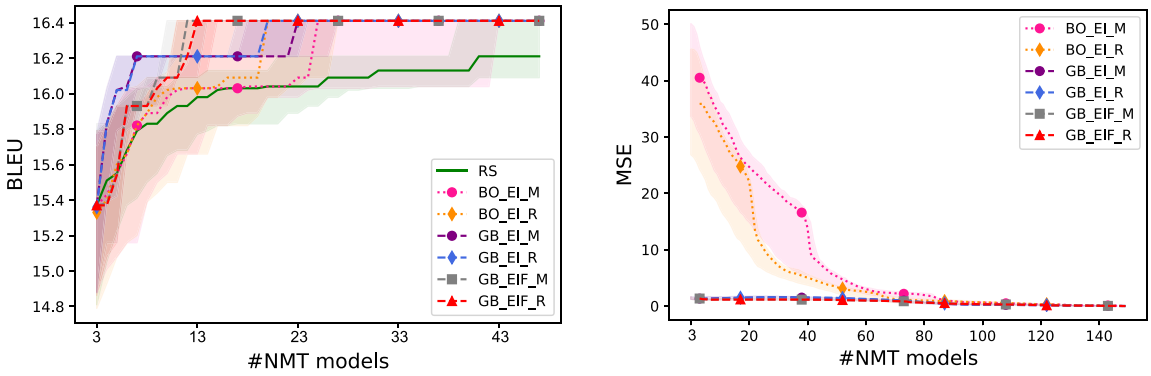

Figure 6: Left: Best BLEU found by different HPO methods over time on ja-en NMT models. Right: Mean squared error achieved by different HPO methods over time on ja-en NMT models. We plot the median and the 25th and 75th quantile across 100 independent runs.

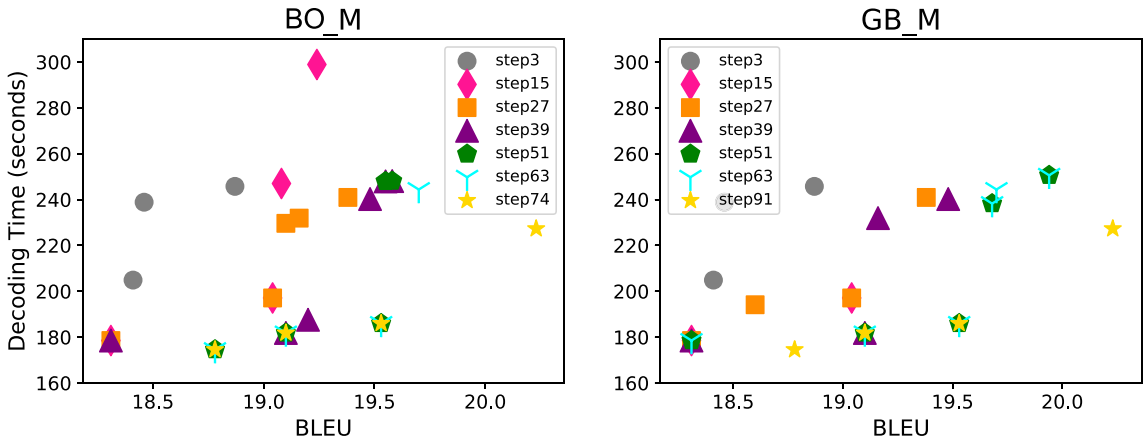

Figure 7: Pareto-front approximation during multiobjective optimization using BO_M and GB_M on ru-en. ‘‘Step’’ is the number of evaluated MT models. Gray circles form the Pareto set of initial seeds. In this example, all three initial seeds happen to be Pareto points. Gold stars are the Pareto solutions of the dataset. Lower-right corner is better.

The results on ja-en and sw-en are shown in Figure 8. The variance of performance is kept in a small range in most cases and the ranking of configurations remains about the same when different random seeds are applied. Based on this observation, we think that it is a reasonable strategy to use a single run to build table-lookup datasets; but at the same time it should be understood that the BLEU scores in the lookup table are only approximations. We note that there can be a few cases where variance is large, and this might be best addressed by inventing HPO methods that explicitly accounts for such uncertainty.

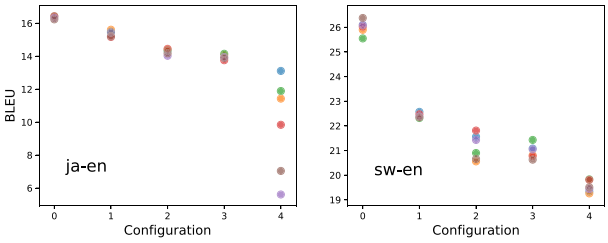

Figure 8: BLEU of ja-en and sw-en models trained with six random seeds. Circles with different colors stand for different random seeds.

8 Related Work

To alleviate the computational burden for benchmarking HPO methods and to improve research reproducibility, several studies have explored the table-lookup framework. Klein and Hutter (2019) published a mix of datasets focusing on feed forward neural networks. Ying et al. (2019) released a dataset of convolutional architectures for image classification problems. To the best of our knowledge, this work is the first that focuses on NMT and transformer models.

One challenge with table-lookup is that sufficient coverage of the hyperparameter grid is assumed. Eggensperger et al. (2015) and Klein et al. (2019) propose using a predictive meta-model trained on a table-lookup benchmark to approximate hyperparameters that are not in the table. This is an interesting avenue for future work.

Studies on HPO for NMT are scarce. Qin et al. (2017) propose an evolution strategy–based HPO method for NMT. So et al. (2019) apply NAS to Transformer on NMT tasks. There is also work on empirically exploring hyperparameters and architectures of NMT systems (Bahar et al., 2017; Britz et al., 2017; Lim et al., 2018), though the focus is on finding general best-practice configurations. This differs from the goal of HPO, which aims to find the best configuration specific to a given dataset.

9 Conclusions

In this paper, we presented a benchmark dataset for hyperparameter optimization of neural machine translation systems. We provided multiple evaluation protocols and analysis approaches for comparing HPO methods. We benchmarked Bayesian optimization and a novel graph-based semi-supervised learning method on the dataset for both single-objective and multiobjective optimization. Our hope is that this kind of dataset will facilitate reproducible research and rigorous evaluation of HPO for complex and expensive models.

Acknowledgments

This work is supported in part by an Amazon Research Award and an IARPA MATERIAL grant. We are especially grateful to Michael Denkowski for helpful discussions and feedback throughout the project.

References

Sivaram Ambikasaran, Daniel Foreman-Mackey, Leslie Greengard, David W. Hogg, and Michael O'Neil. 2014. Fast direct methods for gaussian processes. *arXiv preprint arXiv:1403.6015*.

Parnia Bahar, Tamer Alkhouli, Jan-Thorsten Peter, Christopher Jan-Steffen Brix, and Hermann Ney. 2017. Empirical investigation of optimization algorithms in neural machine translation. *The Prague Bulletin of Mathematical Linguistics*, 108(1):13–25.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. In *Proceedings of the 3rd International Conference on Learning Representations*.

Rémi Bardenet, Mátyás Brendel, Balázs Kégl, and Michele Sebag. 2013. Collaborative hyperparameter tuning. In *Proceedings of the 30th International Conference on Machine Learning*.

Daniel Beck, Adrià de Gispert, Gonzalo Iglesias, Aurelien Waite, and Bill Byrne. 2016. Speed-constrained tuning for statistical machine translation using Bayesian optimization. *arXiv preprint arXiv:1604.05073*.

James Bergstra and Yoshua Bengio. 2012. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(Feb):281–305.

James S. Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. 2011. Algorithms for hyper-parameter optimization. In *Proceedings of the 25th Advances in Neural Information Processing Systems*.

Denny Britz, Anna Goldie, Minh-Thang Luong, and Quoc Le. 2017. Massive exploration of neural machine translation architectures. *arXiv preprint arXiv:1703.03906*.

Eric Brochu, Vlad M. Cora, and Nando De Freitas. 2010. A tutorial on Bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. *arXiv preprint arXiv:1012.2599*.

Han Cai, Tianyao Chen, Weinan Zhang, Yong Yu, and Jun Wang. 2018. Efficient architecture search by network transformation. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

Kevin Duh. 2018. The multitarget TED talks task. `http://www.cs.jhu.edu/~kevinduh/a/multitarget-tedtalks/`.

Russell C. Eberhart and Yuhui Shi. 1998. Comparison between genetic algorithms and particle swarm optimization. In *International Conference on Evolutionary Programming*.

Katharina Eggensperger, Frank Hutter, Holger Hoos, and Kevin Leyton-Brown. 2015. Efficient benchmarking of hyperparameter optimizers via surrogates. In *Proceedings of the 29th AAAI Conference on Artificial Intelligence*.

Michael T. M. Emmerich, André H. Deutz, and Jan Willem Klinkenberg. 2011. Hypervolume-based expected improvement: Monotonicity properties and exact computation. In *2011 IEEE Congress of Evolutionary Computation (CEC)*.

Matthias Feurer and Frank Hutter. 2019. Hyperparameter optimization. *In Automated Machine Learning*, pages 3–33. Springer.

Matthias Feurer, Jost Tobias Springenberg, and Frank Hutter. 2015. Initializing Bayesian hyperparameter optimization via meta-learning. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*.

Peter I. Frazier. 2018. A tutorial on Bayesian optimization. *arXiv preprint arXiv:1807.02811*.

Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N. Dauphin. 2017. Convolutional sequence to sequence learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*.

Parke Godfrey, Ryan Shipley, and Jarek Gryz. 2007. Algorithms and analyses for maximal vector computation. *The VLDB Journal—The International Journal on Very Large Data Bases*, 16(1):5–28.

Nikolaus Hansen, Anne Auger, Dimo Brockhoff, Dejan Tušar, and Tea Tušar. 2016. Coco: Performance assessment. *arXiv preprint arXiv:1605.03560*.

Felix Hieber, Tobias Domhan, Michael Denkowski, David Vilar, Artem Sokolov, Ann Clifton, and Matt Post. 2017. Sockeye: A toolkit for neural machine translation. *arXiv preprint arXiv:1712.05690*.

Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. 2011. Sequential model-based optimization for general algorithm configuration. In *Proceedings of the 5th International Conference on Learning and Intelligent Optimization*.

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

Aaron Klein, Zhenwen Dai, Frank Hutter, Neil Lawrence, and Javier Gonzalez. 2019. Meta-surrogate benchmarking for hyperparameter optimization. *arXiv preprint arXiv:1905.12982*.

Aaron Klein and Frank Hutter. 2019. Tabular benchmarks for joint architecture and hyperparameter optimization. *arXiv preprint arXiv: 1905.04970*.

Liam Li and Ameet Talwalkar. 2019. Random search and reproducibility for neural architecture search. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI)*.

Xian Li, Paul Michel, Antonios Anastasopoulos, Yonatan Belinkov, Nadir Durrani, Orhan Firat, Philipp Koehn, Graham Neubig, Juan Pino, and Hassan Sajjad. 2019. Findings of the first shared task on machine translation robustness. In *Proceedings of the Fourth Conference on Machine Translation*.

Robert Lim, Kenneth Heafield, Hieu Hoang, Mark Briers, and Allen Malony. 2018. Exploring hyper-parameter optimization for neural machine translation on gpu architectures. *arXiv preprint arXiv:1805.02094*.

Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. 2018a. Progressive neural architecture search. In *Proceedings of the European Conference on Computer Vision (ECCV)*.

Hanxiao Liu, Karen Simonyan, Oriol Vinyals, Chrisantha Fernando, and Koray Kavukcuoglu. 2018b. Hierarchical representations for efficient architecture search. In *International Conference on Learning Representations*.

Gang Luo. 2016. A review of automatic selection methods for machine learning algorithms and hyper-parameter values. *Network Modeling*

Analysis in Health Informatics and Bioinformatics, 5(1):18.

Yishu Miao, Ziyu Wang, and Phil Blunsom. 2014. Bayesian optimisation for machine translation. *arXiv preprint arXiv:1412.7180*.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Blue: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*.

Victor Picheny. 2015. Multiobjective optimization using gaussian process emulators via stepwise uncertainty reduction. *Statistics and Computing*, 25(6):1265–1280.

Wolfgang Ponweiser, Tobias Wagner, Dirk Biermann, and Markus Vincze. 2008. Multiobjective optimization on a limited budget of evaluations using model-assisted $\mathcal{S}$-metric selection. In *International Conference on Parallel Problem Solving from Nature*, pages 784–794. Springer.

Hao Qin, Takahiro Shinozaki, and Kevin Duh. 2017. Evolution strategy based automatic tuning of neural machine translation systems. In *Proceedings of the 14th International Workshop on Spoken Language Translation*.

Carl Edward Rasmussen. 2003. Gaussian processes in machine learning. In *Summer School on Machine Learning*, pages 63–71. Springer.

Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V. Le. 2019. Regularized evolution for image classifier architecture search. In *Proceedings of the AAAI Conference on Artificial Intelligence*.

Matthias Schonlau, William J. Welch, and Donald R. Jones. 1998. Global versus local search in constrained optimization of computer models. *Lecture Notes-Monograph Series* pages 11–25.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*.

Amar Shah and Zoubin Ghahramani. 2016. Pareto frontier learning with expensive correlated objectives. In *International Conference on Machine Learning*, pages 1919–1927.

Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P. Adams, and Nando De Freitas. 2015. Taking the human out of the loop: A review of Bayesian optimization. *Proceedings of the IEEE*, 104(1):148–175.

Dan Simon. 2013. *Evolutionary optimization algorithms*. John Wiley & Sons.

Jasper Snoek, Oren Rippel, Kevin Swersky, Ryan Kiros, Nadathur Satish, Narayanan Sundaram, Mostofa Patwary, M.R. Prabhat, and Ryan Adams. 2015. Scalable Bayesian optimization using deep neural networks. In *Proceedings of the 32nd International Conference on Machine Learning*.

David So, Quoc Le, and Chen Liang. 2019. The evolved transformer. In *Proceedings of the 36th International Conference on Machine Learning*.

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. 2014. Sequence to sequence learning with neural networks. In *Proceedings of the 28th Advances in Neural Information Processing Systems*.

Joshua Svenson and Thomas Santner. 2016. Multiobjective optimization of expensive-to-evaluate deterministic computer simulator models. *Computational Statistics & Data Analysis*, 94:250–264.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Proceedings of the 31st Advances in Neural Information Processing Systems*.

Chris Ying, Aaron Klein, Esteban Real, Eric Christiansen, Kevin Murphy, and Frank Hutter. 2019. Nas-bench-101: Towards reproducible neural architecture search. *arXiv preprint arXiv:1902.09635*.

Xiaojin Zhu. 2005. *Semi-supervised learning with graphs*. Ph.D. Thesis.

Xiaojin Zhu, Zoubin Ghahramani, and John D. Lafferty. 2003. Semi-supervised learning using gaussian fields and harmonic functions. In *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*.

Eckart Zitzler and Lothar Thiele. 1998. Multiobjective optimization using evolutionary algorithms—a comparative case study. In *International Conference on Parallel Problem Solving from Nature*.

Barret Zoph and Quoc V. Le. 2016. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*.