# Best Practices of Successive Halving on Neural Machine Translation and Large Language Models

**Xuan Zhang** xuanzhang@jhu.edu
**Kevin Duh** kevinduh@cs.jhu.edu
Department of Computer Science, Johns Hopkins University, Baltimore, Maryland, 21218, USA

**Abstract**

Hyperparameter optimization (HPO) enhances neural machine translation (NMT) models but demands substantial computational resources. Successive halving, a multi-fidelity HPO method, mitigates this by early stopping unpromising models and allocating more resources to promising ones. This method is particularly relevant for NMT and large language models, which are computationally intensive. However, successive halving relies on a noisy estimation of model performance and assumes that early performance is highly correlated with final performance. We introduce a table lookup benchmark dataset to study the reliability of successive halving and propose best practices for its application in NMT and large language models.

## 1 Introduction

Hyperparameter optimization (HPO) is crucial yet resource-intensive for transformer-based neural machine translation (NMT) models. Hyperparameters such as learning rate, optimizer, batch size, and the number of nodes in each layer significantly influence the achievement of a state-of-the-art (SOTA) system. According to the ARR Responsible NLP Research guidelines, presenting extensive tables of hyperparameters and the best-found values is essential in research publications. [1]

Recently, with the rise of large language models (LLMs), NMT built upon LLMs has shown promising results (Hendy et al., 2023; Zhu et al., 2023; Sia and Duh, 2023; Zhang et al., 2023b). Adapting LLMs to NMT tasks typically involves in-context learning and supervised fine-tuning. Given the abundance of parallel data, fine-tuning has proven to be more effective than in-context learning (Zhang et al., 2023b). Parameter efficient fine-tuning (PEFT), such as Low-Rank Adapter (LoRA, Hu et al., 2021), is often favored over full fine-tuning due to its efficiencyfewer parameters are trained while achieving comparable or superior performance. Despite the fixed architecture of LLMs during PEFT, new hyperparameters are introduced, including the LoRA rank and the specific parameters to tune, alongside traditional hyperparameters like batch size and learning rate.

NMT models, whether trained from scratch or fine-tuned from LLMs, require extensive computational time, often taking days or weeks to converge. This makes hyperparameter searches over a reasonable space challenging. For instance, if an NMT model takes 2 GPU days to train, tuning 5 hyperparameters with 3 different values each would result in a total of $3^5 * 2 = 486$ GPU days! Practitioners with limited computational resources are thus often forced to resort to manual tuning or random search instead of more systematic methods like grid search or advanced HPO algorithms, increasing the risk of unfair comparisons between systems.

Successive halving (Karnin et al., 2013; Jamieson and Talwalkar, 2016) accelerates HPO by terminating unpromising models early in a set of models trained in parallel, saving more resources with more aggressive early stopping strategies. It has shown effectiveness in computer vision (Li et al., 2018) and NLP tasks (Dodge et al., 2020).

[1] https://aclrollingreview.org/responsibleNLPresearch/

However, its effectiveness for training NMT models or adapting LLMs for NMT tasks remains unclear.

The termination decision in successive halving is heuristic, based on the ranking of model performance up to the current timestamp. It assumes that early performance is highly correlated with late performance, which may not always be true. This raises the question: Does this assumption hold for NMT? If not, can we make it more reliable without relying solely on this assumption?

This paper focuses on the effectiveness of successive halving for HPO in NMT models, whether trained from scratch or fine-tuned from an LLM. Our main contributions are summarized as follows:

- **Dataset:** We build a benchmark dataset, ***NMTLC***[2], to facilitate NMT HPO research. This dataset contains models trained from scratch and fine-tuned from LLMs, with recorded learning curves for various hyperparameter settings. In total, it comprises 2469 models trained on 9 different corpora, costing approximately 2519 GPU days. This is the first HPO benchmark dataset that contains NMT learning curves and features models fine-tuned from LLMs.
- **Evaluation:** We evaluate the effectiveness of successive halving for NMT HPO under different experimental setups.
- **Model:** We introduce a novel model for learning curve extrapolation, built upon the LCRankNet introduced in Wistuba and Pedapati (2020), and name it ***LCRankNet-v2***[3]. We aim to determine whether "looking into the predicted future" enhances the reliability of successive halving compared to "looking back to the completed past."

Our findings indicate that the initial assumption of successive halving-that early performance predicts late performancegenerally holds true for NMT HPO with appropriate setups.

## 2 Related work

### 2.1 Hyperparameter optimization

Hyperparameter optimization (HPO) aims to find the optimal hyperparameter configuration with minimal evaluations. HPO methods can be broadly classified into sequential and parallel approaches. Sequential methods, such as Bayesian optimization (Brochu et al., 2010; Shahriari et al., 2015; Frazier, 2018), evaluate one configuration at a time, using the results to inform subsequent evaluations. Parallel methods evaluate multiple configurations simultaneously; examples include population-based training (Jaderberg et al., 2017), CMA-ES, and successive halving (Karnin et al., 2013; Jamieson and Talwalkar, 2016). While most HPO methods are black-box approaches that treat the model training process as opaque, successive halving is a multi-fidelity method that leverages approximations. It uses smaller subsets of data or limits training time to obtain noisy measurements, thereby accelerating the search for optimal configurations.

### 2.2 Hyperparameter search for NMT

Research on HPO for NMT is limited. Qin et al. (2017) propose an evolution strategy-based HPO method for NMT. Zhang and Duh (2020) release a benchmark dataset (Section 4.1) for comparing HPO methods on NMT, focusing on models trained from scratch. Deb et al. (2022) use a glass-box method to analyze how hyperparameters influence NMT performance, highlighting its connection with HPO. Zhang et al. (2023a) present an HPO toolkit for NMT, implemented as a wrapper on top of the open-source Sockeye NMT software. This toolkit implements the Asynchronous Successive Halving Algorithm (Li et al., 2020), promoting configurations as soon as they are guaranteed to be in the top half, thus running successive halving asynchronously and effectively utilizing computational resources.

### 2.3 Learning curve extrapolation

Learning curve extrapolation aims to predict model performance later in training based on early checkpoints. Kolachina et al. (2012) model learning curves for statistical machine translation systems by fitting them to various power-law family functions. Domhan et al. (2015) use a weighted combination of parametric model families to model learning curves. Klein et al. (2022) build a Bayesian neural network, while Chandrashekaran and Lane (2017) propose an ensemble method, and Baker et al. (2017)

[2]NMTLC dataset: https://github.com/Este1le/hpo_nmt

[3]LCRankNet-v2 code: https://github.com/Este1le/hpo_nmt

use frequentist regression models for learning curve extrapolation. Adriaensen et al. (2024) propose a transformer pretrained on data generated from a prior, performing approximate Bayesian inference. Wistuba and Pedapati (2020) introduce LCRankNet, which encodes hyperparameters, dataset IDs, model architectures, and partial learning curves for performance prediction.

## 3 Successive halving

The goal of successive halving (Karnin et al., 2013; Jamieson and Talwalkar, 2016) is to efficiently find the optimal hyperparameter configuration within a given search space. Suppose we have $N$ configurations to explore. We begin by training all $N$ models, and at every $c$ checkpoints, we continue training only the top $\frac{1}{p}$ configurations based on their performance up to that point, discarding the rest. This process is repeated until only one configuration remains, which is then trained to convergence.

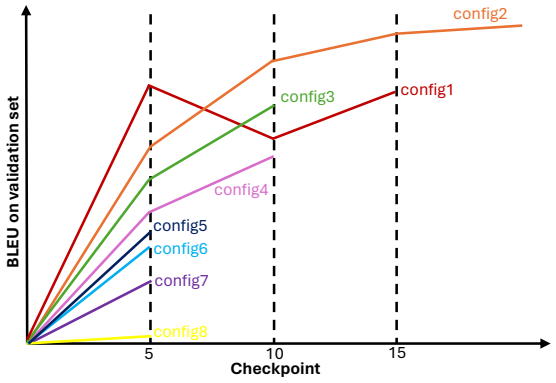


Figure 1: An example of successive halving, where $N = 10$, $c = 5$, $p = 2$.

As shown in Figure 1, we start with $N = 10$ configurations and halve ($p = 2$) the number of configurations every $c = 5$ checkpoint. Each cut is based on the best performance of the configurations up to the current checkpoint. For example, at checkpoint 10, when comparing config1 and config3, we compare config1's performance at checkpoint 5 with config3's performance at checkpoint 10.

In this example, assuming it takes one GPU day (20 checkpoints) for each model to converge, successive halving can reduce the total time for hyperparameter search from 10 days to 3.75 days. The aggressiveness of successive halving can be adjusted by changing the values of $p$ and $c$. For instance, if $p = 3$ and $c = 2$, the total time could be further reduced to 1.3 days. However, a more aggressive strategy increases the risk of discarding good configurations too early. In the case of $p = 3$ and $c = 2$, config1 might be chosen over config2, even if config2 could have performed better in the long run.

## 4 NMTLC benchmark datasets

To evaluate successive halving on NMT HPO, extensive model training until convergence is required to determine if good models are prematurely discarded. This process is resource-intensive, as each new configuration sampled for successive halving necessitates training a new NMT system from scratch. To facilitate this study, we have created a benchmark dataset that supports a table-lookup framework. We pre-train a large set of NMT systems and record their configurations and learning curves in a table. This allows for efficient evaluation of successive halving by looking up the table as needed, without training each model from scratch, significantly speeding up the experimental process.

Our dataset includes 2469 models trained on 9 different corpora, encompassing both models trained from scratch and those fine-tuned from LLMs, with a total computational cost of approximately 2519 GPU days. This is the first HPO benchmark dataset to contain NMT learning curves, enabling detailed studies on learning curves. It is also the first to include models fine-tuned from LLMs, facilitating HPO research on this emerging task.

| method | domain[4] | lang | train | dev | #cfg |
|---|---|---|---|---|---|
| scratch | IARPA | sw-en | 24k | 2675 | 767 |
| | IARPA | so-en | 24k | 2675 | 604 |
| | TED Talks | zh-en | 170k | 1,958 | 118 |
| | TED Talks | ru-en | 170k | 1,958 | 176 |
| | WMT19 | ja-en | 4M | 5,405 | 150 |
| | WMT19 | en-ja | 4M | 5,405 | 168 |
| FT | WMT23 | fr-en | 404k | 289 | 162 |
| | WMT23 | zh-en | 421k | 2139 | 162 |
| | WMT23 | de-en | 435k | 2342 | 162 |

Table 1: Data used for training NMT systems.

[4]IAPRA: IARPA MATERIAL; TED Talks: Duh (2018); WMT19: Li et al. (2019); WMT23: Neves et al. (2023).

| 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 2: Hyperparameter search space for trained-from-scratch NMT systems.

### 4.1 Data & setup for training from scratch

Zhang and Duh (2020) provided a HPO benchmark for NMT tasks, where models are trained from scratch without incorporating learning curves, primarily focusing on evaluating black-box HPO methods such as Bayesian optimization. From their data, we extracted hyperparameter configurations, evaluation results, and learning curves with perplexity on the development set for 1983 NMT models. Table 1 summarizes the 6 MT corpora used for training the systems. The data cover five language pairs and three domains with varying resource levels, from low to high. Table 2 presents the hyperparameter search space for different language pairs.

### 4.2 Data & setup for fine-tuning from LLMs

LLMs excel in most NLP tasks (Yang et al., 2024). Recently, fine-tuning LLMs for machine translation has shown promising results (Zhang et al., 2023b; Moslem et al., 2023; Zhu et al., 2024). Learning curves from fine-tuned models are rarely studied in the context of HPO and learning curve extrapolation, particularly for fine-tuned LLMs in machine translation. To address this gap, we include fine-tuned LLMs in our NMTLC benchmark datasets.

**MT Data:** We explore 3 language pairs as shown in Table 1. For fr-en, the input format is as follows:

> Translate French to English: French: [fr sent] English: [en sent] <eos>

A special <eos> token is added for post-processing.

**Hyperparameters:** We consider four hyperparameters to define the search space:

- **LLM (6):** BLOOMZ 560m, 1b7, and 3b, XGLM 564M, 1.7B, and 2.9B. BLOOMZ is a multilingual model fine-tuned with the xP3 dataset (Muennighoff et al., 2022). XGLM is a multilingual model trained on 30 diverse languages. We treat the choice of LLM as a hyperparameter since in practice the choice of the base model affects final MT accuracy. Various versions affect model size, feed-forward size, number of layers, and vocabulary size.
- **LoRA rank (3):** 2, 16, and 64.
- **Batch size (3):** 16, 32, and 64.
- **Learning rate (3):** $2e-5$, $1e-4$, and $2e-4$.

**Fine-tuning Setup:** We utilize QLoRA (Dettmers et al., 2023) for parameter-efficient fine-tuning. We set the LoRA scaling factor to 32, limit trainable parameters to the self-attention layers, and apply a dropout rate of 0.05 in the LoRA layer. The model weights are quantized to 4-bit precision, and mixed-precision training (using float16 and float32) is enabled to accelerate the process. We use the Adam optimizer, evaluating performance every 1000 steps, and consider the model converged when performance does not improve for 12 checkpoints. Models are trained on a single NVIDIA RTX GPU with 24GB of memory.

**Samples:** Each sample in the benchmark dataset includes:

1. Hyperparameter configuration.
2. Meta-information about the MT dataset (size, language pairs, domain).
3. Learning curve: a list of evaluation results (perplexity, BLEU[5]) on the development set throughout training until convergence.
4. Optimal performance: the best point on the learning curve.

For de-en, we provide perplexity learning curves. For fr-en and zh-en, we include both perplexity and BLEU learning curves to study the correlation between these metrics.

### 4.3 Statistics

We present the statistics of samples in the NMTLC benchmark dataset in this section.

**BLEU distribution** Figure 2 illustrates the performance variance of NMT models trained with differ-

[5]Obtained by greedy search.

ent hyperparameter configurations in the NMTLC dataset, measured by the BLEU score. Models trained from scratch (*scratch*) and those fine-tuned from LLMs (*ft*) exhibit distinct BLEU score distributions. The BLEU scores of the *scratch* models generally follow a left-skewed distribution, indicating that most configurations result in good performance. In contrast, the BLEU scores of the *ft* models display a multimodal distribution, suggesting a wide variation in performance, with many configurations yielding either very good or poor results. For instance, in *ft_fr-en*, the BLEU scores range widely, with differences up to 30 points between the best and worst models. Additionally, some configurations in almost all tasks (except *scratch_ja-en*) produce nearly zero BLEU scores, underscoring the importance of extensive hyperparameter search. This highlights the necessity of successive halving in efficiently exploring a large search space to find optimal hyperparameter configurations.

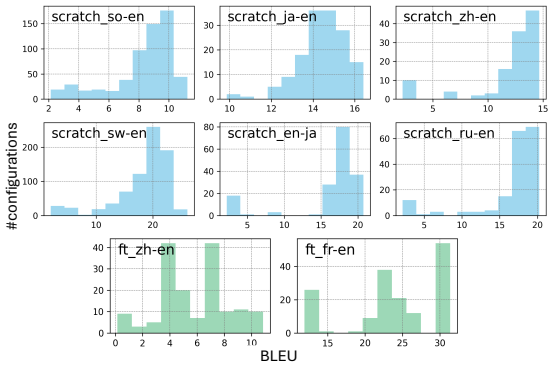


Figure 2: BLEU distribution on the hyperparameter search space.

**Length distribution** Figure 3 shows the distribution of the lengths of the learning curves in the NMTLC dataset, where longer curves indicate models that take more time to converge. The length distribution reveals that in most tasks, a small number of models have extended training times, resulting in a long right tail in the distribution. In these cases, successive halving can be particularly beneficial, as it can terminate unpromising models early in the training process, thereby saving substantial computational resources. Additionally, the length distributions vary across different tasks. While *scratch_ja-en*, *scratch_en-ja*, and *ft_zh-en* exhibit distributions similar to a normal distribution, other tasks display more left-skewed distributions. This variability further underscores the importance of using successive halving to efficiently navigate the diverse convergence behaviors and optimize hyperparameter configurations.

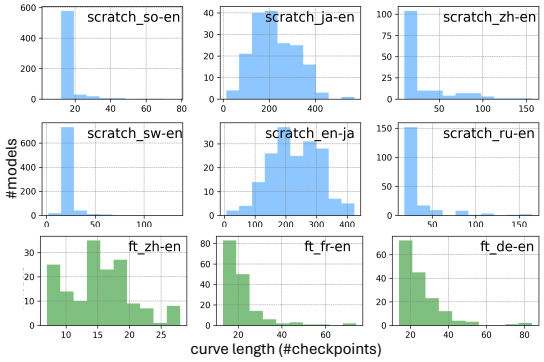


Figure 3: Learning curve length distribution on the hyperparameter search space.

## 5 On the reliability of successive halving on neural machine translation

To evaluate the reliability of successive halving in NMT, we begin by identifying an appropriate evaluation metric (perplexity vs. BLEU) for termination decisions (Section 5.1). We then investigate whether halving consistently retains the best-performing half of configurations at different learning curve lengths (Section 5.2.1). Finally, we conduct extensive successive halving runs on random subsets of the configuration search space to assess its ability to consistently select the best configuration (Section 5.2.2).

### 5.1 BLEU vs. perplexity

During training, models can be evaluated on the development set using either BLEU or perplexity. BLEU is more aligned with the ultimate goal of NMT, as BLEU scores are commonly reported for system comparison on development and test sets. However, perplexity is more closely aligned with the training objective and is significantly more efficient to compute. In our experiments, calculating perplexity is approximately 1000 times faster than BLEU on a single sentence, which means obtaining a BLEU score for an evaluation set can take hours. For HPO, we aim to select a configuration quickly while en-

suring it achieves the best BLEU score. This raises the question: can we use perplexity instead of BLEU for selection and termination decisions in successive halving to accelerate HPO?

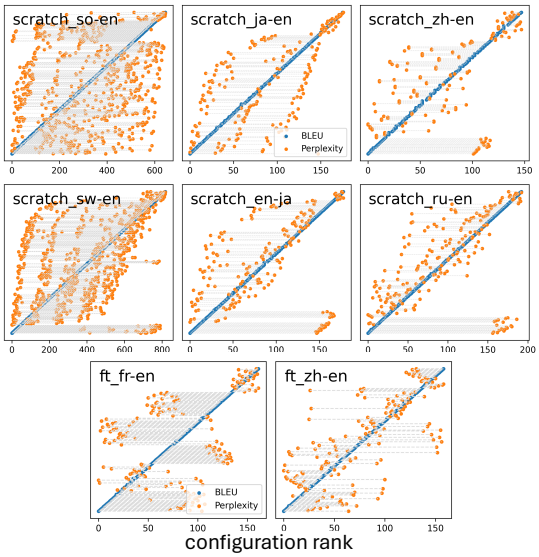


Figure 4: Configurations ranked by perplexity and BLEU. Configurations are ranked by their lowest perplexity on the development set and highest BLEU score, respectively.

Figure 4 shows the ranking of configurations by their best BLEU and perplexity scores on the development set. The results indicate that perplexity does not consistently align with BLEU across all datasets. For example, in *scratch_sw-en*, *scratch_en-ja*, and *scratch_ja-en*, configurations with the best BLEU scores (lower left) often have the worst perplexity. This suggests that perplexity may not be a suitable alternative to BLEU for model selection and early stopping in HPO for NMT tasks.

## 5.2 Successive halving on NMT

In this section, we evaluate the reliability of successive halving on NMT tasks.

### 5.2.1 Binary rank

In successive halving, at each checkpoint, the bottom half of the configurations are discarded based on their performance up to that point. To understand how the ranking of partial learning curves correlates with the full curves, we calculate Spearman's rank correlation coefficient ($\rho$) on the binary ranks of configurations at each checkpoint (Figure 5).

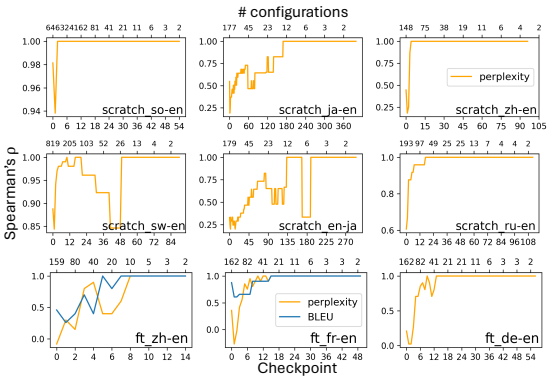


Figure 5: Spearman's rank correlation coefficient $\rho$ on binary ranks of learning curves at each checkpoint. At each checkpoint, learning curves are ranked based on their best performance (perplexity or BLEU on the development set) up to that point. Curves are assigned a rank of 0 if they are in the top half and 1 if they are in the bottom half. There are fewer longer learning curves, as shown in the figure, as the checkpoint number increases, the number of models (upper x-axis) decreases.

Generally, as the number of checkpoints increases, the correlation between the rankings of partial and full learning curves improves. This trend holds true for both perplexity and BLEU. Some datasets, such as *scratch_so-en*, *scratch_zh-en*, and *scratch_ru-en* for perplexity, and *ft_fr-en* for BLEU, achieve high correlation early in training.

### 5.2.2 Evaluation results

We run successive halving 100 times on randomly sampled subsets of hyperparameter configurations, varying $p$ and $c$ as shown in Table 3. The reliability of successive halving is measured by whether the best configuration is selected at the end (**acc**) and when the best configuration is discarded (**dif**).

Most runs achieve either perfect **acc** or a **dif** of around 1, indicating that the best configuration is usually selected, and if not, it is discarded near the final stage. Increasing the discarding aggressiveness by increasing $p$ and decreasing $c$ reduces reliability (lower **acc** and higher **dif**) unevenly across datasets– *fr-en(ft)* is significantly affected, while *so-en(st)* and *zh-en(st)* remain stable.

| | | p=2,c=10 | | p=2,c=5 | | p=4,c=10 | |
|---|---|---|---|---|---|---|---|
| | **average** | **acc** | **dif** | **acc** | **dif** | **acc** | **dif** |
| **st** | sw-en | 99 | 0 | 97 | 0 | 95 | 0 |
| | so-en | 100 | 0 | 100 | 0 | 100 | 0 |
| | zh-en | 100 | 0 | 100 | 0 | 100 | 0 |
| | ru-en | 100 | 0 | 96 | 0 | 100 | 0 |
| | ja-en | 69 | 0.2 | 67 | 0.1 | 68 | 0.1 |
| | en-ja | 77 | 0.1 | 69 | 0.2 | 70 | 0.1 |
| **ft** | fr-en | 69 | 1.2 | 11 | 3.6 | 54 | 0.9 |
| | zh-en | 100 | 0 | 83 | 0.7 | 100 | 0 |
| | de-en | 100 | 0 | 61 | 1.6 | 57 | 0.8 |

Table 3: Successive halving evaluation results. Each dataset runs successive halving 100 times on randomly selected 40 configurations. The discarding ratio $\frac{p-1}{p}$ and frequency $c$ checkpoints are varied. **Acc** indicates the percentage of runs where the best configuration is selected, and **dif** represents the average difference between total stages and the stage that discards the best configuration. A **dif** of 1 means the best configuration was discarded at the last stage.

## 6 Learning curve extrapolation

Successive halving uses the best performance observed so far (*BSF*) to rank configurations at each checkpoint, assuming early performance correlates with final performance. However, as shown in Figure 5, this correlation can be low when learning curves are short. To improve on the heuristic BSF, we explore "looking forward into the predicted future" by extrapolating the optimal performance of a configuration based on partial learning curves. This predicted optimal accuracy can then be used to rank configurations more effectively in successive halving.

### 6.1 LCRankNet-v2

Our learning curve extrapolation model, LCRankNet-v2, is a variation of LCRankNet (Wistuba and Pedapati, 2020). It takes three inputs: partial learning curves, hyperparameter configurations, and task meta-information (including dataset ID, task type, source and target language, and base model). The architecture is shown in Figure 6. We removed the architecture embedding component from LCRankNet since it is defined in the hyperparameter configuration in our settings. The experimental setup and configurations for LCRankNet-v2 are detailed in Appendix A.

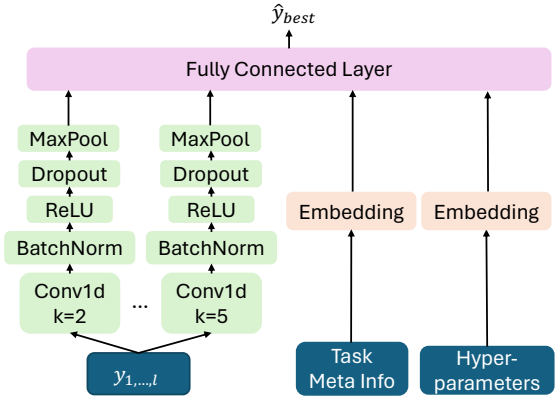


Figure 6: Architecture of LCRankNet-v2. Partial learning curves ($y_{1,\dots,l}$) are processed through convolutional layers with kernel sizes ranging from 2 to 5. Task meta-information and hyperparameter configurations are embedded and then combined with the curve features. The concatenated features are fed into fully connected layers to predict the best performance of the configuration ($\hat{y}_{best}$).

### 6.2 Training objectives

LCRankNet-v2 is trained using two loss functions: reconstruction loss $\mathcal{L}_{rec}$ and rank loss $\mathcal{L}_{rank}$. Given the true best performance $y^i_{best}$ and the prediction $\hat{y}^i_{best}$ for learning curve $i$, the reconstruction loss is:

$$\mathcal{L}_{rec} = \sum_i (y^i_{best} - \hat{y}^i_{best})^2. \tag{1}$$

The probability that configuration $i$ outperforms configuration $j$ is defined as:

$$p_{i>j} = \begin{cases} 1 & \text{if } y^i_{best} > y^j_{best} \\ 0.5 & \text{if } y^i_{best} = y^j_{best} \\ 0 & \text{if } y^i_{best} < y^j_{best} \end{cases} \tag{2}$$

The corresponding prediction is:

$$\hat{p}_{i>j} = \frac{e^{\hat{y}^i_{best} - \hat{y}^j_{best}}}{1 + e^{\hat{y}^i_{best} - \hat{y}^j_{best}}}. \tag{3}$$

The rank loss is a binary cross-entropy loss:

$$\mathcal{L}_{rank} = \sum_{i,j} -p_{i>j} \log \hat{p}_{i>j} - (1-p_{i>j}) \log(1-\hat{p}_{i>j}) \tag{4}$$

To ensure fair comparisons, we always compare partial learning curves of the same length when computing $\mathcal{L}_{rank}$. To handle curves of different lengths,

we include multiple truncated versions of each full learning curve in the training set. The total loss is:

$$\mathcal{L}_0 = w_{rec}\mathcal{L}_{rec} + w_{rank}\mathcal{L}_{rank}. \quad (5)$$

Additionally, we consider the *BSF* when ranking configurations. If the model predicts that performance will not improve beyond *BSF*, we set $\hat{y}_{best}$ to BSF. The probability of improvement $p^{imp}$ over BSF is defined similarly to $pi > j$, and the improvement loss $\mathcal{L}_{imp}$ is:

$$\mathcal{L}_{imp} = \sum_i -p_i^{imp}\log\hat{p}_i^{imp} - (1-p_i^{imp})\log(1-\hat{p}_i^{imp}). \quad (6)$$

The updated total loss is:

$$\mathcal{L}_1 = \mathcal{L}_0 + w_{imp}\mathcal{L}_{imp}. \quad (7)$$

During training, we set $w_{rec}$ to 1, $w_{rank}$ to 1000, and $w_{imp}$ to 100. At inference, if $\hat{p}_i^{imp} > 0.5$, we set $\hat{y}_{best}$ to *BSF*.

### 6.3 Experiment results

We conduct experiments to evaluate whether learning curve extrapolation improves the reliability of successive halving. Specifically, we compare the accuracy of ranking configurations on perplexity using LCRankNet-v2's predictions versus the heuristic *BSF*. LCRankNet-v2 was trained using a leave-one-out strategy, excluding the target dataset from the training data and warming up the network with 20 examples from the target dataset, as suggested by Wistuba and Pedapati (2020).

In Table 4, we compare the performance of the heuristic *BSF* and LCRankNet-v2 trained to minimize $\mathcal{L}_0$ in predicting the rank between two configurations given partial learning curves, where we consider all the possible pairs with the same length. There are four cases: both methods rank correctly (**B⭕P⭕**), both methods rank incorrectly (**B❌P❌**), or one is correct and the other is incorrect (**B❌P⭕** or **B⭕P❌**). On 7 out of 9 datasets, **B❌P⭕** is less than **B⭕P❌**, indicating that while LCRankNet-v2 can sometimes correct *BSF*'s mistakes, overall, *BSF* performs better.

When trained to minimize $\mathcal{L}_1$, LCRankNet-v2 converges to *BSF* on all datasets, resulting in **B❌P⭕** = **B⭕P❌** = 0, and **B⭕** = **P⭕**. Therefore, LCRankNet-v2 does not outperform the heuristic *BSF* in most of our settings.

| | acc | B⭕P⭕ | B❌P⭕ | B⭕P❌ | B❌P❌ |
|---|---|---|---|---|---|
| sw-en | 99.78% | 95.30% | 0.04% | 4.48% | 0.18% |
| so-en | 99.76% | 93.79% | 0.19% | 5.97% | 0.05% |
| zh-en | 75.73% | 63.07% | 17.61% | 12.66% | 6.63% |
| ru-en | 99.63% | 83.35% | 0.12% | 16.28% | 0.24% |
| ja-en | 95.86% | 73.19% | 2.35% | 22.67% | 2.08% |
| en-ja | 94.73% | 64.64% | 4.73% | 30.09% | 0.56% |
| fr-en | 84.43% | 57.69% | 6.94% | 26.64% | 8.73% |
| zh-en | 75.73% | 63.07% | 17.61% | 12.66% | 6.63% |
| de-en | 85.94% | 35.20% | 5.32% | 50.74% | 8.74% |

Table 4: Performance of LCRankNet-v2 trained with $\mathcal{L}_0$. **Acc** (or **B⭕**) indicates the accuracy of ranking configuration pairs based on *BSF*. $B$ represents ranking by *BSF* (vanilla successive halving), while $P$ represents ranking by LCRankNet-v2's prediction. If **B❌P⭕** > **B⭕P❌**, successive halving is more reliable with LCRankNet-v2's prediction.

**Is learning curve extrapolation necessary for successive halving on NMT?** Not really. In Table 4, **P** generally underperforms compared to **B** in ranking configurations. This suggests that incorporating learning curve extrapolation is unlikely to significantly alter the results of successive halving.

## 7 Conclusions

Successive halving is both efficient and effective for hyperparameter search in NMT tasks, significantly reducing computational resources and reliably selecting the best model with appropriate setups. However, its reliability depends on the target task and the choices of the cutting ratio ($p$) and cutting frequency ($c$). Based on the studies conducted in this paper, we propose the following **best practices for successive halving in NMT and LLMs**:

1. Rank configurations at each checkpoint based on BLEU rather than perplexity.
2. Before running an extensive hyperparameter search with successive halving, train several configurations to convergence to estimate training time and learning curve trends, which helps in determining appropriate values for $p$ and $c$.
3. Instead of keeping only one configuration at the end, increase the number of configurations that are trained to convergence (two might be sufficient, as our experiments suggest) to reduce the risk of discarding the best one at the last stage.

## Acknowledgements

This work is supported in part by an Amazon Initiative for Artificial Intelligence (AI2AI) Fellowship for Ph.D. students.

## References

Adriaensen, S., Rakotoarison, H., Müller, S., and Hutter, F. (2024). Efficient bayesian learning curve extrapolation using prior-data fitted networks. *Advances in Neural Information Processing Systems*, 36.

Baker, B., Gupta, O., Raskar, R., and Naik, N. (2017). Accelerating neural architecture search using performance prediction. *arXiv preprint arXiv:1705.10823*.

Brochu, E., Cora, V. M., and De Freitas, N. (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*.

Chandrashekaran, A. and Lane, I. R. (2017). Speeding up hyper-parameter optimization by extrapolation of learning curves using previous builds. In *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2017, Skopje, Macedonia, September 18–22, 2017, Proceedings, Part I 10*, pages 477–492. Springer.

Deb, K., Zhang, X., and Duh, K. (2022). Post-hoc interpretation of transformer hyperparameters with explainable boosting machines. In *Proceedings of the Fifth BlackboxNLP Workshop on Analyzing and Interpreting Neural Networks for NLP*, pages 51–61, Abu Dhabi, United Arab Emirates (Hybrid). Association for Computational Linguistics.

Dettmers, T., Pagnoni, A., Holtzman, A., and Zettlemoyer, L. (2023). Qlora: Efficient finetuning of quantized llms. *arXiv preprint arXiv:2305.14314*.

Dodge, J., Ilharco, G., Schwartz, R., Farhadi, A., Hajishirzi, H., and Smith, N. (2020). Fine-tuning pretrained language models: Weight initializations, data orders, and early stopping. *arXiv preprint arXiv:2002.06305*.

Domhan, T., Springenberg, J. T., and Hutter, F. (2015). Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves. In *Twenty-fourth international joint conference on artificial intelligence*.

Duh, K. (2018). The multitarget ted talks task. http://www.cs.jhu.edu/~kevinduh/a/multitarget-tedtalks/.

Frazier, P. I. (2018). A tutorial on bayesian optimization. *arXiv preprint arXiv:1807.02811*.

Hendy, A., Abdelrehim, M., Sharaf, A., Raunak, V., Gabr, M., Matsushita, H., Kim, Y. J., Afify, M., and Awadalla, H. H. (2023). How good are gpt models at machine translation? a comprehensive evaluation. *arXiv preprint arXiv:2302.09210*.

Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., and Chen, W. (2021). Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*.

Jaderberg, M., Dalibard, V., Osindero, S., Czarnecki, W. M., Donahue, J., Razavi, A., Vinyals, O., Green, T., Dunning, I., Simonyan, K., et al. (2017). Population based training of neural networks. *arXiv preprint arXiv:1711.09846*.

Jamieson, K. and Talwalkar, A. (2016). Non-stochastic best arm identification and hyperparameter optimization. In *Artificial intelligence and statistics*, pages 240–248. PMLR.

Karnin, Z., Koren, T., and Somekh, O. (2013). Almost optimal exploration in multi-armed bandits. In *International conference on machine learning*, pages 1238–1246. PMLR.

Klein, A., Falkner, S., Springenberg, J. T., and Hutter, F. (2022). Learning curve prediction with bayesian neural networks. In *International conference on learning representations*.

Kolachina, P., Cancedda, N., Dymetman, M., and Venkatapathy, S. (2012). Prediction of learning curves in machine translation. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 22–30.

Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A., and Talwalkar, A. (2018). Hyperband: A novel bandit-based approach to hyperparameter optimization. *Journal of Machine Learning Research*, 18(185):1–52.

Li, L., Jamieson, K., Rostamizadeh, A., Gonina, E., Ben-Tzur, J., Hardt, M., Recht, B., and Talwalkar, A. (2020).

A system for massively parallel hyperparameter tuning. *Proceedings of Machine Learning and Systems*, 2:230–246.

Li, X., Michel, P., Anastasopoulos, A., Belinkov, Y., Durrani, N., Firat, O., Koehn, P., Neubig, G., Pino, J., and Sajjad, H. (2019). Findings of the first shared task on machine translation robustness. In *Proceedings of the Fourth Conference on Machine Translation*.

Moslem, Y., Haque, R., and Way, A. (2023). Fine-tuning large language models for adaptive machine translation. *arXiv preprint arXiv:2312.12740*.

Muennighoff, N., Wang, T., Sutawika, L., Roberts, A., Biderman, S., Scao, T. L., Bari, M. S., Shen, S., Yong, Z.-X., Schoelkopf, H., et al. (2022). Crosslingual generalization through multitask finetuning. *arXiv preprint arXiv:2211.01786*.

Neves, M., Jimeno Yepes, A., NÃl'vÃl'ol, A., Bawden, R., Di Nunzio, G. M., Roller, R., Thomas, P., Vezzani, F., Vicente Navarro, M., Yeganova, L., Wiemann, D., and Grozea, C. (2023). Findings of the wmt 2023 biomedical translation shared task: Evaluation of chatgpt 3.5 as a comparison system. In *Proceedings of the Eighth Conference on Machine Translation*, pages 43–54, Singapore. Association for Computational Linguistics.

Qin, H., Shinozaki, T., and Duh, K. (2017). Evolution strategy based automatic tuning of neural machine translation systems. In *Proceedings of the 14th International Workshop on Spoken Language Translation*.

Shahriari, B., Swersky, K., Wang, Z., Adams, R. P., and De Freitas, N. (2015). Taking the human out of the loop: A review of bayesian optimization. *Proceedings of the IEEE*, 104(1):148–175.

Sia, S. and Duh, K. (2023). In-context learning as maintaining coherency: A study of on-the-fly machine translation using large language models. In *Proceedings of Machine Translation Summit XIV (Volume 1: Research Track)*.

Wistuba, M. and Pedapati, T. (2020). Learning to rank learning curves. In *International Conference on Machine Learning*, pages 10303–10312. PMLR.

Yang, J., Jin, H., Tang, R., Han, X., Feng, Q., Jiang, H., Zhong, S., Yin, B., and Hu, X. (2024). Harnessing the power of llms in practice: A survey on chatgpt and beyond. *ACM Transactions on Knowledge Discovery from Data*, 18(6):1–32.

Zhang, X. and Duh, K. (2020). Reproducible and efficient benchmarks for hyperparameter optimization of neural machine translation systems.

Zhang, X., Duh, K., and McNamee, P. (2023a). A hyperparameter optimization toolkit for neural machine translation research. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*, pages 161–168, Toronto, Canada. Association for Computational Linguistics.

Zhang, X., Rajabi, N., Duh, K., and Koehn, P. (2023b). Machine translation with large language models: Prompting, few-shot learning, and fine-tuning with qlora. In *Proceedings of the Eighth Conference on Machine Translation*, pages 468–481, Singapore. Association for Computational Linguistics.

Zhu, D., Chen, P., Zhang, M., Haddow, B., Shen, X., and Klakow, D. (2024). Fine-tuning large language models to translate: Will a touch of noisy data in misaligned languages suffice? *arXiv preprint arXiv:2404.14122*.

Zhu, W., Liu, H., Dong, Q., Xu, J., Kong, L., Chen, J., Li, L., and Huang, S. (2023). Multilingual machine translation with large language models: Empirical results and analysis. *arXiv preprint arXiv:2304.04675*.

## A LCRankNet-v2 Training Setup

We pad partial learning curves to a length of 450. The convolutional layers have an output channel size of 128. Each hyperparameter and task meta-information is embedded with a size of 2. The feed-forward layer size is set to 128. For regularization, we use a dropout rate of 0.1 and a weight decay of $10^{-3}$. The initial learning rate is set to $10^{-4}$, with Adam as the optimizer and cosine annealing as the learning rate scheduler. The minimum learning rate ($\eta_{min}$) is set to $10^{-7}$, and $T_{\max}$ is set to 10,000. Validation occurs every 1000 steps, and the batch size is 64. Training runs for 5 epochs.