Mixture-of-Supernets: Improving Weight-Sharing Supernet Training with Architecture-Routed Mixture-of-Experts

Anonymous ACL submission

Abstract

Weight-sharing supernets are crucial for performance estimation in cutting-edge neural architecture search (NAS) frameworks. Despite their ability to generate diverse subnetworks without retraining, the quality of these subnetworks is not guaranteed due to weight sharing. In NLP tasks like machine translation and pre-trained language modeling, there is a significant performance gap between supernet and training from scratch for the same model architecture, necessitating retraining post optimal architecture identification.

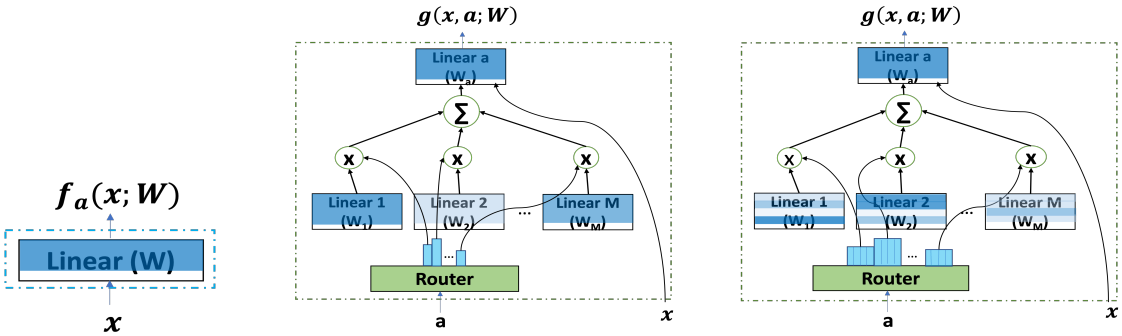
This study introduces a solution called *mixture-of-supernets*, a generalized supernet formulation leveraging mixture-of-experts (MoE) to enhance supernet model expressiveness with minimal training overhead. Unlike conventional supernets, this method employs an architecture-based routing mechanism, enabling indirect sharing of model weights among subnetworks. This customization of weights for specific architectures, learned through gradient descent, minimizes retraining time, significantly enhancing training efficiency in NLP. The proposed method attains state-of-the-art (SoTA) performance in NAS for fast machine translation models, exhibiting a superior latency-BLEU trade-off compared to HAT, the SoTA NAS framework for machine translation. Furthermore, it excels in NAS for building memory-efficient task-agnostic BERT models, surpassing NAS-BERT and AutoDistil across various model sizes.

1 Introduction

Neural architecture search (NAS) automates the design of high-quality architectures for natural language processing (NLP) tasks while meeting specified efficiency constraints (Wang et al., 2020a; Xu et al., 2021, 2022a). NAS is commonly treated as a black-box optimization (Zoph et al., 2018; Pham et al., 2018), but obtaining the best accuracy requires repetitive training and evaluation, which is impractical for large datasets. To address this, weight sharing is applied via a *supernet*, where subnetworks represent different model architectures (Pham et al., 2018).

Recent studies demonstrate successful direct use of subnetworks for image classification with performance comparable to training from scratch (Cai et al., 2020; Yu et al., 2020). However, applying this supernet approach to NLP tasks is more challenging, revealing a significant performance gap when using subnetworks directly. This aligns with recent NAS works in NLP (Wang et al., 2020a; Xu et al., 2021), which address the gap by retraining or finetuning the identified architecture candidates. This situation introduces uncertainties about the optimality of selected architectures and requires repeated training for obtaining final accuracy on the Pareto front, i.e., the best models for different efficiency (e.g., model size or inference latency) budgets. This work aims to enhance the weight-sharing mechanism among subnetworks to minimize the observed performance gap in NLP tasks.

The weight-sharing supernet is trained by iteratively sampling architectures from the search space and training their specific weights from the supernet. Standard weight-sharing (Yu et al., 2020; Cai et al., 2020) involves directly extracting the first few output neurons to create a smaller subnetwork (see Figure 1 (a)), posing two challenges due to limited model capacity. First, the supernet imposes strict weight sharing among architectures, causing co-adaptation (Bender et al., 2018; Zhao et al., 2021a) and gradient conflicts (Gong et al., 2021). For example, in standard weight-sharing, if a 5M-parameters model is a subnetwork of a 90M-parameters model, 5M weights are directly shared. However, the optimal shared weights for the 5M model may not be optimal for the 90M model, leading to significant gradient conflicts during optimization (Gong et al., 2021). Second, the supernet's



(a) Standard (b) Layer-wise Mixture-of-Supernet (c) Neuron-wise Mixture-of-Supernet

Figure 1: Choices of linear layers for supernet training. The length and the height of the 'Linear' blocks correspond to the number of input and output features of the supernet respectively. The highlighted portions in blue color correspond to the architecture-specific weights extracted from the supernet. Different intensities of blue color in the 'Linear' blocks of the mixture-of-supernet correspond to different alignment scores generated by the router.

Supernet	Weight sharing	Capacity	Overall Time (↓)	Average BLEU (↑)
HAT (Wang et al., 2020a)	Strict	Single Set	508 hours	25.93
Layer-wise MoS	Flexible	Multiple Set	407 hours (20%)	27.21 (4.9%)
Neuron-wise MoS	Flexible	Multiple Set	**394 hours (22%)**	**27.25 (5.1%)**

Table 1: Overall time savings and average BLEU improvements of MoS supernets vs. HAT for computing pareto front (latency constraints: 100 ms, 150 ms, 200 ms) for the WMT'14 En-De task. Overall time (single NVIDIA V100 hours) includes supernet training time, search time, and additional training time for the optimal architectures. Average BLEU is the average of BLEU scores of architectures in the pareto front (see Table 5 for individual scores). MoS supernets yield architectures that enjoy better latency-BLEU trade-offs than HAT and have an overall GPU hours (see A.5.10 for breakdown) savings of at least 20% w.r.t. HAT.

overall capacity is constrained by the parameters of a single deep neural network (DNN), i.e., the largest subnetwork in the search space. However, when dealing with a potentially vast number of subnetworks (e.g., billions), relying on a single set of weights to parameterize all of them could be insufficient (Zhao et al., 2021a).

To address these challenges, we propose a Mixture-of-Supernets (MoS) framework. MoS enables architecture-specific weight extraction, allowing smaller architectures to avoid sharing some output neurons with larger ones. Additionally, it allocates large capacity without being constrained by the number of parameters in a single DNN. MoS includes two variants: *layer-wise MoS*, where architecture-specific weight matrices are constructed based on weighted combinations of expert weight matrices at the level of sets of neurons, and *neuron-wise MoS*, which operates at the level of individual neurons in each expert weight matrix. Our proposed NAS method proves effective in constructing efficient task-agnostic BERT models (Devlin et al., 2019) and machine translation (MT) models. For efficient BERT, our best supernet outperforms SuperShaper (Ganesan et al., 2021) by 0.85 GLUE points, surpasses NAS-BERT (Xu et al., 2021) and AutoDistil (Xu et al., 2022a) in various model sizes ($\leq 50M$ parameters). Compared to HAT (Wang et al., 2020a), our top supernet reduces the supernet vs. standalone model gap by 26.5%, provides a superior pareto front for latency-BLEU tradeoff (100 to 200 ms), and decreases the steps needed to close the gap by 39.8%. A summary in the Table 1 illustrates the time savings and BLEU improvements of MoS supernets for the WMT'14 En-De task.

Main contributions: (1) We propose a formulation that generalizes weight-sharing methods, encompassing direct weight sharing (e.g., once-for-all network (Cai et al., 2020), BigNAS (Yu et al., 2020)) and flexible weight sharing (e.g., few-shot NAS (Zhao et al., 2021b)). This formulation enhances the expressive power of the supernet. **(2)** We apply the MoE concept to enhance model capability. The model's weights are dynamically generated based on the activated subnetwork architecture. Post-training, the MoE can be converted into equivalent static models as our supernets solely depend on the fixed subnetwork architecture after training. **(3)** Our experiments show that our supernets achieve SoTA NAS results in building efficient task-agnostic BERT and MT models.

2 Supernet - Fundamentals

A supernet, utilizing weight sharing, parameterizes weights for millions of architectures, offering rapid performance predictions and significantly reducing NAS search costs. The training objective can be formalized as follows. Let $\mathcal{X}_{tr}$ denote the training data distribution. Let x, y denote the training sample and label respectively, i.e., $x, y \sim \mathcal{X}_{tr}$. Let a_{rand} denote an architecture uniformly sampled from the search space $\mathcal{A}$. Let f_a denote the subnetwork with architecture a, and f be parameterized by the supernet model weights W. Then, the training objective of the supernet can be given by,

$$\min_{W} \mathbb{E}_{x,y\sim\mathcal{X}_{tr}} \mathbb{E}_{a_{rand}\sim\mathcal{A}}[\mathcal{L}(f_{a_{rand}}(x;W), y)]. \quad (1)$$

The mentioned formulation is termed single path one-shot (SPOS) optimization (Guo et al., 2020) for supernet training. Another popular technique is sandwich training (Yu et al., 2020), where the largest (a_{big}), smallest (a_{small}), and uniformly sampled architectures (a_{rand}) from the search space are jointly optimized.

3 Mixture-of-Supernets

Existing supernets typically have limited model capacity to extract architecture-specific weights. For simplicity, assume the model function $f_a(x;W)$ is a fully connected layer (output $o = Wx$, omitting bias term for brevity), where $x \in n_{in} \times 1$, $W \in n_{out} \times n_{in}$, and $o \in n_{out} \times 1$. n_{in} and n_{out} correspond to the number of input and output features respectively. Then, the weights ($W_a \in n_{out_a} \times n_{in}$) specific to architecture a with n_{out_a} output features are typically extracted by taking the first n_{out_a} rows[1] (as shown in Figure 1 (a)) from the supernet weight W. Assume one samples two architectures (a and b) from the search space with the number of output features n_{out_a} and n_{out_b} respectively. Then, the weights corresponding to the architecture with the smallest number of output features will be a subset of those of the other architecture, sharing the first $|n_{out_a} - n_{out_b}|$ output features exactly. This weight extraction technique enforces strict weight sharing between architectures, irrespective of their global architecture information (e.g., different features in other layers). For example, architectures a and b may have vastly different capacities (e.g., $5M$ vs $90M$ parameters). The smaller architecture (e.g., $5M$) must share all weights with the larger one (e.g., $90M$), and the supernet (modeled by $f_a(x;W)$) cannot allocate weights specific to the smaller architecture. Another issue with $f_a(x;W)$ is that the supernet's overall capacity is constrained by the parameters in the largest subnetwork (W) in the search space. Yet, these supernet weights W must parameterize numerous diverse subnetworks. This represents a fundamental limitation of the standard weight-sharing mechanism. Section 3.1 proposes a reformulation to overcome this limitation, implemented through two methods (Layer-wise MoS, Section 3.2, Neuron-wise MoS, Section 3.3), suitable for integration into Transformers (see Section 3.4).

[1]We assume the number of input features remains constant. If it changes, only the initial columns of W_a are extracted.

3.1 Generalized Model Function

We can reformulate the function $f_a(x;W)$ to a generalized form $g(x, a; E)$, which takes 2 inputs: the input data x, and the activated architecture a. E includes the learnable parameters of g. Then, the training objective of the proposed supernet becomes,

$$\min_{E} \mathbb{E}_{x,y\sim\mathcal{X}_{tr}} \mathbb{E}_{a_{rand}\sim\mathcal{A}}[\mathcal{L}(g(x, a_{rand}; E), y)]. \quad (2)$$

For the standard weight sharing mechanism mentioned above, $E = W$ and function g just uses a to perform the "trimming" operation on the weight matrix W, and forwards the subnetwork. To further minimize objective equation 2, enhancing the capacity of the model function g is a potential approach. However, conventional methods like adding hidden layers or neurons are impractical here since the final subnetwork architecture of mapping x to $f_a(x;W)$ cannot be altered. This work introduces the concept of Mixture-of-Experts (MoE) (Fedus et al., 2022) to enhance the capacity of g. Specifically, we dynamically generate weights W_a for a specific architecture a by routing to certain weight matrices from a set of expert weights. We term this architecture-routed MoE-based supernet as *Mixture-of-Supernets (MoS)* and design two routing mechanisms for function $g(x, a; E)$. Due to lack of space, the detailed algorithm for supernet training and search is shown in A.2.

3.2 Layer-wise MoS

Assume there are m (number of experts) unique weight matrices ($\{E^i \in \mathcal{R}^{n_{out_{big}} \times n_{in_{big}}}\}_{i=1}^{m}$, or expert weights), which are learnable parameters. For

simplicity, we only use a single linear layer as the example. For an architecture a with n_{out_a} output features, we propose the layer-wise MoS that computes the weights specific to the architecture a (i.e. $W_a \in \mathcal{R}^{n_{out_a} \times n_{in}}$) by performing a weighted combination of expert weights, $W_a = \sum_i \alpha_a^i E_a^i$. Here, $E_a^i \in \mathcal{R}^{n_{out_a} \times n_{in}}$ corresponds to the standard top rows extraction from the ith expert weights. The alignment vector ($\alpha_a \in [0,1]^m, \sum_i \alpha_a^i = 1$) captures the alignment scores of the architecture a with respect to each expert (weights matrix). We encode the architecture a as a numeric vector $\text{Enc}(a) \in \mathcal{R}^{n_{enc} \times 1}$ (e.g., a list of the number of output features for different layers), and apply a learnable router $r(\cdot)$ (an MLP with softmax) to produce such scores, i.e. $\alpha_a = r(\text{Enc}(a))$. Thus, the generalized model function for the linear layer (as shown in Figure 1 (b)) can be defined as (omitting bias for brevity):

$$g(x, a; E) = W_a x = \sum_i r(\text{Enc}(a))^i E_a^i x. \quad (3)$$

The router $r(\cdot)$ governs the degree of weight sharing between two architectures through modulation of alignment scores (α_a). For instance, if $m = 2$ and a is a subnetwork of architecture b, the supernet can allocate weights specific to the smaller architecture a by setting $\alpha_a = (1, 0)$ and $\alpha_b = (0, 1)$. In this scenario, $g(x, a; E)$ exclusively utilizes weights from E_1, and $g(x, b; E)$ uses weights from E_2, enabling updates to E_1 and E_2 towards the loss from architectures a and b without conflicts. It's worth noting that few-shot NAS (Zhao et al., 2021a) is a special case of our framework when the router r is rule-based. Moreover, $g(\cdot)$ functions as an MoE, enhancing expressive power and reducing the objective equation 2. Once supernet training is done, for a given architecture a, the score $\alpha_a = r(\text{Enc}(a))$ can be generated offline. Expert weights collapse, reducing the number of parameters for architecture a to $n_{out_a} \times n_{in_a}$. Layer-wise MoS results in a lower degree of weight sharing between differently sized architectures, as evidenced by a higher Jensen-Shannon distance between their alignment probability vectors compared to similarly sized architectures. Refer to A.1.1 for more details.

3.3 Neuron-wise MoS

Layer-wise MoS employs a standard MoE setup, where each expert is a linear layer/module. The router determines the combination of experts to use for forwarding the input x based on a. In this setup, the degree of freedom for weight generation is m, and the parameter count grows by $m \times |W|$, with $|W|$ being the parameters in the standard supernet. Therefore, a sufficiently large m is needed for flexibility in subnetwork weight generation, but it also introduces too many parameters into the supernet, making layer-wise MoS challenging to train. To address this, we opt for a smaller granularity of weights to represent each expert, using neurons in DNN as experts. In terms of the weight matrix, neuron-wise MoS represents an individual expert with one row, whereas layer-wise MoS uses an entire weight matrix. For neuron-wise MoS, the router output $\beta_a = r(\cdot) \in [0,1]^{n_{out_{big}} \times m}$ for each layer, and the sum of each row in β_a is 1. Similar to layer-wise MoS, we use an MLP to produce the $n_{out_{big}} \times m$ matrix and apply softmax on each row. We formulate the function $g(x, a; E)$ for neuron-wise MoS as

$$W_a = \sum_i \text{diag}(\beta_a^i) E_a^i, \quad (4)$$

where $\text{diag}(\beta)$ constructs a $n_{out_{big}} \times n_{out_{big}}$ diagonal matrix by putting β on the diagonal, and β_a^i is the i-th column of β_a. E^i is still an $n_{out_{big}} \times n_{in}$ matrix as in layer-wise MoS. Compared to layer-wise MoS, neuron-wise MoS offers increased flexibility ($m \times n_{out_a}$ instead of only m) to manage the degree of weight sharing between different architectures, with the number of parameters remaining proportional to m. Neuron-wise MoS provides finer control over weight sharing between subnetworks. Gradient conflict, computed using cosine similarity between the supernet and smallest subnet gradients following NASVIT (Gong et al., 2021), is lowest for neuron-wise MoS compared to layer-wise MoS and HAT, as shown by the highest cosine similarity (see A.1.2).

3.4 Adding $g(x, a; E)$ to Transformer

MoS is adaptable to a single linear layer, multiple linear layers, and other parameterized layers (e.g., layer-norm). Given that the linear layer dominates the number of parameters, we adopt the approach used in most MoE work (Fedus et al., 2022). We apply MoS to the standard weight-sharing-based Transformer ($f_a(x; W)$) by replacing the two linear layers in every feed-forward network block with $g(x, a; E)$.

Supernet	MNLI	CoLA	MRPC	SST2	QNLI	QQP	RTE	Avg. GLUE (↑)
Standalone	82.61	**59.03**	86.54	91.52	89.47	90.68	71.53	81.63
Supernet (Sandwich)	82.34	57.58	86.54	91.74	88.67	90.39	73.26	81.50 (-0.13)
Layer-wise MoS (ours)	82.40	57.62	87.26	92.08	**89.57**	**90.68**	**77.08**	82.38 (+0.75)
Neuron-wise MoS (ours)	**82.68**	58.71	87.74	**92.16**	89.22	90.49	76.39	**82.48 (+0.85)**

Table 2: GLUE validation performance of different supernets (0 additional pretraining steps) compared to standalone (1x pretraining budget). The BERT architecture ($67M$ parameters) is the top model from the pareto front of Supernet (Sandwich) on SuperShaper's search space. Improvement (%) in GLUE average over standalone is enclosed in parentheses in the last column. Layer-wise and neuron-wise MoS perform significantly better than standalone.

Supernet	#Params	#Steps	CoLA	MRPC	SST2	QNLI	QQP	RTE	Avg. GLUE
NAS-BERT	5M	125K	19.8	79.6	**87.3**	84.9	85.8	66.7	70.7
AutoDistil (proxy)	6.88M	0	24.8	78.5	85.9	**86.4**	**89.1**	64.3	71.5
Neuron-wise MoS	5M	0	**28.3**	**82.7**	86.9	84.1	88.5	**68.1**	**73.1**
NAS-BERT	10M	125K	34.0	79.1	**88.6**	**86.3**	88.5	**66.7**	73.9
Neuron-wise MoS	10M	0	**34.7**	**81.0**	88.1	85.1	**89.1**	**66.7**	**74.1**
AutoDistil (proxy)	26.1M	0	48.3	**88.3**	90.1	**90.0**	90.6	69.4	79.5
AutoDistil (agnostic)	26.8M	0	47.1	87.3	**90.6**	89.9	**90.8**	69.0	79.1
Neuron-wise MoS	26.8M	0	**52.7**	88.0	90.0	87.7	89.9	**78.1**	**81.1**
NAS-BERT	30M	125K	48.7	84.6	90.5	**88.4**	**90.2**	71.8	79.0
Neuron-wise MoS	30M	0	**51.0**	**87.3**	**91.1**	87.9	**90.2**	**72.2**	**80.0**
AutoDistil (proxy)	50.1M	0	**55.0**	**88.8**	91.1	**90.8**	**91.1**	71.9	81.4
Neuron-wise MoS	50M	0	**55.0**	88.0	**91.9**	89.0	90.6	**75.4**	**81.6**

Table 3: Comparison of neuron-wise MoS with NAS-BERT and AutoDistil for different model sizes ($\leq 50M$ parameters) based on GLUE validation performance. Neuron-wise MoS use a search space of 550 architectures, which is on par with AutoDistil. The third column corresponds to the number of additional training steps required to obtain the weights for the final architecture after supernet training. Performance numbers for the baseline models are taken from the corresponding papers. See A.4.3 for the hyperparameters of the best architectures. On average GLUE, neuron-wise MoS can perform similarly or improves over NAS-BERT for different model sizes without any additional training. Neuron-wise MoS can improve over AutoDistil for most model sizes in average GLUE.

4 Experiments - Efficient BERT

4.1 Experiment Setup

We explore the application of our proposed supernet in constructing efficient task-agnostic BERT (Devlin et al., 2019) models, focusing on the BERT pretraining task. This involves pretraining a language model from scratch to learn task-agnostic text representations using a masked language modeling objective. The pretrained BERT model is then finetuned on various downstream NLP tasks. Emphasis is on building highly accurate yet small BERT models (e.g., $5M - 50M$ parameters). Both BERT supernet and standalone models are pretrained from scratch on Wikipedia and Books Corpus (Zhu et al., 2015). Performance evaluation is conducted by finetuning on seven tasks from the GLUE benchmark (Wang et al., 2018), chosen by AutoDistil (Xu et al., 2022a). The architecture encoding, data preprocessing, pretraining settings, and finetuning settings are detailed in A.4.1. Baseline models include standalone and standard supernet models proposed in SuperShaper (Ganesan et al., 2021). Our proposed models are layer-wise and neuron-wise MoS. All supernets undergo sandwich training [2]. Parameters m and router's hidden dimension are set to 2 and 128, respectively, for MoS supernets.

4.2 Supernet vs. standalone gap

For investigating the supernet vs. standalone gap, the search space is derived from SuperShaper (Ganesan et al., 2021), encompassing BERT architectures differing only in hidden size at each layer (*120, 240, 360, 480, 540, 600, 768*) with fixed numbers of layers (12) and attention heads (12). This search space includes around 14 billion architectures. We examine the supernet vs. standalone model gap for the top model architecture from the pareto front of Supernet (Sandwich) (Ganesan et al., 2021). Table 2 illustrates the GLUE benchmark performance of standalone training for the architecture (1x pretraining budget, equivalent to 2048

[2] SuperShaper notes that SPOS performs poorly compared to sandwich training; hence, we do not study SPOS for building BERT models. The learning curve is shown in A.4.2.

batch size * 125,000 steps) as well as architecture-specific weights from different supernets (0 additional pretraining steps; i.e., only supernet pretraining). MoS (layer-wise or neuron-wise) bridges the gap between task-specific supernet and standalone performance for 6 out of 7 tasks, including MNLI, a widely used task for evaluating pretrained language models (Liu et al., 2019; Xu et al., 2022b). The average GLUE gap between the standalone model and standard supernet is 0.13 points. Remarkably, with customization and expressivity properties, layer-wise and neuron-wise MoS significantly improve standalone training by 0.75 and 0.85 average GLUE points, respectively [3].

4.3 Comparison with SoTA NAS

The SoTA NAS frameworks for constructing a task-agnostic BERT model are NAS-BERT (Xu et al., 2021) and AutoDistil (Xu et al., 2022a).[4] The NAS-BERT pipeline comprises: (1) supernet training (with a Transformer stack containing multi-head attention, feed-forward network [FFN], and convolutional layers at arbitrary positions), (2) search based on the distillation (task-agnostic) loss, and (3) pretraining the best architecture from scratch (1x pretraining budget, equivalent to 2048 batch size * 125,000 steps). The third step needs to be repeated for every constraint change and hardware change, incurring substantial costs. The AutoDistil pipeline involves: (1) constructing K search spaces and training supernets for each search space independently, (2a) *agnostic*-search mode: searching based on the self-attention distillation (task-agnostic) loss, (2b) *proxy*-search mode: searching based on the MNLI validation score, and (3) extracting architecture-specific weights from the supernet without additional training. The first step can be costly as pretraining K supernets requires K times training compute and memory, compared to training a single supernet. The proxy-search mode may favor AutoDistil unfairly, as it finetunes all architectures in its search space on MNLI and utilizes the MNLI score for ranking. For a fair comparison with SoTA, MNLI task is excluded from evaluation. [5]

Our proposed NAS pipeline addresses challenges in NAS-BERT and AutoDistil. In comparison to SoTA NAS, our search space includes BERT architectures with uniform Transformer layers: hidden size (120 to 768 in increments of 12), attention heads (*6, 12*), intermediate FFN hidden dimension ratio (*2, 2.5, 3, 3.5, 4*). This search space comprises 550 architectures, similar to AutoDistil. The supernet is based on neuron-wise MoS, and the search uses perplexity (task-agnostic) to rank architectures. Unlike NAS-BERT, our final architecture weights are directly extracted from the supernet without additional pretraining. Unlike AutoDistil, our pipeline pretrains only one supernet, significantly reducing training compute and memory. We use only task-agnostic metrics for search, similar to AutoDistil's agnostic setting. Table 3 compares neuron-wise MoS supernet with NAS-BERT and AutoDistil for various model sizes. NAS-BERT and AutoDistil performances are obtained from respective papers. On average GLUE, our pipeline improves over NAS-BERT for $5M$, $10M$, and $30M$ model sizes, with no additional training (100% additional training compute savings, equivalent to 2048 batch size * 125,000 steps). On average GLUE, our pipeline: (i) surpasses AutoDistil-proxy for $6.88M$ and $50M$ model sizes with $1.88M$ and $0.1M$ fewer parameters respectively, and (ii) outperforms both AutoDistil-proxy and AutoDistil-agnostic for $26M$ model size. Besides achieving SoTA results, our method significantly reduces the heavy workload of training multiple models in subnetwork retraining (NAS-BERT) or supernet training (AutoDistil).

5 Experiments - Efficient MT

5.1 Experiment setup

We discuss the application of proposed supernets for building efficient MT models following the setup by Hardware-aware Transformers (HAT (Wang et al., 2020a)), the SoTA NAS framework for MT models with good latency-BLEU tradeoffs. Focusing on WMT'14 En-De, WMT'14 En-Fr, and WMT'19 En-De benchmarks, we maintain consistent architecture encoding and training settings for supernet and standalone models (details in A.5.2). Baseline supernets include HAT and Supernet (Sandwich). Proposed supernets are Layer-wise MoS and Neuron-wise MoS, both us-

[3] Consistency of these results across different seeds is discussed in A.4.5.

[4] AutoDistil (proxy) outperforms SoTA distillation approaches such as TinyBERT (Jiao et al., 2020) and MINILM (Wang et al., 2020b) by 0.7 average GLUE points. Hence, we do not compare against these works.

[5] Refer to A.4.4 for a comparison of neuron-wise MoS against baselines that don't directly tune on the MNLI task. Neuron-wise MoS consistently outperforms baselines in terms of both average GLUE and MNLI task performance.

Dataset	WMT'14 En-De		WMT'14 En-Fr		WMT'19 En-De	
Supernet	MAE (↓)	Kendall (↑)	MAE (↓)	Kendall (↑)	MAE (↓)	Kendall (↑)
HAT	1.84	0.81	1.37	0.63	2.07	0.71
Supernet (Sandwich)	1.62 (12%)	0.81	1.37 (0%)	0.63	2.02 (2.4%)	0.87
Layer-wise MoS (ours)	1.61 (12.5%)	0.54	1.24 (9.5%)	0.73	1.57 (24.2%)	0.87
Neuron-wise MoS (ours)	**1.13 (38.6%)**	0.71	**1.2 (12.4%)**	0.85	**1.48 (28.5%)**	0.81

Table 4: Mean absolute error (MAE) and Kendall rank correlation coefficient between the supernet and the standalone model BLEU performance for 15 random architectures from the MT search space. Improvements (%) in mean absolute error over HAT are in parentheses. Our supernets enjoy minimal MAE and comparable ranking quality with respect to the baseline models.

ing sandwich training, with m and router's hidden dimension set to 2 and 128, respectively. Refer to A.5.8 for the rationale behind choosing 'm'.

5.2 Supernet vs. standalone gap

In HAT's search space of $6M$ encoder-decoder architectures, featuring flexible parameters like embedding size (512 or 640), decoder layers (1 to 6), self/cross attention heads (4 or 8), and number of top encoder layers for decoder attention (1 to 3), good supernets should exhibit minimal mean absolute error (MAE) and high rank correlation between supernet and standalone performance for a given architecture. Table 4 presents MAE and Kendall rank correlation for 15 random architectures, showcasing that sandwich training yields better MAE and rank quality compared to HAT. While our proposed supernets achieve comparable rank quality for WMT'14 En-Fr and WMT'19 En-De, and slightly underperform for WMT'14 En-De, they exhibit minimal MAE across all tasks. Particularly, neuron-wise MoS achieves substantial MAE improvements, suggesting lower additional training steps needed to make MAE negligible (as detailed in Section 5.4). Supernet and standalone performance plots reveal neuron-wise MoS excelling for almost all top-performing architectures (see A.5.3). The training overhead for MoS is generally negligible, e.g., for WMT'14 En-De, supernet training takes 248 hours, with neuron-wise MoS and layer-wise MoS requiring 14 and 18 additional hours, respectively (less than 8% overhead, see Section A.5.10 for details).

5.3 Comparison with the SoTA NAS

The pareto front from the supernet is obtained using an evolutionary search algorithm that leverages the supernet for quickly identifying top-performing candidate architectures and a latency estimator for promptly discarding candidates with latencies surpassing a threshold. Settings for the evolutionary search algorithm and latency estimator are detailed in A.5.4. Three latency thresholds are explored: 100 ms, 150 ms, and 200 ms. Table 5 illustrates the latency vs. supernet performance tradeoff for models in the pareto front from different supernets. Compared to HAT, the proposed supernets consistently achieve significantly higher BLEU for each latency threshold across all datasets, emphasizing the importance of architecture specialization and expressiveness of the supernet. See A.5.6 for the consistency of these trends across different seeds.

5.4 Additional training to close the gap

The proposed supernets significantly minimize the gap between the supernet and standalone MAE (as discussed in Section 5.2), yet the gap remains non-negligible. Closing the gap for an architecture involves extracting architecture-specific weights from the supernet and conducting additional training until the standalone performance is reached (achieving a gap of 0). An effective supernet should demand a minimal number of additional steps and time for the extracted architectures to close the gap. In the context of additional training, we evaluate the test BLEU for each architecture after every $10K$ steps, stopping when the test BLEU matches or exceeds the test BLEU of the standalone model. Table 6 presents the average number of additional training steps required for all models on the pareto front from each supernet to close the gap. Compared to HAT, layer-wise MoS achieves an impressive reduction of 9% to 51% in training steps, while neuron-wise MoS delivers the most substantial reduction of 21% to 60%. For the WMT'14 En-Fr task, both MoS supernets require at least 2.7% more time than HAT to achieve SoTA BLEU across different constraints. These results underscore the importance of architecture specialization and supernet expressivity in significantly improving the training efficiency of subnets extracted from the supernet.

Dataset	WMT'14 En-De			WMT'14 En-Fr			WMT'19 En-De		
Supernet / Latency Constraint	100 ms	150 ms	200 ms	100 ms	150 ms	200 ms	100 ms	150 ms	200 ms
HAT	25.26	26.25	26.28	38.94	39.26	39.16	42.61	43.07	43.23
Layer-wise MoS (ours)	26.28	27.31	**28.03**	39.34	**40.29**	**41.24**	43.45	**44.71**	46.18
Neuron-wise MoS (ours)	**26.37**	**27.59**	27.79	**39.55**	40.02	41.04	**43.77**	44.66	**46.21**

Table 5: Latency vs. Supernet BLEU for the models on the pareto front, obtained by performing search with different latency constraints (100 ms, 150 ms, 200 ms) on the NVIDIA V100 GPU. Our supernets yield architectures that enjoy better latency-BLEU tradeoffs than HAT.

Dataset	Additional training steps (↓)			Additional training time (NVIDIA V100 hours) (↓)		
Supernet	WMT'14 En-De	WMT'14 En-Fr	WMT'19 En-De	WMT'14 En-De	WMT'14 En-Fr	WMT'19 En-De
HAT	33K	33K	26K	63.9	**60.1**	52.3
Laye. MoS	16K (51.5%)	30K (9%)	20K (23%)	35.5 (44.4%)	66.5 (-10.6%)	45.2 (13.5%)
Neur. MoS	**13K (60%)**	**26K (21%)**	**16K (38.4%)**	**31.0 (51.4%)**	61.7 (-2.7%)	**39.5 (24.5%)**

Table 6: Average number of additional training steps and time required for the models on the pareto front to close the supernet vs. standalone gap. Improvements (%) over HAT are shown in parentheses. Our supernets require minimal number of additional training steps and time to close the gap compared to HAT for most tasks. See A.5.5 for each latency constraint.

6 Related Work

In this section, we briefly review existing research on NAS in NLP. Evolved Transformer (ET) (So et al., 2019) is an initial work that explores NAS for efficient MT models. It employs evolutionary search and dynamically allocating training resources for promising candidates., HAT (Wang et al., 2020a) introduces a weight-sharing supernet as a performance estimator, amortizing the training cost for candidate MT evaluations in evolutionary search. NAS-BERT (Xu et al., 2021) partitions the BERT-Base model into blocks and trains a weight-sharing supernet to distill each block. NAS-BERT uses progressive shrinking during supernet training to prune less promising candidates, identifying top architectures for each efficiency constraint quickly. However, NAS-BERT requires pretraining the top architecture from scratch for every constraint change, incurring high costs. Super-Shaper (Ganesan et al., 2021) pretrains a weight-sharing supernet for BERT using a masked language modeling objective with sandwich training. AutoDistil (Xu et al., 2022a) employs few-shot NAS (Zhao et al., 2021c): constructing K search spaces of non-overlapping BERT architectures and training a weight-sharing BERT supernet for each search space. The search is based on self-attention distillation loss with BERT-Base (task-agnostic search) and MNLI score (proxy search). AutoMoE (Jawahar et al., 2023) augments the search space of HAT with mixture-of-expert models to design efficient translation models. Refer to A.3 for the main differences between our framework and the AutoMoE framework.

In the computer vision community, K-shot NAS (Su et al., 2021) generates weights for each subnet as a convex combination of different supernet weights in a dictionary using a simplex code. While K-shot NAS shares similarities with layer-wise MoS, there are key distinctions. K-shot NAS trains the architecture code generator and supernet iteratively due to training difficulty, whereas layer-wise MoS trains all its components jointly. K-shot NAS has been applied specifically to convolutional architectures for image classification tasks. However, it introduces too many parameters with an increase in the number of supernets (K), a concern alleviated by neuron-wise MoS due to its granular weight specialization. In our work, we focus on NLP tasks and relevant baselines, noting that supernets in NLP tend to lag significantly behind standalone models in terms of performance. Additionally, the authors of K-shot NAS have not released the code to reproduce their results, preventing a direct evaluation against their method.

7 Conclusion

We introduced Mixture-of-Supernets, a formulation aimed at enhancing the expressive power of supernets. By adopting the idea of MoE, we demonstrated the ability to generate flexible weights for subnetworks. Through extensive evaluations for constructing efficient BERT and MT models, our supernets showcased the capacity to: (i) minimize retraining time, thereby significantly improving NAS efficiency, and (ii) produce high-quality architectures that meet user-defined constraints.

8 Limitations

The limitations of this work are as follows:

1. Applying Mixture-of-Supernet (MoS) to popular benchmarks in NLP, focusing on efficient machine translation and BERT, offers valuable insights. A potential impactful future direction could involve extending the application of MoS to build efficient autoregressive decoder-only language models, such as GPT-4 (OpenAI, 2023).

2. Introducing MoE architecture potentially need more training budget. In our work, we do not use large number of training iteration for fair comparison and fixing the number of expert weights (m) to 2 works well. We will investigate the full potential of the proposed supernets by combining larger training budget (e.g., $\geq 200K$ steps) and larger number of expert weights (e.g., ≥ 16 expert weights) in the future work.

9 Acknowledgments

We used ChatGPT for rephrasing and grammar checking of the paper.

References

Gabriel Bender, Pieter-Jan Kindermans, Barret Zoph, Vijay Vasudevan, and Quoc Le. 2018. Understanding and simplifying one-shot architecture search. In *International conference on machine learning*, pages 550–559. PMLR.

Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. 2020. Once for all: Train one network and specialize it for efficient deployment. In *International Conference on Learning Representations*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

William Fedus, Barret Zoph, and Noam Shazeer. 2022. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research*, 23(120):1–39.

Vinod Ganesan, Gowtham Ramesh, and Pratyush Kumar. 2021. Supershaper: Task-agnostic super pretraining of BERT models with variable hidden dimensions. *CoRR*, abs/2110.04711.

Chengyue Gong, Dilin Wang, Meng Li, Xinlei Chen, Zhicheng Yan, Yuandong Tian, Vikas Chandra, et al. 2021. Nasvit: Neural architecture search for efficient vision transformers with gradient conflict aware supernet training. In *International Conference on Learning Representations*.

Zichao Guo, Xiangyu Zhang, Haoyuan Mu, Wen Heng, Zechun Liu, Yichen Wei, and Jian Sun. 2020. Single path one-shot neural architecture search with uniform sampling. In *Computer Vision – ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XVI*, page 544–560, Berlin, Heidelberg. Springer-Verlag.

Peter Izsak, Moshe Berchansky, and Omer Levy. 2021. How to train BERT with an academic budget. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 10644–10652, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Ganesh Jawahar, Subhabrata Mukherjee, Xiaodong Liu, Young Jin Kim, Muhammad Abdul-Mageed, Laks Lakshmanan, V.S., Ahmed Hassan Awadallah, Sebastien Bubeck, and Jianfeng Gao. 2023. Auto-MoE: Heterogeneous mixture-of-experts with adaptive computation for efficient neural machine translation. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 9116–9132, Toronto, Canada. Association for Computational Linguistics.

Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu. 2020. TinyBERT: Distilling BERT for natural language understanding. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 4163–4174, Online. Association for Computational Linguistics.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. Cite arxiv:1907.11692.

OpenAI. 2023. Gpt-4 technical report.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 311–318, Philadelphia, Pennsylvania, USA. Association for Computational Linguistics.

Hieu Pham, Melody Guan, Barret Zoph, Quoc Le, and Jeff Dean. 2018. Efficient neural architecture search via parameters sharing. In *International conference on machine learning*, pages 4095–4104. PMLR.

Matt Post. 2018. A call for clarity in reporting BLEU scores. In *Proceedings of the Third Conference on Machine Translation: Research Papers*, pages 186–191, Brussels, Belgium. Association for Computational Linguistics.

David So, Quoc Le, and Chen Liang. 2019. The evolved transformer. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 5877–5886. PMLR.

Xiu Su, Shan You, Mingkai Zheng, Fei Wang, Chen Qian, Changshui Zhang, and Chang Xu. 2021. K-shot nas: Learnable weight-sharing for nas with k-shot supernets. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 9880–9890. PMLR.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. 2018. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 353–355, Brussels, Belgium. Association for Computational Linguistics.

Hanrui Wang, Zhanghao Wu, Zhijian Liu, Han Cai, Ligeng Zhu, Chuang Gan, and Song Han. 2020a. HAT: Hardware-aware transformers for efficient natural language processing. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7675–7688, Online. Association for Computational Linguistics.

Wenhui Wang, Furu Wei, Li Dong, Hangbo Bao, Nan Yang, and Ming Zhou. 2020b. Minilm: Deep self-attention distillation for task-agnostic compression of pre-trained transformers. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS'20.

Dongkuan Xu, Subhabrata Mukherjee, Xiaodong Liu, Debadeepta Dey, Wenhui Wang, Xiang Zhang, Ahmed Hassan Awadallah, and Jianfeng Gao. 2022a. Few-shot task-agnostic neural architecture search for distilling large language models. In *Advances in Neural Information Processing Systems*.

Jin Xu, Xu Tan, Renqian Luo, Kaitao Song, Jian Li, Tao Qin, and Tie-Yan Liu. 2021. Nas-bert: Task-agnostic and adaptive-size bert compression with neural architecture search. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, KDD '21, page 1933–1943, New York, NY, USA. Association for Computing Machinery.

Jin Xu, Xu Tan, Kaitao Song, Renqian Luo, Yichong Leng, Tao Qin, Tie-Yan Liu, and Jian Li. 2022b. Analyzing and mitigating interference in neural architecture search. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 24646–24662. PMLR.

Jiahui Yu, Pengchong Jin, Hanxiao Liu, Gabriel Bender, Pieter-Jan Kindermans, Mingxing Tan, Thomas Huang, Xiaodan Song, Ruoming Pang, and Quoc Le. 2020. Bignas: Scaling up neural architecture search with big single-stage models. In *Computer Vision – ECCV 2020*, pages 702–717, Cham. Springer International Publishing.

Yiyang Zhao, Linnan Wang, Yuandong Tian, Rodrigo Fonseca, and Tian Guo. 2021a. Few-shot neural architecture search. In *International Conference on Machine Learning*, pages 12707–12718. PMLR.

Yiyang Zhao, Linnan Wang, Yuandong Tian, Rodrigo Fonseca, and Tian Guo. 2021b. Few-shot neural architecture search. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 12707–12718. PMLR.

Yiyang Zhao, Linnan Wang, Yuandong Tian, Rodrigo Fonseca, and Tian Guo. 2021c. Few-shot neural architecture search. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 12707–12718. PMLR.

Yukun Zhu, Ryan Kiros, Rich Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. 2015. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In *The IEEE International Conference on Computer Vision (ICCV)*.

Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. 2018. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8697–8710.

A Appendix

A.1 Weight Sharing and Gradient Conflict Analysis

A.1.1 Jensen-Shannon distance of alignment vector as a weight sharing measure

We use the Jensen-Shannon distance of alignment vector generated by Layer-wise MoS for two architectures as a proxy to quantify the degree of weight sharing. Ideally, the lower the Jensen-Shannon distance, the higher the degree of weight sharing and vice-versa. We experiment with two architectures of 23M parameters (Smallest A and Smallest B) and two architectures of 118M parameters (Largest A and Largest B). From Table 7, it is clear that Layer-wise MoS induces low degree of weight sharing between differently sized architectures shown by higher Jensen-Shannon distance between their alignment vectors. On the other hand, there is a high degree of weight sharing between similarly sized architectures where Jensen-Shannon distance is significantly low.

Model 1	Model 2	WMT'14 En-De	WMT'14 En-Fr	WMT'19 En-De
Smallest A ($23M$)	Largest A ($118M$)	0.297	0.275	0.263
Smallest B ($23M$)	Largest B ($118M$)	0.281	0.258	0.245
Smallest A ($23M$)	Largest B ($118M$)	0.284	0.263	0.249
Smallest B ($23M$)	Largest A ($118M$)	0.294	0.27	0.259
Smallest A ($23M$)	Smallest B ($118M$)	0.006	0.008	0.004
Largest A ($23M$)	Largest B ($118M$)	0.014	0.012	0.015

Table 7: Jensen-Shannon distance of Layer-wise MoS alignment vector across models as a weight sharing measure. Layer-wise MoS induces low degree of weight sharing between differently sized architectures shown by higher Jensen-Shannon distance between their alignment vectors compared to that of similarly sized architectures. Note that architectures A and B differ by number of encoder/decoder attention heads.

A.1.2 Cosine similarity between the supernet gradient and the smallest subnet gradient as a gradient conflict measure.

We compute gradient conflict using cosine similarity between the supernet gradient and the smallest subnet gradient, following NASVIT work (Gong et al., 2021). In Table 8, we show that Neuron-wise MoS enjoys lowest gradient conflict compared to Layer-wise MoS and HAT, shown by highest cosine similarity.

A.2 Detailed algorithm for Supernet training and Search

A.2.1 Supernet training algorithm

The detailed algorithm for supernet training is shown in Algorithm 1.

A.2.2 Search algorithm

The detailed algorithm for search is shown in Algorithm 2.

A.3 Comparison to the AutoMoE work

Goals: Given a search space of dense and mixture-of-expert models, the goal of the AutoMoE framework (Jawahar et al., 2023) is to search for high-performing model architectures that satisfy user-defined efficiency constraints. The final architectures can be dense or mixture-of-expert models. On the other hand, given a search space of dense models only, the goal of the Mixture-of-Supernets framework is to search for high-performing dense model architectures that satisfy user-defined efficiency constraints. The final architecture can be a dense model only. In addition, the MoS framework minimizes the retraining compute required for the searched architecture to approach the standalone performance. The MoS framework designs the supernet with flexible weight sharing and high capacity. On the other hand, the supernet underlying the AutoMoE framework suffers from strict weight sharing and limited capacity.

Applications of mixture-of-experts: The main application of mixture-of-experts idea by the AutoMoE framework is to augment the standard NAS search space of dense models with mixture-of-experts models. To this end, the AutoMoE framework modifies the standard weight sharing supernet to support weight generation for mixture-of-expert models. On the other hand, the Mixture-of-Supernets framework uses the mixture-of-expert design to: (i) increase the capacity of standard weight sharing supernet and (ii) customize weights for each architecture. Post training, the expert weights are collapsed to create a single weight for the dense architecture.

Router specifications: The router underlying the AutoMoE framework takes token embedding as input, outputs a probability distribution over experts, and passes token embedding to top-k experts. On the other hand, the router underlying the Mixture-of-Supernets framework takes architecture embedding as input, outputs a probability distribution over experts (layer-wise MoS) / neurons (neuron-wise MoS), uses the probability distribution to combine ALL the expert weights into a single weight, and passes token embedding to the single weight (all experts).

A.4 Additional Experiments - Efficient BERT

A.4.1 BERT pretraining / finetuning settings

Pretraining data: The pretraining data consists of text from Wikipedia and Books Corpus (Zhu et al., 2015). We use the data preprocessing scripts provided by Izsak et al. to construct the tokenized text.

Supernet and standalone pretraining settings: The pretraining settings for supernet and standalone models are taken from SuperShaper (Ganesan et al., 2021): batch size of 2048, maximum sequence length of 128, training steps of 125K, learning rate of $5e^{-4}$, weight decay of 0.01, and warmup steps

Supernet	WMT'14 En-De	WMT'19 En-De
HAT	0.522	0.416
Layer-wise MoS	0.515	0.517
Neuron-wise MoS	0.555	0.52

Table 8: Gradient conflict via cosine similarity between the supernet gradient and the smallest subnet gradient. Neuron-wise MoS enjoys lower gradient conflict, shown via. high cosine similarity.

Algorithm 1 Training algorithm for Mixture-of-Supernets used in MT.

Input: `Training data:` $\mathcal{X}_{tr}$, `Search space:` $\mathcal{A}$, `No. of training steps: num-train-steps`, `Type of MoS: mos-type`

Output: `Training Supernet Weights:` $\mathbb{E}$

$\mathbb{E} \leftarrow$ Random weights from Normal Distribution.
for $iter \leftarrow 1$ `to num-train-steps` **do**
    // sample data
    $x, y \sim \mathcal{X}_{tr}$
    // perform sandwich sampling
    for a in $[a_{rand} \sim \mathcal{A}, a_{big}, a_{small}]$ **do**
        Enc(a) // create the architecture encoding
        // generate architecture-specific weights
        if mos-type == Layer wise MoS **then**
            $W_a = \sum_i r(\text{Enc}(a))^i E_a^i$
        else if mos-type == Neuron wise MoS **then**
            $W_a = \sum_i \text{diag}(\beta_a^i) E_a^i$
        // compute task-specific loss
        loss $\leftarrow \mathcal{L}(W_a x, y)$
        loss.backward() // compute gradients
    Update $\mathbb{E}$ using accumulated gradients // learning rule
return $\mathbb{E}$

Algorithm 2 Evolutionary search algorithm for Neural architecture search used in MT.

Input: `supernet`, `latency-predictor`, `num-iterations`, `num-population`, `num-parents`, `num-mutations`, `num-crossover`, `mutate-prob`, `latency-constraint`

Output: `best-architecture`

```
// create initial population
popu ← num-population random samples from the search space
for iter ← 1 to num-iterations do
    // generate parents by picking top candidates
    cur-parents ← top 'num-parents' architectures from popu by MoS validation loss
    // generate candidates via mutation
    cur-mutate-popu = {}
    for mi ← 1 to num-mutations do
        cur-mutate-gene ← mutate a random example from popu with mutation probability mutate-prob
        if cur-mutate-gene satisfies latency-constraint via latency-predictor then
            cur-mutate-popu = cur-mutate-popu ∪ cur-mutate-gene
    // generate candidates via cross-over
    cur-crossover-popu = {}
    for ci ← 1 to num-crossover do
        cur-crossover-gene ← crossover two random examples from popu
        if cur-crossover-gene satisfies latency-constraint via latency-predictor then
            cur-crossover-popu = cur-crossover-popu ∪ cur-crossover-gene
    // update population
    popu = cur-parents ∪ cur-mutate-popu ∪ cur-crossover-popu
return top architecture from popu by MoS's validation loss
```

of $10K$ (0 for standalone). For experiments with the search space from SuperShaper (Ganesan et al., 2021) (Section 4.2), the architecture encoding a is a list of hidden size at each layer of the architecture (12 elements since the supernet is a 12 layer model). For experiments with the search space on par with AutoDistil (Xu et al., 2022a) (Section 4.3), the architecture encoding a is a list of four elastic hyperparameters of the homogeneous BERT architecture: number of layers, hidden size of all layers, feedforward network (FFN) expansion ratio of all layers and number of attention heads of all layers (see Table 9 for sample homogeneous BERT architectures).

Finetuning settings: We evaluate the performance of the BERT model by finetuning on each of the seven tasks (chosen by AutoDistil (Xu et al., 2022a)) in the GLUE benchmark (Wang et al., 2018). The evaluation metric is the average accuracy (Matthews's correlation coefficient for CoLA only) on all the tasks (GLUE average). The finetuning settings are taken from the BERT paper (Devlin et al., 2019): learning rate from $\{5e^{-5}, 3e^{-5}, 2e^{-5}\}$, batch size from $\{16, 32\}$, and epochs from $\{2, 3, 4\}$.

A.4.2 Learning curve for BERT supernet variants

Figure 2 shows the training steps versus validation MLM loss (learning curve) for the standalone BERT model and different supernet based BERT variants. The standalone model and the supernet are compared for the biggest architecture (big) and the smallest architecture (small) from the search space of SuperShaper (Ganesan et al., 2021). For the biggest architecture, the standalone model performs the best. For the smallest architecture, the standalone model is outperformed by all the supernet variants. In both cases, the proposed supernets (especially neuron-wise MoS) perform much better than the standard supernet.

A.4.3 Architecture comparison of Neuron-wise MoS vs. AutoDistil

Table 9 shows the comparison of the BERT architecture designed by our proposed neuron-wise MoS with AutoDistil.

A.4.4 Fair comparison of Neuron-wise MoS w.r.t SoTA with MNLI

We compare neuron-wise MoS with NAS-BERT and AutoDistil (agnostic) for different model sizes ($\leq 50M$ parameters) based on GLUE validation performance. In Table 10, we include results on MNLI task. For fair comparison, we drop AutoDistil (proxy), which directly uses MNLI task for architecture selection. Neuron-wise MoS improves over the baselines in all model sizes, in terms of average GLUE. For MNLI task, neuron-wise MoS improves over the baselines in most model sizes.

A.4.5 BERT results with different random seeds

Table 11 displays BERT results on CoLA and RTE with various random seeds. Layer-wise MoS consistently enhances performance over baselines in RTE and diminishes performance compared to baselines in CoLA for both seeds. The BERT architecture (67M parameters) corresponds to the top model from the pareto front of Supernet (Sandwich) in SuperShaper's search space (consistent with Table 2).

A.5 Additional Experiments - Efficient Machine Translation

A.5.1 Machine translation benchmark data

Table 12 shows the statistics of three machine translation datasets: WMT'14 En-De, WMT'14 En-Fr, and WMT'19 En-De.

A.5.2 Training settings and metrics

The training settings for both supernet and standalone models are the same: $40K$ training steps, Adam optimizer, a cosine learning rate scheduler, and a warmup of learning rate from 10^{-7} to 10^{-3} with cosine annealing. The best checkpoint is selected based on the validation loss, while the performance of the MT model is evaluated based on BLEU. The beam size is four with length penalty of 0.6. The architecture encoding a is a list of following 10 values:

1. *Encoder embedding dimension* corresponds to embedding dimension of the encoder.

2. *Encoder #layers* corresponds to number of encoder layers.

3. *Average encoder FFN. intermediate dimension* corresponds to average of FFN intermediate dimension across encoder layers.

4. *Average encoder self attention heads* corresponds to average of number of self attention heads across encoder layers.

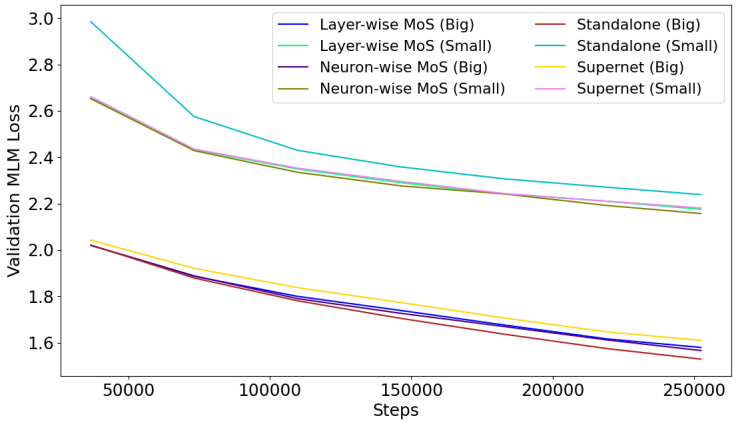


Figure 2: Learning Curve - Training steps vs. Validation MLM loss. 'Big' and 'Small' correspond to the largest and the smallest BERT architecture respectively from the search space of SuperShaper. 'Standalone' and 'Supernet' correspond to training from scratch and sampling from the supernet respectively. All the supernets are trained with sandwich training.

Standalone / Supernet	Model Size	#Layers	#Hidden Size	#FFN Expansion Ratio	#Heads
BERT	109M	12	768	4	12
AutoDistil (proxy)	6.88M	7	160	3.5	10
Neuron-wise MoS	5M	12	120	2.0	6
Neuron-wise MoS	10M	12	180	3.5	6
AutoDistil (agnostic)	26.8M	11	352	4	10
Neuron-wise MoS	26.8M	12	372	2.5	6
Neuron-wise MoS	30M	12	384	3	6
AutoDistil (proxy)	50.1M	12	544	3	9
Neuron-wise MoS	50M	12	504	3.5	12

Table 9: Architecture comparison of the best architecture designed by the neuron-wise MoS with AutoDistil (Xu et al., 2022a) and BERT-Base (Devlin et al., 2019).

Supernet	#Params	#Steps	MNLI	CoLA	MRPC	SST2	QNLI	QQP	RTE	Avg. GLUE
NAS-BERT	5M	125K	74.4	19.8	79.6	**87.3**	**84.9**	85.8	66.7	71.2
Neuron-wise MoS	5M	0	**75.5**	**28.3**	**82.7**	86.9	84.1	**88.5**	**68.1**	**73.4**
NAS-BERT	10M	125K	76.4	34.0	79.1	**88.6**	**86.3**	88.5	**66.7**	74.2
Neuron-wise MoS	10M	0	**77.2**	**34.7**	**81.0**	88.1	85.1	**89.1**	**66.7**	**74.6**
AutoDistil (agnostic)	26.8M	0	**82.8**	47.1	87.3	**90.6**	**89.9**	**90.8**	69.0	79.6
Neuron-wise MoS	26.8M	0	80.7	**52.7**	**88.0**	90.0	87.7	89.9	**78.1**	**81.0**
NAS-BERT	30M	125K	81.0	48.7	84.6	90.5	**88.4**	**90.2**	71.8	79.3
Neuron-wise MoS	30M	0	**81.6**	**51.0**	**87.3**	**91.1**	87.9	**90.2**	**72.2**	**80.2**
Neuron-wise MoS	50M	0	82.4	55.0	88.0	91.9	89.0	90.6	75.4	81.8

Table 10: Comparison of neuron-wise MoS with NAS-BERT and AutoDistil (agnostic) for different model sizes ($\leq 50M$ parameters) based on GLUE validation performance. We include results on MNLI task. For fair comparison, we drop AutoDistil (proxy), which directly uses MNLI task for architecture selection. Neuron-wise MoS improves over the baselines in all model sizes, in terms of average GLUE. For MNLI task, neuron-wise MoS improves over the baselines in most model sizes.

Seeds	Seed 1		Seed 2		
Model	CoLA	RTE	CoLA	RTE	Average
Standalone	**59.03**	71.53	**58.04**	72.22	65.21
Supernet (Sandwich)	57.58	73.26	57.1	72.92	65.22
Layer-wise MoS	57.62	**77.08**	56.3	**76.74**	**66.91**

Table 11: BERT results on CoLA and RTE with different random seeds. Layer-wise MoS improves over baselines in RTE and degrades over baselines in CoLA consistently across both seeds.

5. *Decoder embedding dimension* corresponds to embedding dimension of the decoder.

6. *Decoder #Layers* corresponds to number of decoder layers.

7. *Average Decoder FFN. Intermediate Dimension* corresponds to average of FFN intermediate dimension across decoder layers.

8. *Average decoder self attention heads* corresponds to average of number of self attention heads across decoder layers.

9. *Average decoder cross attention heads* corresponds to average of number of cross attention heads across decoder layers.

10. *Average arbitrary encoder decoder attention* corresponds to average number of encoder layers attended by cross-attention heads in each decoder layer (-1 means only attend to the last layer, 1 means attend to the last two layers, 2 means attend to the last three layers).

A.5.3 Supernet vs. Standalone performance plot

Figure 3 displays the supernet vs. the standalone performance for 15 randomly sampled architectures on all the three tasks. Neuron-wise MoS excel for almost all the top performing architectures (≥ 26.5 and ≥ 42.5 standalone BLEU for WMT'14 En-De and WMT'19 En-De respectively), which indicates that the models especially in the pareto front can benefit immensely from neuron level specialization.

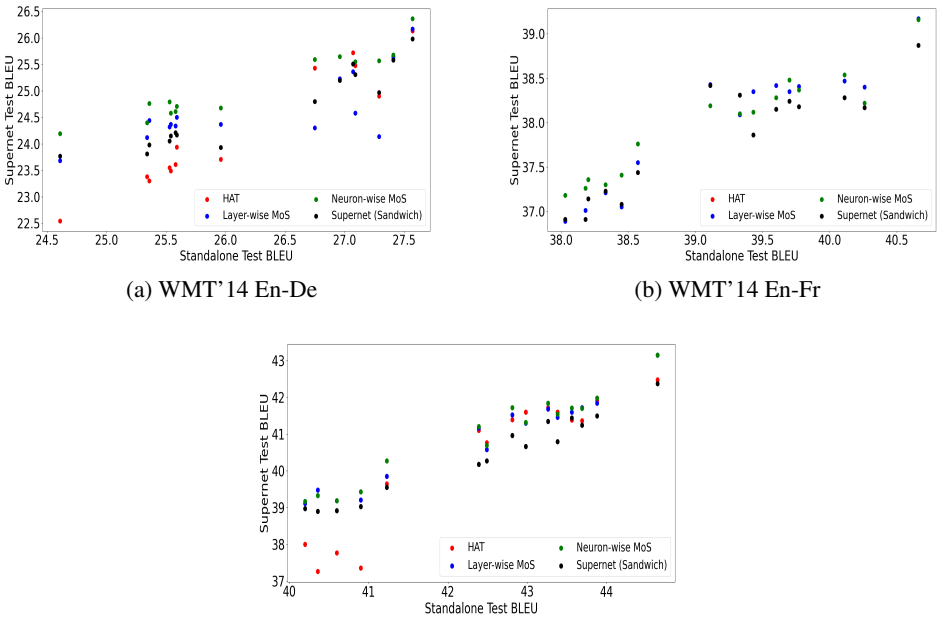
A.5.4 HAT Settings

Evolutionary search: The settings for the evolutionary search algorithm include: 30 iterations, population size of 125, parents population of 25, crossover population of 50, and mutation population of 50 with 0.3 mutation probability.

Latency estimator: The latency estimator is developed in two stages. First, the latency dataset is constructed by measuring the latency of 2000 randomly sampled architectures directly on the user-defined hardware (NVIDIA V100 GPU). Latency is the time taken to translate a source sentence to a target sentence (source and target sentence lengths of 30 tokens each). For each architecture, 300 latency measurements are taken, outliers (top 10% and bottom 10%) are removed, and the rest (80%) is averaged. Second, the latency estimator is a 3 layer multi-layer neural network based regressor, which is trained using encoding and latency of the architecture as features and labels respectively.

Dataset	Year	Source Lang	Target Lang	#Train	#Valid	#Test
WMT	2014	English (en)	German (de)	4.5M	3000	3000
WMT	2014	English (en)	French (fr)	35M	26000	26000
WMT	2019	English (en)	German (de)	43M	2900	2900

Table 12: Machine translation benchmark data.



(a) WMT'14 En-De (b) WMT'14 En-Fr

(c) WMT'19 En-De

Figure 3: Supernet vs. Standalone model performance for 15 random architectures from MT search space. Supernet performance is obtained by evaluating the architecture-specific weights extracted from the supernet. Standalone model performance is obtained by training the architecture from scratch to convergence and evaluating it.

A.5.5 Additional training steps to close the gap vs. performance

Figure 4, Figure 5, and Figure 6 show the additional training steps vs. BLEU for different latency constraints on the WMT'14 En-De task, WMT'14 En-Fr and WMT'19 En-De tasks respectively.

A.5.6 Evolutionary Search - Stability

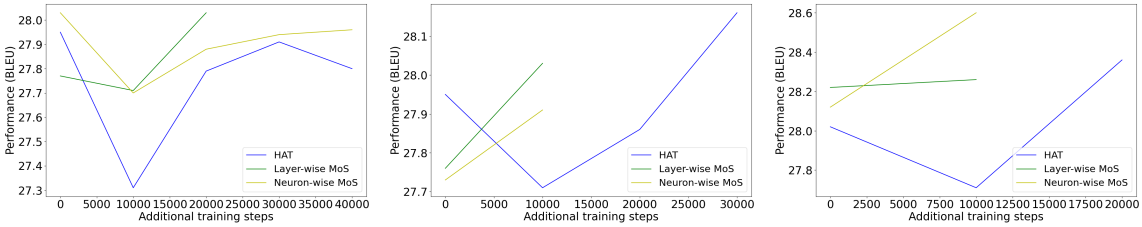
We study the initialization effects on the stability of the pareto front outputted by the evolutionary search for different supernets. Table 13 displays sampled (direct) BLEU and latency of the models in the pareto front for different seeds on the WMT'14 En-Fr task. The differences in the latency and BLEU across seeds are mostly marginal. This result highlights that the pareto front outputted by the evolutionary search is largely stable for all the supernet variants.

A.5.7 Impact of different router function

Table 14 displays the impact of varying the number of hidden layers in the router function for neuron-wise MoS on the WMT'14 En-De task. Two hidden layers provide the right amount of router capacity, while adding more hidden layers results in steady performance drop.

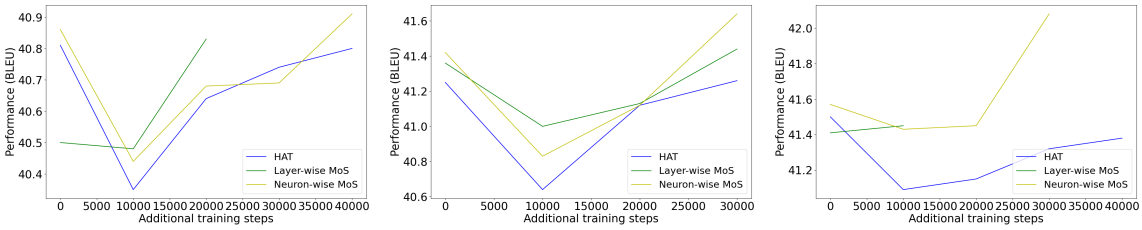
A.5.8 Impact of increasing the number of expert weights 'm'

Table 15 displays the impact of increasing the number of expert weights 'm' for the WMT'14 En-Fr task, where the architecture for all the supernets is the top architecture from the pareto front of HAT for the latency constraint of 200 ms. Under the standard training budget ($40K$ steps for MT), the performance of layer-wise MoS does not seem to improve by increasing 'm' from 2 to 4. Increasing 'm' introduces too many parameters, which might necessitate a significant increase in the training budget (e.g., 2 times more training steps than the standard training budget). For fair comparison with existing literature, we use the standard training budget for all the experiments. We will investigate the full potential of the proposed supernets by combin-



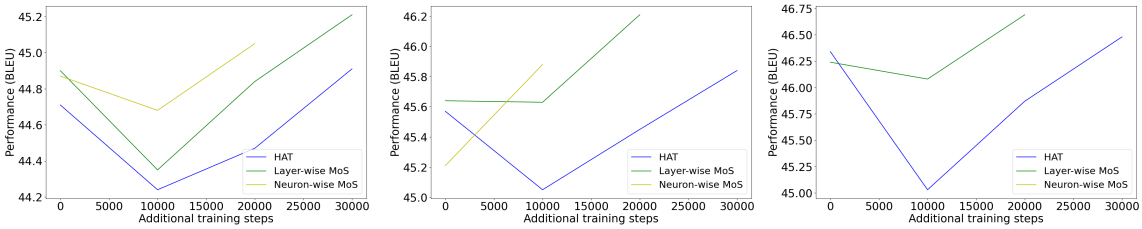
(a) 100ms (b) 150ms (c) 200ms

Figure 4: Additional training steps to close the supernet - standalone gap vs. performance for different latency constraints on the WMT'14 En-De dataset.



(a) 100ms (b) 150ms (c) 200ms

Figure 5: Additional training steps to close the supernet - standalone gap vs. performance for different latency constraints on the WMT'14 En-Fr dataset.



(a) 100 ms (b) 150 ms (c) 200 ms

Figure 6: Additional training steps to close the supernet - the standalone gap vs. performance for different latency constraints on the WMT'19 En-De dataset. For 200 ms latency constraint, neuron-wise MoS closes the gap without additional training.

ing larger training budget (e.g., $\geq 200K$ steps) and larger number of expert weights (e.g., ≥ 16 expert weights) in future work.

A.5.9 SacreBLEU vs. BLEU

We use the standard BLEU (Papineni et al., 2002) to quantify the performance of supernet following HAT for a fair comparison. In Table 16, we also experiment with SacreBLEU (Post, 2018), where the similar trend of MoS yielding better performance for a given latency constraint holds true.

A.5.10 Breakdown of the overall time savings

Table 17 shows the breakdown of the overall time savings of MoS supernets versus HAT for computing pareto front for the WMT'14 En-De task. The latency constraints include 100 ms, 150 ms, 200 ms. MoS have an overall GPU hours savings of at least 20% w.r.t. HAT, thanks to significant savings in additional training time (45%-51%).

A.5.11 Codebase

We share the codebase in the supplementary material, which can be used to reproduce all the results in this paper. For both BERT and machine translation evaluation benchmarks, we add a README file that contains the following instructions: (i) environment setup (e.g., software dependencies), (ii) data download, (iii) supernet training, (iv) search, and (v) subnet retraining.

Supernet / Pareto Front		Model 1		Model 2		Model 3	
	Seed	Latency	BLEU	Latency	BLEU	Latency	BLEU
HAT (SPOS)	1	96.39	38.94	176.44	39.26	187.53	39.16
HAT (SPOS)	2	98.91	38.96	159.87	39.20	192.11	39.09
HAT (SPOS)	3	100.15	38.96	158.67	39.24	189.53	39.16
Layer-wise MoS	1	99.42	39.34	158.68	40.29	205.55	41.24
Layer-wise MoS	2	99.60	39.32	156.48	40.29	209.80	41.13
Layer-wise MoS	3	119.65	39.32	163.17	40.36	208.52	41.18
Neuron-wise MoS	1	97.63	39.55	200.17	40.02	184.09	41.04
Neuron-wise MoS	2	100.46	39.55	155.96	40.04	188.87	41.15
Neuron-wise MoS	3	100.47	39.57	157.26	40.04	190.40	41.17

Table 13: Stability of the evolutionary search w.r.t. different seeds on the WMT'14 En-Fr task. Search quality is measured in terms of latency and sampled (direct) supernet performance (BLEU) of the models in the pareto front.

# layers in router function	BLEU (↑)
2-layer	**26.61**
3-layer	26.14
4-layer	26.12

Table 14: Validation BLEU of different router functions for neuron-wise MoS on the WMT'14 En-De task.

Supernet	m	BLEU (↑)	Supernet GPU Memory (↓)
HAT	-	39.13	11.4 GB
Layer-wise MoS	2	**40.55**	15.9 GB
Layer-wise MoS	4	40.33	16.1 GB

Table 15: Impact of increasing the number of expert weights 'm' for the WMT'14 En-Fr task. The architecture is the top model from the pareto front of HAT for the latency constraint of 200 ms.

Supernet	BLEU (↑)	SacreBLEU (↑)
HAT	26.25	25.68
Layer-wise MoS	27.31	26.7
Neuron-wise MoS	**27.59**	**27.0**

Table 16: Performance of supernet as measured by BLEU and SacreBLEU for the latency constraint of 150 ms on the WMT'14 En-De task.

Supernet	Overall Time (↓)	Supernet Training Time (↓)	Search Time (↓)	Additional Training Time (↓)
HAT	508 hours	**248 hours**	**3.7 hours**	256 hours
Layer-wise MoS	407 hours (20%)	262 hours (-5.6%)	4.5 hours (-21.6%)	140 hours (45.3%)
Neuron-wise MoS	**394 hours (22%)**	266 hours (-7.3%)	4.3 hours (-16.2%)	**124 hours (51.6%)**

Table 17: Breakdown of the overall time savings of MoS supernets vs. HAT for computing pareto front (latency constraints: 100 ms, 150 ms, 200 ms) for the WMT'14 En-De task. Overall time (measured as single NVIDIA V100 hours) includes supernet training time, search time, and additional training time for the optimal architectures. Savings in parentheses.