AutoMoE: Heterogeneous Mixture-of-Experts with Adaptive Computation for Efficient Neural Machine Translation

Ganesh Jawahar[♣], **Subhabrata Mukherjee**[♠]
Xiaodong Liu[♠], **Young Jin Kim**[♠], **Muhammad Abdul-Mageed**[♣♢], **Laks V.S. Lakshmanan**[♣]
Ahmed Hassan Awadallah[♠], **Sebastien Bubeck**[♠], **Jianfeng Gao**[♠]

[♣]University of British Columbia, [♠]Microsoft Research, [♢]MBZUAI

Abstract

Mixture-of-Expert (MoE) models have obtained state-of-the-art performance in Neural Machine Translation (NMT) tasks. Existing works in MoE mostly consider a homogeneous design where the same number of experts of the same size are placed uniformly throughout the network. Furthermore, existing MoE works do not consider computational constraints (e.g., FLOPs, latency) to guide their design. To this end, we develop `AutoMoE` – a framework for designing heterogeneous MoE's under computational constraints. `AutoMoE` leverages Neural Architecture Search (NAS) to obtain efficient sparse MoE sub-transformers with $4\times$ inference speedup (CPU) and FLOPs reduction over manually designed Transformers, with parity in BLEU score over dense Transformer and within 1 BLEU point of MoE SwitchTransformer, on aggregate over benchmark datasets for NMT. Heterogeneous search space with dense and sparsely activated Transformer modules (e.g., *how many experts? where to place them? what should be their sizes?*) allows for adaptive compute – where different amounts of computations are used for different tokens in the input. Adaptivity comes naturally from routing decisions which send tokens to experts of different sizes. `AutoMoE` code, data, and trained models are available at `https://aka.ms/AutoMoE`.

1 Introduction

Sparsely activated models like the Mixture-of-Experts (MoE) (Fedus et al., 2022b) perform conditional computation in which only a subset of the weights of the network are activated per input. Selective compute allows us to design neural networks with a large number of model parameters, without significant increase in the computational cost. With increased capacity, these sparse models have demonstrated state-of-the-art performance in natural language tasks such as neural machine translation (NMT) (Kim et al., 2021; Kudugunta et al., 2021; Zuo et al., 2022).

MoE architectures require several design choices: *(a) Expert placement:* Identifying Transformer layers for introducing expert sub-networks. *(b) Number of experts:* How many experts to place in different layers? *(c) Expert FFN size*: What should be the feedforward network (FFN) size for each expert? Given the large search space of potential architectures and the exorbitant computational cost of training and evaluating them, existing approaches manually design MoE architectures from a highly-restricted homogeneous space. For instance, they use the same number of experts of the same capacity in different layers and make ad-hoc decisions like introducing experts in every other layer (Fedus et al., 2022b; Kim et al., 2021; Zuo et al., 2022; Du et al., 2022; Artetxe et al., 2021) or every four layers (Zoph et al., 2022).

While these MoE's support conditional computation, homogeneity (specifically, fixed-size experts) results in the same amount (albeit different subsets) of weights to be applied to each input. We hypothesize that this is not an optimal solution and that we can reduce the number of experts (in some layers) to reduce communication cost, and the size (of some experts) to reduce computation cost resulting in reduction in model size, FLOPs and latency without much quality degradation.

This naturally extends MoEs to be adaptive compute models (similar to work on early exit (Schuster et al., 2022)) where different amounts of computations are used for different inputs. The adaptivity comes naturally from the routing decisions which would send tokens to experts of different sizes.

The above observations are depicted in Table 1, which shows demonstrative examples of manually designed MoE's vs. those designed by our `AutoMoE` framework. We compare these architectures against various computational metrics (e.g., latency, FLOPs, active MoE parameters), archi-

*Correspondence to {ganeshjwhr@gmail.com, subhabrata.mukherjee@microsoft.com}.

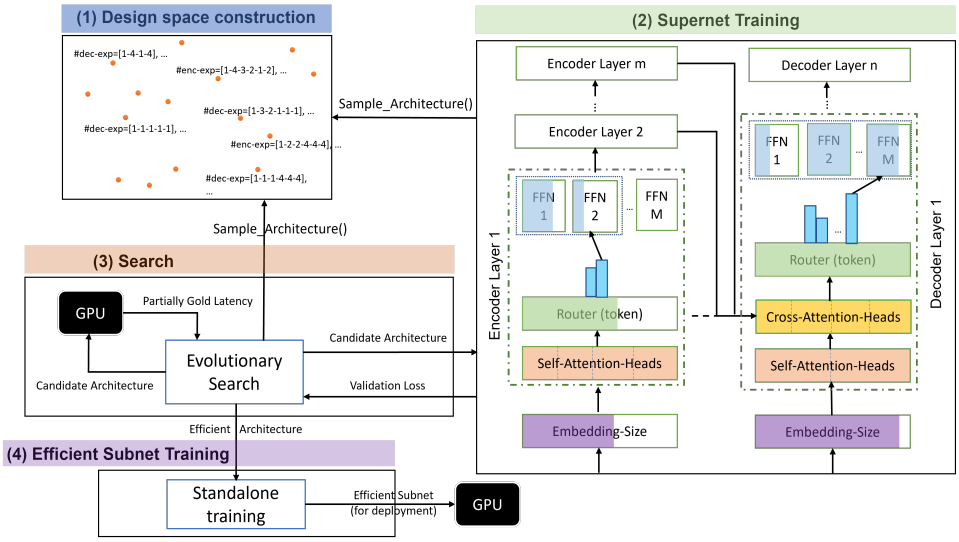


Figure 1: AutoMoE Framework. (1) Heterogeneous MoE with variable dimensions for dense Transformer blocks and sparsely activated expert modules. (2) Supernet training by sampling subnetworks from search space and training them by sharing common weights with Supernet. (3) Evolutionary search to find efficient architectures by (a) sampling MoE subnetworks from the search space; (b) using latency measured in the target device; and (c) performance estimation from Supernet as feedback for iterative optimization via crossover and mutation. (4) Efficient MoE subnetwork(s) from evolutionary search is trained on downstream task.

tectural configurations and task performance. For the most efficient configuration (last row in the table), AutoMoE reduces the number of decoder layers, compensating for the capacity with increased experts in the bottom layer, and places most of the experts in the encoder. Overall AutoMoE introduces the following components and contributions:

- *Heterogeneous design with adaptive computation* for MoEs with variable number, size and placement of experts in both encoders and decoders.
- Extends *Supernet training* and evolutionary search from prior work on dense Transformers to new search space of sparse MoE's. This combines all possible MoE sub-architectures in a single graph; jointly training them via weight-sharing; and searching for optimal one with best possible performance on a downstream task satisfying a user-specified computational constraint.
- Experiments on NMT benchmarks demonstrate AutoMoE-designed MoE's to obtain $4\times$ inference speedup on CPU and equal FLOPs reduction over manually designed Transformers, with parity in BLEU with dense Transformer and within 1 BLEU point of MoE SwitchTransformer. Further, it outperforms NAS methods in the dense search space (e.g., $1.3\times$ and $2.4\times$ FLOPs reduction and inference speedup over HAT (Wang et al., 2020) and Evolved Transformer (So et al., 2019)).

2 Background

Mixture-of-Experts: MoE's have a rich literature in machine learning dating back to the early 90s (Yuksel et al., 2012). They have received significant attention with works such as (Shazeer et al., 2017), Switch Transformers (Fedus et al., 2022b), GShard (Lepikhin et al., 2020), BASE (Lewis et al., 2021), Hash (Roller et al., 2021), GLaM (Du et al., 2022), Stochastic Experts (Zuo et al., 2022), Gating Dropout (Liu et al., 2022) and ST-MoE (Zoph et al., 2022). Some crucial differences in these works include choice of expert routing function, expert placement technique, stability/performance enhancement techniques and nature of the task (pre-training vs. fine-tuning). Some challenges in building sparse expert models include: (i) lack of diversity in expert design (expert layer selection, number of experts, expert size, etc.), (ii) training instability, (iii) poor out-of-distribution generalization, (iv) cross-task adaptation of pre-trained models, (v) communication bottleneck, (vi) high memory and (vii) expert load balancing issue, to name a few. A comprehensive review of recent sparse expert models can be found at (Fedus et al., 2022a).

MoE design: Most works in MoE rely on ad-hoc manual choices for expert placement, number of experts and their sizes. Existing approaches mostly use manual design, where they add experts on (i) alternate layers (Fedus et al., 2022b; Kim et al., 2021;

Machine Translation	#Experts in each layer		Accuracy	Computational Footprint		
Design Approach	**Encoder**	**Decoder**	**BLEU**	**Latency**	**# Active Params**	**FLOPs (G)**
Manually designed (every layer)	4-4-4-4-4-4	4-4-4-4-4-4	27.87	861ms	56M	3.4
Manually designed (every other layer)	1-4-1-4-1-4	1-4-1-4-1-4	28.48	794ms	56M	3.4
AutoMoE	1-1-4-4-4-1	4-1-1-1	28.15	585ms	46M	2.9

Table 1: Manual vs. AutoMoE designed MoE for illustration with 6-layer encoder-decoder Transformer. Detailed results in Table 4. We report computational metrics (measured on Intel Xeon CPU) and BLEU score of MoE's on WMT'14 En-De MT task. Number of experts per layer are separated by hyphen (-) for encoder and decoder.

Zuo et al., 2022; Du et al., 2022; Artetxe et al., 2021), (ii) every four layers (Zoph et al., 2022), or (iii) final few layers (Rajbhandari et al., 2022). While these MoE's support conditional computation, they generally do not support adaptive compute since same number of expert parameters apply to every input, largely given by their homogeneous design (e.g., all experts of same size). Further, MoE design is generally agnostic to computational constraints (e.g., latency, memory) of the hardware in which the MoE model has to be deployed.

Neural Architecture Search (NAS): Given a search space of architectures and efficiency constraints (e.g., model size, latency), NAS typically aims to identify the optimal architecture that maximizes the task performance, while satisfying the efficiency constraints. NAS has been recently used for natural language understanding tasks to build efficient BERT (Devlin et al., 2019) and GPT (Brown et al., 2020) based pre-trained language models (Xu et al., 2021; Yin et al., 2021; Xu et al., 2022a,b; Gao et al., 2022; Dong et al., 2021; So et al., 2021; Javaheripi et al., 2022) as well as for machine translation tasks (So et al., 2019; Wang et al., 2020). Hardware aware transformers (HAT) (Wang et al., 2020) is a state-of-the-art NAS framework with dense Transformers for MT that uses hardware latency as feedback for optimization.

However, all of the above NAS works consider a search space with densely activated Transformers and non-MoE architectures, They primarily search over typical Transformer architectural hyper-parameters like number of layers, attention heads and hidden size. In contrast, we propose the first NAS framework that searches for efficient sparsely activated Mixture-of-Expert modules in Transformers. Our heterogeneous AutoMoE framework addresses some longstanding design choices for MoE's like *how many experts? which layers to place them? what should be their sizes?* and so on.

3 Designing Heterogeneous Mixture-of-Experts

We now present the components of AutoMoE framework (illustrated in Figure 1) for designing efficient MoE's under computational constraints.

3.1 Heterogeneous MoE Search Space

Existing MoE approaches restrict their design space by considering uniform distribution of size and number of experts placed in different Transformer layers. For instance, the standard MoE design (Fedus et al., 2022b) for an L-layer Transformer with M experts placed in alternate layers have only two possible configurations viz., $\{1\text{-}M\text{-}1\text{-}\cdots\}, \{M\text{-}1\text{-}M\text{-}\cdots\}$. (a) Our design space allows *variable number of experts* in each layer resulting in M^L possible configurations. (b) Furthermore, our design space also allows *variable expert size*, e.g., by modulating the width of the feedforward (FFN) subnetworks for different experts. Considering N possible FFN dimensions for each expert results in N^{M^L} possible configurations for designing the expert space. (c) Finally, given the autoregressive nature of tasks like neural machine translation, the inference cost is dominated by the decoder (Kasai et al., 2021). For instance, for token-based MoE, decoders take $200\times$ the time per step compared to encoders at peak throughput (Kudugunta et al., 2021). Therefore, we further consider *variable number of decoder layers* along with the above choices for expert placement and expert capacity. ***To the best of our knowledge, our work is the first to study such a flexible and exhaustive design space for MoE architectures***.

In addition to heterogeneous experts, we allow flexible design for non-expert Transformer modules like the number of attention heads, hidden size and intermediate feedforward dimensions. This heterogeneous design of non-MoE, i.e., dense Transformer modules, has been explored in prior works such as HAT (Wang et al., 2020) for generation

Attributes	AutoMoE	Transformer Base / Big
Encoder-Embedding-Size	{512, 640}	512 / 1024
Decoder-Embedding-Size	{512, 640}	512 / 1024
#Encoder-Layers	{6}	6
#Decoder-Layers	{1, 2, 3, 4, 5, 6}	6
Encoder-QKV-Dim	{512}	512 / 1024
Decoder-QKV-Dim	{512}	512 / 1024
#Encoder-Self-Att-Heads (PL)	{4, 8}	8 / 16
#Decoder-Self-Att-Heads (PL)	{4, 8}	8 / 16
#Decoder-Cross-Att-Heads (PL)	{4, 8}	8 / 16
#Decoder-Arbitrary-Att (PL)	{-1, 1, 2}	-1
Encoder-FFN-Intermediate-Size (PL, PE)	{1024, 2048, 3072}	2048 / 4096
Decoder-FFN-Intermediate-Size (PL, PE)	{1024, 2048, 3072}	2048 / 4096
#Encoder-Experts (PL)	{1, 2, $\cdots$ M}	-
#Decoder-Experts (PL)	{1, 2, $\cdots$ M}	-

Table 2: Search space of AutoMoE compared to manually configured Transformer Base / Big. 'PL' and 'PE' refer to per layer and per expert search dimensions. Decoder arbitrary attn. searches last k encoder layers to attend for each decoder layer. FFN size varies across layers and experts. M denotes maximum experts per layer.

tasks like NMT, and AutoDistil (Xu et al., 2022a) for understanding tasks like those in the GLUE benchmark (Wang et al., 2018). Table 2 shows our search space. We demonstrate our heterogeneous MoE search to perform better than both manual and NAS-searched architectures in the dense space.

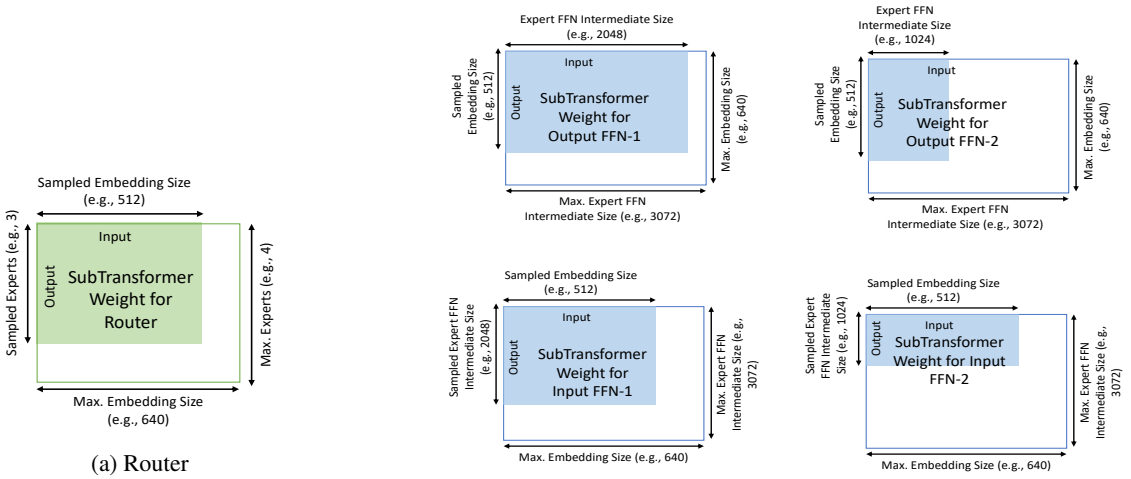
3.2 Supernet Training for MoE

AutoMoE leverages the idea of Supernet training from prior works (Cai et al., 2020; Xu et al., 2022a; Wang et al., 2020) in Neural Architecture Search that were developed for standard non-MoE architectures. We extend Supernet training to the search space for MoE's by incorporating experts, gating and routing protocols. Typically, a Supernet consists of thousands of subnetworks that are all jointly trained via weight-sharing. The Supernet for AutoMoE is the largest sparsely activated MoE in the search space. It consists of the maximum number of experts (M) placed in every layer of the Transformer in both encoder and decoder. Each expert FFN has the maximum intermediate hidden size in the search space. Similar principles apply to the non-expert dense modules initialized with corresponding full dimension.

The Supernet is trained with the following steps: (i) sample a candidate architecture randomly from the search space (Guo et al., 2020); (ii) train the sampled architecture by extracting the common portion of weights from different layers in the Supernet (i.e., by weight sharing) for one training step on the task; (iii) repeat steps (i) and (ii) until the training budget is exhausted. Once the Supernet training converges, we can obtain a quick accuracy estimate for a candidate architecture (i.e. subnetwork) by extracting its shared weights from the Supernet and evaluating on the validation set.

The key challenge here is to build weight sharing techniques for MoE components, which include: (i) *router*: a neural network that is trained to route each token (of 'embedding size') in an incoming example to exactly one expert (out of M experts) for top-1 routing; (ii) *FFN expert*: a standard Transformer FFN block that has unique weights and is learned independently. AutoMoE's expert layers follow the Switch Transformer (Fedus et al., 2022b) specification. For subnetwork extraction from the Supernet, AutoMoE extracts front rows and front columns of the Supernet's router weight matrix, corresponding to the subnet design. For example, consider the Supernet's router to be designed for 4 experts and 640 embedding size with the shape of the router weight matrix as 4×640. Consider a sampled subnet during Supernet training to consist of $3 < 4$ experts and $512 < 640$ embedding size with the subnet's router matrix as 3×512. To populate this matrix, we extract the first 3 rows and first 512 columns from the Supernet's weight matrix (as illustrated in Figure 2 (a)). Such a weight sharing technique allows us to design hetegogeneous MoE architectures with varying number of experts in each Transformer layer.

AutoMoE also extracts front rows and front columns from the weight matrices of each FFN expert from the Supernet, corresponding to the subnet design. For the previous example, assume the intermediate FFN size of each expert in the Supernet to be 3072 (shape of weight matrix for first FFN layer is 3072×640 and second FFN layer is 640×3072). Assume the sampled subnet to be designed for 2 experts with intermediate FFN size of one expert to be 2048 while the other to be 1024. For the first expert, the weight matrices of the subnet of shape 2048×512 (Input) and 512×2048 (Output) are extracted from the first 2048 rows, 512 columns (Input) and first 512 rows, 2048 columns



(a) Router

(b) Experts (e.g., 2 FFN experts)

Figure 2: Weight sharing in the MoE Supernet for sparsely activated expert modules.

(Output) of the corresponding Supernet weights. For the second expert, the weight matrices of shape 1024×512 (Input) and 512×1024 (Output) are extracted from the first 1024 rows, 512 columns (Input) and first 512 rows, 1024 columns (Output) of the corresponding Supernet weights. This example is illustrated in Figure 2 (b). The subnet extraction technique does not extract weights from the third and fourth experts of the Supernet as the subnet is designed to have only two experts (not shown in the figure). Such a weight sharing technique allows us to design architectures with varying intermediate FFN size for each expert. Additional techniques for improving expert capacity such as stacking FFNs, and techniques for improving Supernet performance with sandwich sampling (Yu et al., 2019), inplace knowledge distillation (Yu et al., 2019), gradient conflict reduction (Gong et al., 2022) are left for future work.

3.3 Searching for Efficient MoE Subnetwork with Computational Constraint

AutoMoE search is based on an evolutionary algorithm that takes the hardware computational constraint (e.g., CPU latency $\leq$ 600ms) as input and aims to identify the MoE subnetwork from the Supernet which achieves maximum accuracy for the task while satisfying the constraint. The algorithm works by sampling an initial set of MoE candidate architectures randomly from the Supernet; evolving the top architectures iteratively by mutation; followed by crossover; until the search iterations are exhausted. Candidate MoE architectures are easily ranked by the Supernet performance estimator based on the validation score for the task.

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

Table 3: Machine translation benchmark data.

Latency estimate for each architecture is obtained by measuring the latency directly on the target device. The standard approach measures gold latency for forward propagation of a batch of examples for a large number (e.g., 300) of passes and then computes the truncated mean (after removing bottom and top 10% outlier latencies). This latency estimation can be costly given the large space of candidate architectures. To overcome this challenge, AutoMoE uses *partially gold latency*, which is obtained by forward propagation of a batch of examples for a small number (e.g., 100) of passes and then computing truncated mean. After the search is completed, the MoE architecture with the highest performance is selected as the optimal one.

3.4 Training Efficient MoE Sub-Transformer

Once the optimal MoE architecture is identified, we train the model weights for the final architecture to convergence for the same number of training steps as our baseline models for a fair comparison.

4 Experiments

Datasets and evaluation metrics.

We evaluate AutoMoE on standard machine translation benchmarks: WMT'14 En-De, WMT'14 En-Fr and WMT'19 En-De with dataset statistics in

Dataset	Network	#Active Params (M)	Sparsity (%)	FLOPs (G)	BLEU	GPU hours	Latency (ms)
WMT'14 En-De							
Transformer-Big	Dense	176	0	10.6 (1×)	28.4	184	2199 (1×)
SwitchTransformer-Big	Sparse	176	36	10.6 (1×)	**28.8**	236	
Evolved Transformer	NAS over Dense	47	0	**2.9** (3.7×)	28.2	2,192,000	-
HAT	NAS over Dense	56	0	3.5 (3×)	28.2	264	669 (3.3×)
Random Search	NAS over Sparse	42	21	2.2 (4.8×)	27.3	126	416 (5.3×)
AutoMoE (6 Experts)	NAS over Sparse	**45**	62	**2.9** (3.7×)	28.2	224	504 (4.4×)
WMT'14 En-Fr							
Transformer-Big	Dense	176	0	10.6 (1×)	41.2	240	2199 (1×)
SwitchTransformer-Big	Sparse	176	36	10.6 (1×)	**42.3**	234	
Evolved Transformer	NAS over Dense	175	0	10.8 (1×)	41.3	2,192,000	-
HAT	NAS over Dense	57	0	3.6 (2.9×)	41.5	248	723 (3×)
Random Search	NAS over Sparse	42	21	2.2 (4.8×)	40.3	130	416 (5.3×)
AutoMoE (6 Experts)	NAS over Sparse	**46**	72	**2.9** (3.7×)	41.6	236	547 (4×)
AutoMoE (16 Experts)	NAS over Sparse	135	65	3.0 (3.5×)	41.9		672 (3.3×)
WMT'19 En-De							
Transformer-Big	Dense	176	0	10.6 (1×)	46.1	184	2199 (1×)
SwitchTransformer-Big	Sparse	176	36	10.6 (1×)	**47.0**	223	
HAT	NAS over Dense	63	0	4.1 (2.6×)	45.8	264	758 (2.9×)
Random Search	NAS over Sparse	42	21	2.2 (4.8×)	43.7	126	416 (5.3×)
AutoMoE (2 Experts)	NAS over Sparse	**45**	41	**2.8** (3.8×)	45.5	248	558 (3.9×)
AutoMoE (16 Experts)	NAS over Sparse	69	81	3.2 (3.3×)	45.9		656 (3.3×)

Table 4: Comparison of AutoMoE vs. baselines with Pareto-optimal architectures highlighted in blue color. We report active model parameters, and sparsity measured as non-active parameters as a percentage of total parameters. We train all baselines and AutoMoE for the same $40K$ training steps for fair comparison to report BLEU[1]. Training time (with search, if applicable) is reported in hours for one Nvidia V100 GPU. Inference latency is measured in Intel Xeon CPU. AutoMoE significantly reduces FLOPs and latency with parity in BLEU, on aggregate, over NAS methods in dense search space (e.g., $1.3\times$ and $2.4\times$ FLOPs reduction and speedup over HAT and Evolved Transformer). AutoMoE with Random Search obtains the best speedup but results in significant regression in BLEU.

Table 3. We use pre-processed datasets and evaluation setup from (Wang et al., 2020). We report BLEU score (Papineni et al., 2002) as a performance metric with beam of size 5 and a length penalty of 0.6 (for WMT).

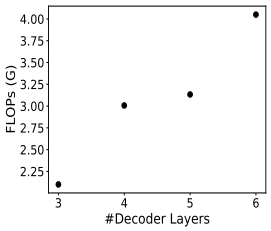
Baselines. We compare AutoMoE against both manually designed and NAS-searched architectures.

For **manual baselines**, we consider: (a) densely activated Transformers (Vaswani et al., 2017) with no experts; (b) sparsely activated MoE with homogeneous experts (i.e. same number and FFN size) placed in every other layer (Fedus et al., 2022b; Kim et al., 2021; Zuo et al., 2022; Du et al., 2022; Artetxe et al., 2021).

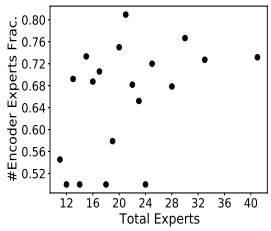
For **NAS baselines**, we consider (c) HAT (Wang et al., 2020), which is a Supernet-based state-of-the-art NAS framework for identifying efficient dense sub-Transformers for neural machine translation (same task setting as ours); and (d) Evolved Transformer (So et al., 2019) which is one of the earlier works on finding efficient dense sub-Transformers with evolution-based architecture search. *Note that both the NAS baselines apply only to dense non-MoE transformers, and* AutoMoE *is the first work to leverage NAS to identify efficient sparse MoE sub-transformers.* Finally, we consider (e) AutoMoE with Random Search (typically treated as a strong baseline for NAS) that samples an MoE subnetwork (given latency constraints) randomly from AutoMoE search space and trains it till convergence.

Training configurations and search space. All the baselines and AutoMoE including the Supernet and final model are trained with the same setting for fair comparison. All the models are trained for $40K$ steps, with a warmup of $10K$ steps from 10^{-7} to 10^{-3} and use cosine annealing to 10^{-7} for the rest of the steps. All models are trained using fairseq toolkit (Ott et al., 2019) with an effective batch size of $524K$ tokens on 16 V100 GPUs. All the NAS baselines have the same search space for dense Transformer modules (e.g., number of decoder layers, q-k-v dimension, attention heads, etc.) with AutoMoE further incorporating MoE relevant aspects (e.g., experts, gating, routing, etc.) in the search space. The number of encoder layers is kept fixed for all the NAS baselines including AutoMoE since the latency is primarily determined by the decoders for autoregressive generation (as we discuss in Section 5.2).

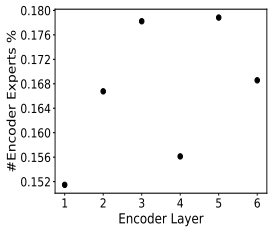
Evolutionary search setup. For performance estimation, we monitor the validation loss of subnets on the NMT task. We compute latency by measuring the time taken to perform translation from a source sentence to a target sentence with same desired input/output length (30 for WMT) and original beam settings (see Section 4) on target device



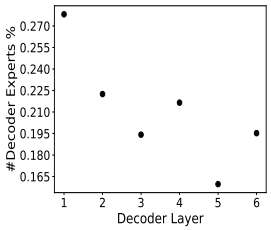
(a) #Decoder layers vs. FLOPs.



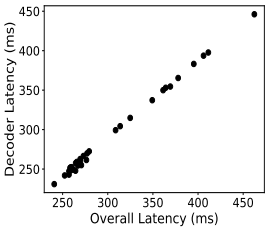
(b) Encoder experts as ratio of total experts.



(c) Encoder layer vs. #Expert (%).



(d) Decoder layer vs. #Expert (%).



(e) Overall vs. Decoder latency.

Figure 3: Architecture analysis for AutoMoE-generated MoEs. We sample several architectures from the Pareto for AutoMoE subnets, and report aggregate statistics in terms of the impact on different computational metrics.

(Intel Xeon CPU). We measure latency 300 times for gold (to report final metrics) and 100 times for partially gold (during evolutionary search) respectively; discard top and bottom 10% (outlier latency) and compute mean of the rest. Hyper-parameter settings for evolutionary search include: 15 as iterations, 125 as population size, 25 as parents' size, 50 as mutation population size with mutation probability of 0.3 and 50 as crossover population size. Unless otherwise stated, latency constraint for all experiments is set to $600ms$.

5 Results

5.1 AutoMoE vs. Baseline Performance

Table 4 presents a comparison of AutoMoE with baselines on several computational metrics and task performance. We report the number of parameters without embedding weights, and FLOPs without the last decoding layer for all the models, consistent with (Wang et al., 2020) evaluation. AutoMoE-generated sparse MoE sub-Transformers obtain $4\times$ reduction in FLOPs over both manually designed (densely activated) Transformer-Big, and (sparsely activated) MoE SwitchTransformer-Big with experts in every layer, and equivalent inference speedups on CPU. Compared to NAS baselines like Evolved Transformer (So et al., 2019) and HAT (Wang et al., 2020) that generate densely activated sub-Transformers, AutoMoE improves on FLOPs and latency by $2.4\times$ and $1.3\times$ respectively with parity in BLEU score on aggregate. Notably, Supernet-based AutoMoE and HAT have massively reduced amortized training cost (GPU hours) compared to Evolved Transformer with progressive evolutionary search. AutoMoE with Random Search, a strong NAS baseline, obtains the best speedup but with significant performance regression.

Compared to all other models (both dense and sparse), we observe AutoMoE to generate networks with high sparsity resulting in massively reduced active parameters and FLOPs. For the NAS models, we train the top-2 sub-Transformers in the Pareto and report the one with the best trade-off in BLEU vs. FLOPs on the validation set. Maximum experts for the best performance vary for different tasks, with 6 experts for WMT'14 En-De, 16 experts for WMT'14 En-Fr and WMT'19 En-De – given the latter two datasets are $10\times$ larger than the former.

5.2 Analysis

Decoder layers vs. FLOPs. Figure 3 (a) shows the average FLOPs for several AutoMoE architectures with different decoder layers as obtained during our search (varying from 3 to 6) from the Pareto, and baseline models. Notice that the FLOPs increase with increase in decoder layers, given the auto-regressive nature of NMT tasks which require generating tokens sequentially. In contrast to manually designed Transformers with 6 decoder layers (both dense and sparsely activated MoE variants), AutoMoE- and HAT-searched architectures reduce the number of decoder layers with a resulting decrease in both FLOPs and latency. This is also evident in Figure 3 (e) which shows that decoder latency dominates the total inference latency for all

Model Dataset	Encoder #Experts per layer	Encoder Expert FFN Inter Size	Decoder #Experts per layer	Decoder Expert FFN Inter Size
Std-expert				
WMT'14 En-De	5-1-1-1-2-1	3072-3072-3072-3072-2048-3072	1-1-1-1	3072-3072-3072-3072
WMT'14 En-Fr	1-4-2-6-5-5	3072-3072-3072-3072-3072-3072	2-1-1-3	3072-3072-3072-3072
WMT'19 En-De	1-1-2-1-2-1	3072-3072-3072-3072-3072-2048	1-1-1-2	3072-3072-3072-3072
Fract-expert				
WMT'14 En-De	3-2-3-4-1-3	[2048-3072-2048]-[3072-1024]-[3072-3072-1024]-[3072-1024-3072-2048]-3072-[3072-1024-3072]	3-1-1-1	[3072-1024-2048]-3072-3072-3072
WMT'14 En-Fr	6-2-3-4-4-5	[2048-1024-2048-1024-1024-3072]-[2048-2048]-[3072-3072-2048]-[3072-3072-2048-3072]-[3072-1024-1024-2048]-[2048-3072-3072-2048-2048]	2-1-4-2	[3072-3072]-3072-[3072-3072-3072-2048]-[3072-2048]
WMT'19 En-De	2-3-1-2-6-1	[3072-3072]-[3072-3072-3072]-3072-[3072-2048]-[3072-1024-2048-3072-1024-2048]-3072	2-4-1-1	[3072-3072]-[3072-1024-2048-3072]-3072-3072

Table 5: AutoMoE-generated Pareto-optimal architectures for different datasets. FFN intermediate sizes for fractional experts (i.e. varying expert sizes within each layer) are enclosed within square brackets.

the models by more than 90%.

Expert distribution in encoder vs. decoder. Figure 3 (b) plots the number of encoder experts as ratio of total experts for AutoMoE-generated sub-Transformers. We observe that AutoMoE assigns significantly larger number of experts to encoder as compared to the decoder. As a result, encoders have much higher capacity (i.e., encoder parameters as a proportion of overall parameters) than decoders. This correlates with the earlier observation that models with higher encoder layers compared to decoder layers enjoy better latency-performance trade-off (Kasai et al., 2021). Our findings from AutoMoE designed architectures indicate that the number of layers and experts are two knobs that jointly help in modulating encoder capacity and decoder latency to design efficient MoE.

Expert distribution in different layers. Figures 3 (c) and (d) show the percentage of experts allocated to different layers for encoders and decoders – averaged over several sampled architectures from AutoMoE Supernet. Notice that the middle encoder layers ($3^{rd}, 5^{th}$) are allocated the maximum number of experts, while the first layer receives the least. The trend reverses for decoder, with the first layer receiving most experts with gradual reduction in expert allocation. This is also consistent with keeping decoders light by dropping layers to reduce latency; while compensating for the reduced capacity with increased experts in the first few layers.

Latency vs. FLOPs as constraint for search. Table 6 presents the impact of latency and FLOPs as computational constraints on the performance-efficiency trade-off. Constraining FLOPs results in models that fully exhaust the FLOPs budget; while leading to higher latency. On the other hand, constraining latency tends to under-utilize the budget leading to relatively superior FLOPs and latency, providing a stricter control.

Search Constraint	BLEU	FLOPs (G)	Latency (ms)
Latency $\leq$ 200ms			
HAT	41.45	3.6	212
AutoMoE (2 Experts)	41.23	2.9	176
AutoMoE (4 Experts)	41.22	3.0	198
FLOPs $\leq$ 3 GFLOPs			
HAT	40.89	3.0	158
AutoMoE (2 Experts)	41.09	3.0	216
AutoMoE (4 Experts)	41.10	3.0	229

Table 6: Impact of latency and FLOPs constraints on WMT'14 En-Fr dataset. Latency is computed on 1 NVIDIA V100 GPU.

Pareto-optimal AutoMoE generated MoE architectures. Table 5 shows sparsely activated MoE architectures designed by two variants of AutoMoE ('std-expert': expert FFN size same in each layer and variable across; 'fract-expert': fully heterogeneous expert size) for different datasets with the best trade-off in BLEU vs. latency. On aggregate 71% of the experts are allocated to the encoder compared to the decoder. Meanwhile, 70% of the expert layers in 'fract-expert' architectures have 2 or more experts, out of which more than 75% of the expert layers have varying capacities (i.e., experts with different FFN intermediate size). Figures 4, 5, 6 in Appendix show full architecture (embedding size, layers, heads, experts, placement, sizes, etc.) of AutoMoE subnets on WMT14 En-De,

[1]We use same hyper-parameters for all models with no tuning (provided in code). Given $40K$ training steps for each model and no tuning, MoE numbers may not be comparable to SOTA numbers which typically train for more steps. HAT and Evol. Transformer numbers are reported from (Wang et al., 2020). We follow their evaluation and reporting protocol.

Search Space Variation	BLEU	FLOPs
HAT	28.2	3.5G
AutoMoE (2 Experts) w/ fixed encoder layers	**28.2**	**2.9G**
Varying number of encoder layers		
HAT w/ #Encoder-Layers $\in$ {1-6}	28.1	3.4G
AutoMoE (2 Experts) w/ #Encoder-Layers $\in$ {1-6}	28.3	3.7G
AutoMoE (2 Experts) w/ manually designed homogeneous experts		
1-2-1-2-1-2	28.3	3.5G
1-1-1-2-2-2	28.3	3.8G
2-2-2-1-1-1	28.3	3.1G
AutoMoE w/ Identity Expert FFN size $\in$ {0, 3072}	28.1	2.7G

Table 7: Variations in AutoMoE's search space on WMT'14 En-De dataset.

WMT14 En-Fr and WMT19 EN-De respectively.

MoE Search space variations. Table 7 presents the impact of search space choices on MoE efficiency and performance trade-off. The first variation is to make '#Encoder Layers' an elastic search dimension. Note that both HAT and AutoMoE consider the number of encoder layers to be fixed (refer to Table 2). We observe that varying encoder layers has a relatively poor trade-off on model performance vs efficiency as compared to varying decoder layers, re-inforcing our prior observations on the importance of encoder capacity and depth.

In the second variation (see third major row), we fix the expert architecture (with 2 experts manually placed uniformly) in the search space and only search for standard Transformer hyper-parameters. Observe that AutoMoE-designed models have better FLOPs than such manually designed ones.

The last variation introduces identity or dummy experts (i.e., expert with 0 intermediate FFN size, equivalent to identity operation). This explores the idea that we can *skip* the computation for some of the tokens based on context rather than always forcing them through an FFN. We observe identity experts to marginally hurt the performance but significantly reduce FLOPs (see last major row).

6 Conclusion

AutoMoE is the first framework to design heterogeneous MoE's under computational constraints. It supports adaptive computation i.e. variable compute for different inputs with variable-size experts. It leverages NAS to explore a heterogeneous search space with variable number of experts, sizes, and placement choices; alongside other standard Transformer architectural hyper-parameters. AutoMoE generated MoE subnetworks reduce FLOPs and latency over both manually designed and NAS-searched architectures on benchmark MT tasks.

7 Limitations

Given our focus on finding efficient MoE models *under computational constraints*, AutoMoE search space and evaluation has been restricted in scale to big-sized Transformer models for benchmark MT tasks. A natural extension of this work is to explore the limits of MoE models like SwitchTransformers (Fedus et al., 2022b) and GShard (Lepikhin et al., 2020) that are significantly larger containing billions to trillions of parameters; as well as designing sparse and transferable efficient expert models (Zoph et al., 2022) for diverse types of tasks like reasoning, summarization and understanding.

The limitations of this work are as follows:

1. Sandwich sampling (Yu et al., 2019), inplace knowledge distillation (Yu et al., 2019), and gradient conflict reduction (Gong et al., 2022) are popular techniques to improve the training procedure of supernet. It would be interesting to study the impact of these techniques to improve AutoMoE's supernet.

2. AutoMoE uses the hidden dimension of intermediate feedforward network (FFN) to modulate the capacity of each expert. It would be interesting to study other techniques to modulate expert capacity such as stacking variable number of hidden layers in FFN.

3. The backbone of AutoMoE's supernet uses Switch Transformer, which adds FFN based expert layers and routes each token to exactly one expert (top-1 routing). It would be interesting to: (i) search for the number of tokens to route, and (ii) search for the Transformer component (e.g., FFN, self-attention projection layers, LayerNorm) to add expert layers.

4. AutoMoE's search space contains classical Transformer components such as multi-head attention and FFN layers. It would be interesting to add components that are efficient by design such as convolutional layer, FLASH (Hua et al., 2022), and g-MLP (Liu et al., 2021).

Acknowledgements

MAM acknowledges support from Canada Research Chairs (CRC), the Natural Sciences and Engineering Research Council of Canada (NSERC; RGPIN-2018-04267), Canadian Foundation for Innovation (CFI; 37771), and Digital Research Al-

liance of Canada.[2] Lakshmanan's research was supported in part by a grant from NSERC (Canada).

References

Mikel Artetxe, Shruti Bhosale, Naman Goyal, Todor Mihaylov, Myle Ott, Sam Shleifer, Xi Victoria Lin, Jingfei Du, Srinivasan Iyer, Ramakanth Pasunuru, Giri Anantharaman, Xian Li, Shuohui Chen, Halil Akin, Mandeep Baines, Louis Martin, Xing Zhou, Punit Singh Koura, Brian O'Horo, Jeff Wang, Luke Zettlemoyer, Mona T. Diab, Zornitsa Kozareva, and Ves Stoyanov. 2021. Efficient large scale language modeling with mixtures of experts. *CoRR*, abs/2112.10684.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc.

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.

Chenhe Dong, Guangrun Wang, Hang Xu, Jiefeng Peng, Xiaozhe Ren, and Xiaodan Liang. 2021. EfficientBERT: Progressively searching multilayer perceptron via warm-up knowledge distillation. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 1424–1437, Punta Cana, Dominican Republic. Association for Computational Linguistics.

Nan Du, Yanping Huang, Andrew M Dai, Simon Tong, Dmitry Lepikhin, Yuanzhong Xu, Maxim Krikun, Yanqi Zhou, Adams Wei Yu, Orhan Firat, Barret Zoph, Liam Fedus, Maarten P Bosma, Zongwei Zhou, Tao Wang, Emma Wang, Kellie Webster, Marie Pellat, Kevin Robinson, Kathleen Meier-Hellstern, Toju Duke, Lucas Dixon, Kun Zhang, Quoc Le, Yonghui Wu, Zhifeng Chen, and Claire Cui. 2022. GLaM: Efficient scaling of language models with mixture-of-experts. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 5547–5569. PMLR.

William Fedus, Jeff Dean, and Barret Zoph. 2022a. A review of sparse expert models in deep learning.

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

Jiahui Gao, Hang Xu, Han Shi, Xiaozhe Ren, Philip L. H. Yu, Xiaodan Liang, Xin Jiang, and Zhenguo Li. 2022. Autobert-zero: Evolving BERT backbone from scratch. In *Thirty-Sixth AAAI Conference on Artificial Intelligence, AAAI 2022, Thirty-Fourth Conference on Innovative Applications of Artificial Intelligence, IAAI 2022, The Twelveth Symposium on Educational Advances in Artificial Intelligence, EAAI 2022 Virtual Event, February 22 - March 1, 2022*, pages 10663–10671. AAAI Press.

Chengyue Gong, Dilin Wang, Meng Li, Xinlei Chen, Zhicheng Yan, Yuandong Tian, qiang liu, and Vikas Chandra. 2022. 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*, volume 12361 of *Lecture Notes in Computer Science*, pages 544–560. Springer.

Weizhe Hua, Zihang Dai, Hanxiao Liu, and Quoc Le. 2022. Transformer quality in linear time. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 9099–9117. PMLR.

Mojan Javaheripi, Shital Shah, Subhabrata Mukherjee, Tomasz L. Religa, Caio C. T. Mendes, Gustavo H. de Rosa, Sebastien Bubeck, Farinaz Koushanfar, and Debadeepta Dey. 2022. Litetransformersearch: Training-free on-device search for efficient autoregressive language models.

Jungo Kasai, Nikolaos Pappas, Hao Peng, James Cross, and Noah Smith. 2021. Deep encoder, shallow decoder: Reevaluating non-autoregressive machine translation. In *International Conference on Learning Representations*.

Young Jin Kim, Ammar Ahmad Awan, Alexandre Muzio, Andrés Felipe Cruz-Salinas, Liyang Lu, Amr Hendy, Samyam Rajbhandari, Yuxiong He, and

[2] https://alliancecan.ca

Hany Hassan Awadalla. 2021. Scalable and efficient moe training for multitask multilingual models. *CoRR*, abs/2109.10465.

Sneha Kudugunta, Yanping Huang, Ankur Bapna, Maxim Krikun, Dmitry Lepikhin, Minh-Thang Luong, and Orhan Firat. 2021. Beyond distillation: Task-level mixture-of-experts for efficient inference. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 3577–3599, Punta Cana, Dominican Republic. Association for Computational Linguistics.

Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. 2020. Gshard: Scaling giant models with conditional computation and automatic sharding. *CoRR*, abs/2006.16668.

Mike Lewis, Shruti Bhosale, Tim Dettmers, Naman Goyal, and Luke Zettlemoyer. 2021. Base layers: Simplifying training of large, sparse models. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 6265–6274. PMLR.

Hanxiao Liu, Zihang Dai, David So, and Quoc V Le. 2021. Pay attention to mlps. In *Advances in Neural Information Processing Systems*, volume 34, pages 9204–9215. Curran Associates, Inc.

Rui Liu, Young Jin Kim, Alexandre Muzio, and Hany Hassan. 2022. Gating dropout: Communication-efficient regularization for sparsely activated transformers. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 13782–13792. PMLR.

Myle Ott, Sergey Edunov, Alexei Baevski, Angela Fan, Sam Gross, Nathan Ng, David Grangier, and Michael Auli. 2019. fairseq: A fast, extensible toolkit for sequence modeling. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics (Demonstrations)*, pages 48–53, Minneapolis, Minnesota. Association for Computational Linguistics.

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.

Samyam Rajbhandari, Conglong Li, Zhewei Yao, Minjia Zhang, Reza Yazdani Aminabadi, Ammar Ahmad Awan, Jeff Rasley, and Yuxiong He. 2022. DeepSpeed-MoE: Advancing mixture-of-experts inference and training to power next-generation AI scale. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 18332–18346. PMLR.

Stephen Roller, Sainbayar Sukhbaatar, Arthur Szlam, and Jason E Weston. 2021. Hash layers for large sparse models. In *Advances in Neural Information Processing Systems*.

Tal Schuster, Adam Fisch, Jai Gupta, Mostafa Dehghani, Dara Bahri, Vinh Q. Tran, Yi Tay, and Donald Metzler. 2022. Confident adaptive language modeling.

Noam Shazeer, *Azalia Mirhoseini, *Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. 2017. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *International Conference on Learning Representations*.

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.

David So, Wojciech Mańke, Hanxiao Liu, Zihang Dai, Noam Shazeer, and Quoc V Le. 2021. Searching for efficient transformers for language modeling. In *Advances in Neural Information Processing Systems*, volume 34, pages 6010–6022. Curran Associates, Inc.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

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. 2020. 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.

Dongkuan Xu, Subhabrata (Subho) Mukherjee, Xiaodong Liu, Debadeepta Dey, Wenhui Wang, Xiang Zhang, Ahmed H. Awadallah, and Jianfeng Gao. 2022a. Autodistil: Few-shot task-agnostic neural architecture search for distilling large language models. *ArXiv*.

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 & 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.

Yichun Yin, Cheng Chen, Lifeng Shang, Xin Jiang, Xiao Chen, and Qun Liu. 2021. AutoTinyBERT: Automatic hyper-parameter optimization for efficient pre-trained language models. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 5146–5157, Online. Association for Computational Linguistics.

Jiahui Yu, Linjie Yang, Ning Xu, Jianchao Yang, and Thomas Huang. 2019. Slimmable neural networks. In *International Conference on Learning Representations*.

Seniha Esen Yuksel, Joseph N. Wilson, and Paul D. Gader. 2012. Twenty years of mixture of experts. *IEEE Transactions on Neural Networks and Learning Systems*, 23(8):1177–1193.

Barret Zoph, Irwan Bello, Sameer Kumar, Nan Du, Yanping Huang, Jeff Dean, Noam Shazeer, and William Fedus. 2022. St-moe: Designing stable and transferable sparse expert models.

Simiao Zuo, Xiaodong Liu, Jian Jiao, Young Jin Kim, Hany Hassan, Ruofei Zhang, Jianfeng Gao, and Tuo Zhao. 2022. Taming sparsely activated transformer with stochastic experts. In *International Conference on Learning Representations*.

A Appendix

A.1 Full Architecture Design

Figure 4, 5 and 6 present the full architecture design of pareto-efficient architectures generated by `AutoMoE`.

A.2 Evolutionary Search - Stability

We study the initialization effects on the stability of the pareto front outputted by the evolutionary search for HAT. Table 8 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 HAT.

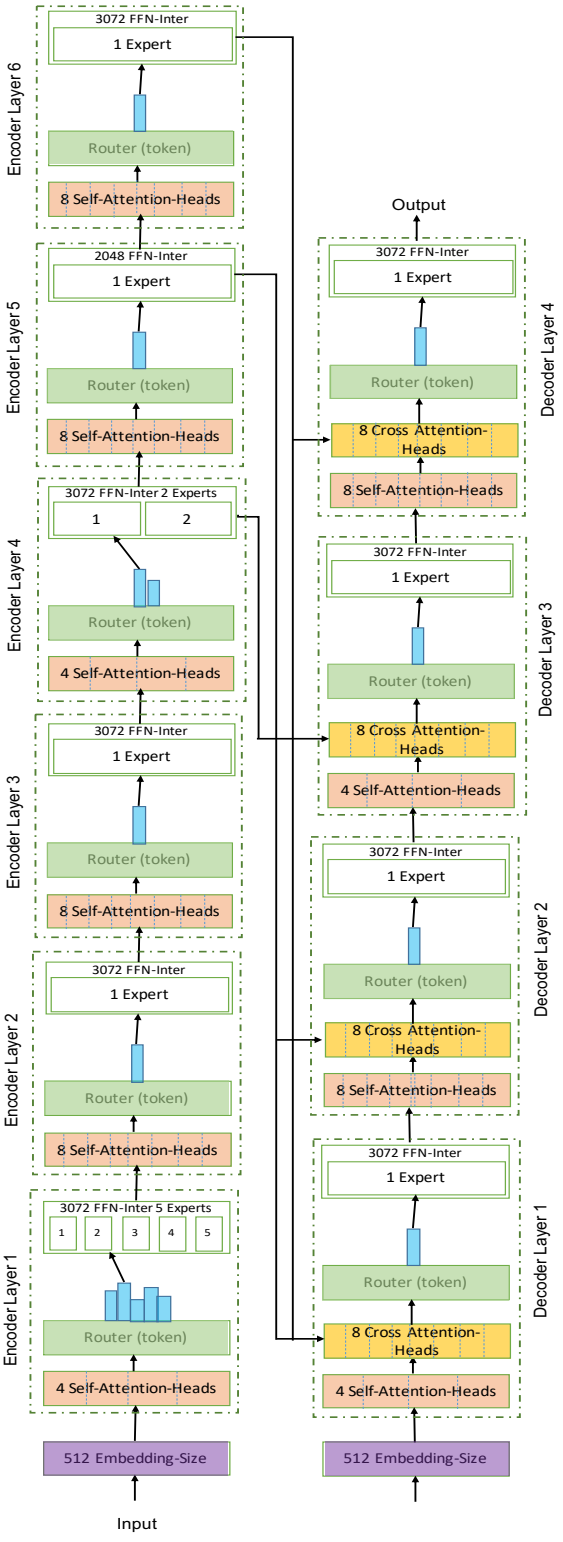


Figure 4: `AutoMoE`-generated architecture for WMT'14 En-De.

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

Table 8: 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.

A.3 Evolutionary Search - Algorithm

We present the pseudo code of the evolutionary search algorithm proposed by HAT in Algorithm 1. This algorithm is also adopted by AutoMoE.

Algorithm 1 Evolutionary search algorithm for Neural architecture search.

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

Output: `best-architecture`

$popu \leftarrow$ `num-population` random samples from the search space
for $iter \leftarrow 1$ to `num-iterations` **do**
    `cur-parents` $\leftarrow$ top '`num-parents`' architectures from $popu$ by supernet's validation loss
    `cur-mutate-popu` = $\{\}$
    for $mi \leftarrow 1$ to `num-mutations` **do**
        `cur-mutate-gene` $\leftarrow$ 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` $\cup$ `cur-mutate-gene`
    `cur-crossover-popu` = $\{\}$
    for $ci \leftarrow 1$ to `num-crossover` **do**
        `cur-crossover-gene` $\leftarrow$ crossover two random examples from $popu$
        if `cur-crossover-gene` satisfies `latency-constraint` via `latency-predictor` **then**
            `cur-crossover-popu` $=$ `cur-crossover-popu` $\cup$ `cur-crossover-gene`
    $popu =$ `cur-parents` $\cup$ `cur-mutate-popu` $\cup$ `cur-crossover-popu`
return top architecture from $popu$ by supernet's validation loss

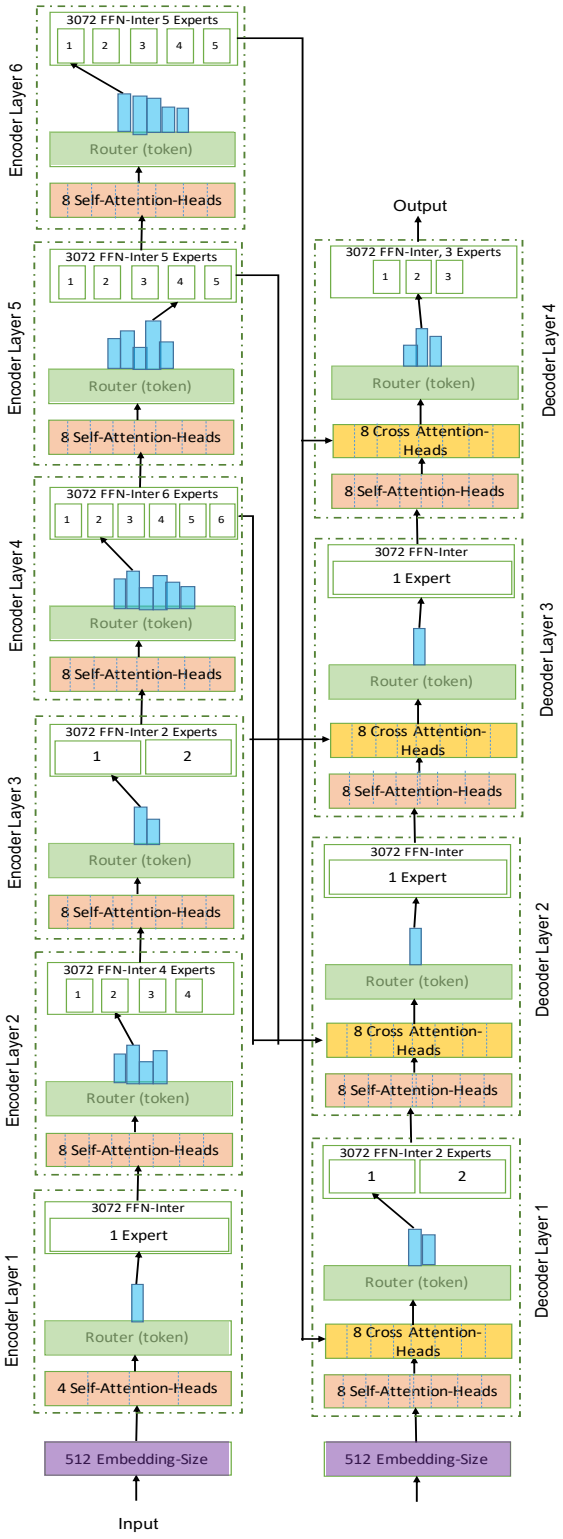


Figure 5: AutoMoE-generated architecture for WMT'14 En-Fr.

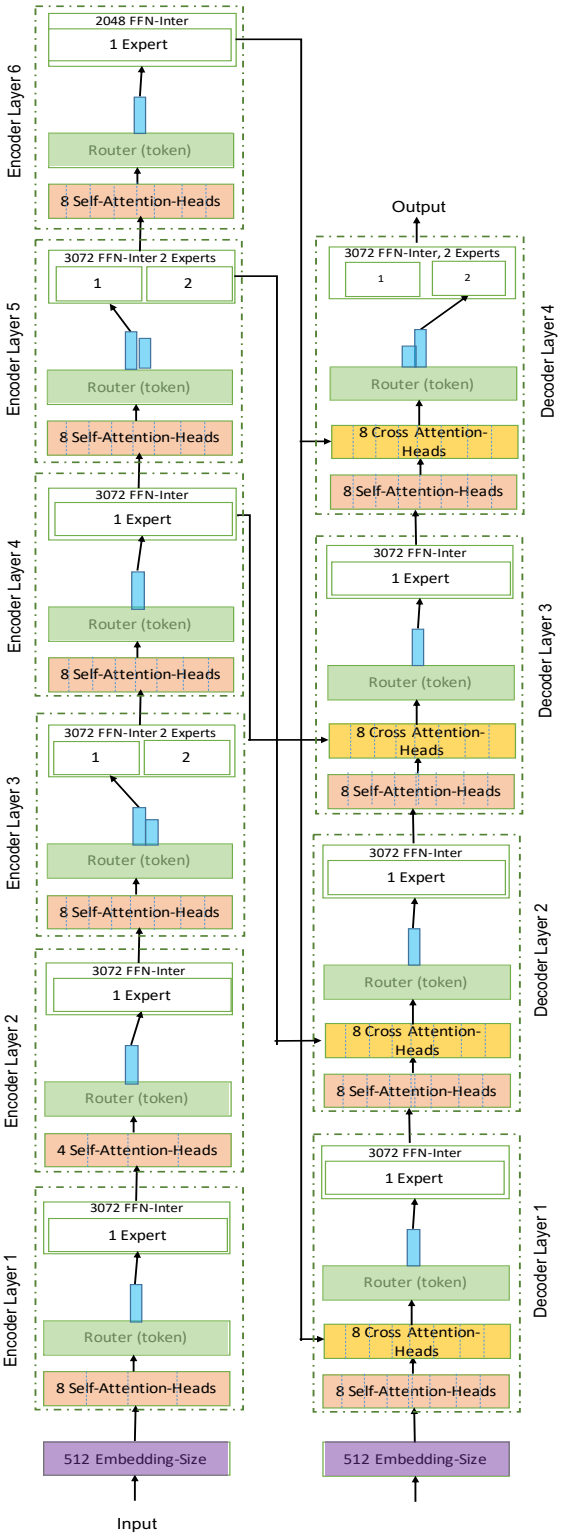


Figure 6: AutoMoE-generated architecture for WMT'19 En-De.